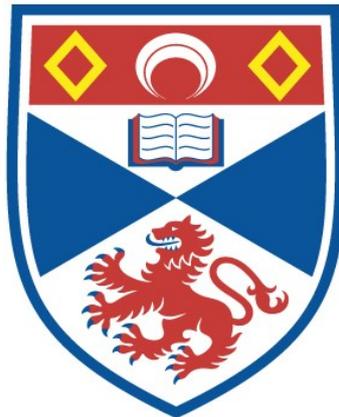


A DISTRIBUTED CONTROL SYSTEM FOR THE ST  
ANDREWS TWIN PHOTOMETRIC TELESCOPE

Richard T. Gears

A Thesis Submitted for the Degree of PhD  
at the  
University of St Andrews



1996

Full metadata for this item is available in  
St Andrews Research Repository  
at:  
<http://research-repository.st-andrews.ac.uk/>

Please use this identifier to cite or link to this item:  
<http://hdl.handle.net/10023/14057>

This item is protected by original copyright

**A Distributed Control System  
for the  
St Andrews Twin Photometric Telescope**

**Richard T. Gears**

**January 1996**

**A thesis submitted to the University of St Andrews in  
accordance with the regulations for admission to the degree of  
Doctor of Philosophy.**



ProQuest Number: 10171192

All rights reserved

INFORMATION TO ALL USERS

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

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



ProQuest 10171192

Published by ProQuest LLC (2017). Copyright of the Dissertation is held by the Author.

All rights reserved.

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

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

Th  
C 2

# Contents

Contents .....	2
Abstract .....	8
Declaration.....	9
Glossary .....	10
Chapter 1 : Introduction .....	13
1.1 : Overview of a Generic Telescope Control System .....	14
1.2 : Review of Telescope Control Systems .....	14
1.3 : Overview of the St Andrews twin photometric telescope .....	21
1.4 : Twin photometric telescope : Status October 1990 .....	24
1.5 : Astronomical projects proposed for the twin photometric telescope.....	26
Chapter 2 : Distributed control system concept .....	29
2.1 : History of process controllers .....	30
2.2 : System architecture .....	34
2.3 : Centralised control architecture .....	34
2.4 : Distributed control architecture .....	36
2.5 : Networks used for distributed architectures .....	43
2.6 : Design of network .....	47
2.7 : Network capacity.....	49
2.8 : Network packet protocol.....	49
Chapter 3 : Design of the distributed control system for the St Andrews twin photometric telescope .....	52
3.1 : Overview .....	53
3.2 : Electrical considerations .....	56
3.3 : Printed circuit board manufacture considerations .....	56
3.4 : Design of microcontroller boards.....	57
3.5 : The CPU board.....	57
3.6 : The power board .....	60
3.7 : I/O board design .....	61
3.8 : Solid state relay design.....	62
3.9 : Pulse width modulated (PWM) DC motor driver board.....	63
3.10 : Board construction.....	64
3.11 : Cube unit .....	64
3.12 : Construction of the sidereal drive .....	65
3.13 : Construction of the dome controller .....	68
3.14 : Construction of focus motors .....	71
Chapter 4 : System software .....	72
4.1 : Software overview .....	73
4.2 : Low level microcontroller software for process control.....	73
4.3 : Software structure.....	76
4.4 : Receiver interrupt : rx_int .....	76
4.5 : Encoder interrupt routine : ctc_int.....	78
4.6 : Main loop routines.....	82
4.7 : Transmit monitor : print_text .....	82
4.8 : Move monitor : movemon.....	84

4.9 : SIO overview .....	86
4.10 : Receive monitor : rx_mon .....	86
4.11 : Commands used in remote microcontrollers.....	87
4.12 : Manual / automatic move .....	89
4.13 : Software operation of the actuators .....	89
4.14 : R.A. reference : slewon and slewoff .....	89
4.15 : Clamp routines : clampon, clampoff .....	90
4.16 : DEC. reference : slewon / slewoff.....	92
4.17 : Other microcontroller axis routines .....	95
4.18 : Memory utilities .....	96
4.19 : Reset commands .....	96
4.20 : PC clone control program.....	97
4.21 : Design of the telescope control program.....	100
4.22 : Module TCLQUICK .....	103
4.23: Module TCLMAN .....	105
4.24 : Module TCLLINK .....	107
4.25 : Module TCLCTRL.....	114
Chapter 5 : Conclusion .....	120
5.1 : Overview .....	121
5.2 : Board design analysis.....	121
5.3 : Board software analysis .....	123
5.4 : Tertiary system analysis.....	124
5.5 : Telescope control analysis .....	124
5.6 : Windows modules.....	125
5.7 : Concluding remarks .....	126
References .....	128
Acknowledgements .....	133
Appendix A : Source code for GAL devices .....	134
A.1 : CPU memory and I/O logic decoder .....	135
A.2 : Motor enable and direction encoder .....	138
A.3 : Network controller and driver.....	140
Appendix B : Files used in the compilation of the microcontroller software .....	143
B.1 : Include file for character definition .....	144
B.2 : Include file for mapping of external I/O .....	145
B.3 : Include file for mapping internal memory variables.....	146
B.4 : Include file for statement field definition (prototype only) .....	148
B.5 : Include file for Z84013/015 internal peripheral mapping.....	149
B.6 : Include file to set up internal peripherals.....	150
B.7 : Include file to configure SCMA assembler .....	153
Appendix C : Z80 Source Code .....	154
C.1 : Source code for R.A. axis microcontroller .....	155
C.2 : Source code for DEC. axis microcontroller .....	172
C.3 : Source code for OFFSET axis microcontrollers .....	189
C.4 : Source code for DOME axis microcontroller .....	204
Appendix D : C source files for the operator interface modules.....	220
D.1 : TCLCTRL : Make file .....	221
D.2 : TCLCTRL : Definition file .....	222

D.3 : TCLCTRL : Include file .....	223
D.4 : TCLCTRL : Source file .....	224
D.5 : TCLCTRL : Resource script.....	240
D.6 : TCLLINK : Make file.....	241
D.7 : TCLLINK : Definition file .....	242
D.8 : TCLLINK : Include file.....	243
D.9 : TCLLINK : Source file .....	244
D.10 : TCLLINK : Resource script.....	250
D.11 : TCLQUICK : Make file .....	251
D.12 : TCLQUICK : Definition file.....	252
D.13 : TCLQUICK : Include file .....	253
D.14 : TCLQUICK : Source file.....	254
D.15 : TCLQUICK : Resource script .....	258
D.16 : TCLMAN : Make file.....	259
D.17 : TCLMAN : Definition file .....	260
D.18 : TCLMAN : Include file .....	261
D.19 : TCLMAN : Source file .....	262
D.20 : TCLMAN : Resource script.....	268
Appendix E : Hardware data.....	269
E.1 : Device list .....	269
E.2 : Spine connector .....	272
E.3 : CPU memory and I/O logic : GAL pinout.....	276
E.4 : Motor limit : GAL pinout .....	276
E.5 : Bus control : GAL pinout .....	276
E.6 : Bus control : Terminal block connectors.....	277
E.7 : Bus control : RS232 pinout .....	277
E.8 : Bus control : Display front end.....	278
E.9 : DC axial motors .....	278
E.10 : Offset motor wiring .....	278
E.11 : Reference R.A. : Slew motor .....	278
E.12 : R.A. Clamp.....	279
E.13 : Reference R.A. : Clutch unit.....	280
E.14 : Sidereal drive .....	280
E.15 : Mains power back connectors .....	281
E.16 : Fuse panel back connectors.....	282
E.17 : Low voltage shelf .....	283
E.18 : R.A. reference output .....	284
E.19 : R.A. reference inputs.....	284
E.20 : DEC. reference output .....	285
E.21 : DEC. reference inputs.....	285
E.22 : R.A. offset outputs.....	286
E.23 : R.A. offset inputs .....	286
E.24 : DEC. offset outputs.....	286
E.25 : DEC. offset inputs .....	287
E.26 : Dome outputs .....	287
E.27 : Dome inputs.....	287
E.28 : Encoders : Reference R.A. / DEC. coarse / fine.....	288

E.29 : Dome hydraulic pump .....	288
E.30 : Dome valves .....	289
E.31 : Interface units (general) : Mutibloc connectors.....	290
E.32 : Manual override : Connectors.....	291
E.33 : Board connectors .....	291
Appendix F : Schematics .....	294
F.1 : Mains input, power switches and 3 phase isolator transformer .....	295
F.2 : Oil pump and dome shutter power supply.....	296
F.3 : 3 Phase motor power supply distribution.....	297
F.4 : R.A. slew and R.A. clamp switchgear .....	298
F.5 : DEC. slew and DEC. clamp switchgear .....	299
F.6 : 1 Phase supply distribution.....	300
F.7 : Low voltage transformers and distribution.....	301
F.8 : Low voltage spine distribution.....	302
F.9 : Reference DEC. fine motor .....	303
F.10 : Offset R.A. fine motor .....	304
F.11 : Offset DEC. fine motor .....	305
F.12 : Reference R.A. limits .....	306
F.13 : Reference DEC. limits.....	307
F.14 : Offset R.A. limits and encoder.....	308
F.15 : Offset DEC. limits and encoder.....	309
F.16 : Dome rotation opto isolator and controller.....	310
F.17 : Sidereal motor driver, R.A. sidereal opto isolator, driver and R.A. clutch .....	311
F.18 : Hydraulic pump 3 phase supply and switchgear .....	312
F.19 : Reference R.A. coarse and fine encoders.....	313
F.20 : Reference DEC. coarse and fine encoders.....	314
F.21 : Reference and offset focus motors and switchgear .....	315
F.22 : Microcontroller power and communications .....	316
F.23 : Microcontroller power supply and monitor display .....	317
F.24 : 3 Phase solid state relay unit.....	318
F.25 : Pulse width modulated motor driver.....	319
F.26 : Sidereal and dome opto isolator .....	320
F.27 : Dome encoder isolator.....	321
F.28 : CPU power board .....	322
F.29 : 3 Phase clamp overload sensor.....	323
F.30 : 3 Phase mechanical pump contactor .....	324
F.31 : 3 Phase pump solid state relay contactor.....	325
F.32 : Manual switch unit.....	326
F.33 : I/O board (output driver section).....	327
F.34 : I/O board (input buffer section) .....	328
F.35 : CPU board (processor section) .....	329
F.36 : CPU board (memory section) .....	330
F.37 : Proposed fibre optic transceiver unit .....	331
F.38 : Transceiver used.....	332
F.39 : Network controller .....	333

## Illustrations

### Flowcharts

Flowchart 1 : Network transmission protocol .....	48
Flowchart 2 : Flowchart of typical actuator direction change .....	63
Flowchart 3 : Receive interrupt routine .....	78
Flowchart 4 : Encoder merging routine .....	81
Flowchart 5 : Main microcontroller loop .....	82
Flowchart 6 : Loading of transmission buffer .....	83
Flowchart 7 : Transmission check.....	84
Flowchart 8 : Move monitor routine.....	85
Flowchart 9 : R.A. reference slew routine.....	90
Flowchart 10 : R.A. reference clamp routines .....	93
Flowchart 11 : DEC. axis slewing routines.....	94
Flowchart 12 : Windows 3.1 DDE link map .....	102
Flowchart 13 : TCELLINK routines.....	107
Flowchart 14 : Sequential interpretation of message acknowledgement .....	118

### Diagrams

Diagram 1 : Twin photometric telescope.....	21
Diagram 2 : Twin photometric telescope photometer head .....	23
Diagram 3 : Proposed search diagram.....	28
Diagram 4 : Centralised control system .....	36
Diagram 5 : Distributed control system .....	38
Diagram 6 : CPU Demand and data transfer bandwidth from a centralised control system .....	41
Diagram 7 : CPU Demand and data transfer bandwidth from a distributed control system .....	42
Diagram 8 : Network node controller block diagram .....	46
Diagram 9 : Arbitration state .....	48
Diagram 10 : IMS T414 transputer block diagram .....	58
Diagram 11 : Block diagram of CPU board .....	59
Diagram 12 : I/O block diagram .....	62
Diagram 13 : Block diagram of sidereal drive unit.....	66
Diagram 14 : Dome rotation schematic .....	69
Diagram 15 : Encoder error from backlash.....	79
Diagram 16 : Modified encoder read routine.....	80
Diagram 17 : Network view of command processing.....	87
Diagram 18 : Clamp unit.....	91
Diagram 19 : Reference, Offset and Dome push-button layouts.....	106
Diagram 20 : TCLCTRL Initial display.....	116
Diagram 21 : A TCLCTRL operational display.....	116

## **Tables**

Table 1 : System architecture characteristics .....	40
Table 2 : Network types .....	44
Table 3 : Network lines.....	47
Table 4 : Network packet protocol used on the twin photometric telescope .....	49
Table 5 : Network addresses .....	50
Table 6 : Relationship between mode selected and shaft rotation.....	67
Table 7 : Encoder detail .....	79
Table 8 : Microcontroller commands .....	88
Table 9 : Microcontroller actuator parameters .....	88
Table 10 : Actuators used for each axis .....	89
Table 11 : Available memory addresses.....	95
Table 12 : Common operating systems.....	97
Table 13 : TCLQUICK error codes.....	104
Table 14 : Available speed modes for different axes .....	105
Table 15 : TCLLINK title text.....	108
Table 16 : TCLLINK error codes .....	109
Table 17 : TCLCTRL commands .....	113
Table 18 : TCLCTRL push buttons.....	115
Table 19 : Variable colour status.....	115
Table 20 : TCLCTRL error messages .....	119

## **Abstract**

Many astronomers require large amounts of observational data to solve astrophysical problems and to validate theoretical hypotheses. It is therefore imperative that both the observer and telescope work efficiently, maximising data collection whilst minimising object selection and acquisition time. One method in which this can be achieved is through telescope automation.

The advent of cheap integrated process controllers enables the system designer to realise novel control system architectures which were previously prohibitive to all but the largest of sites.

This thesis reviews the development of processor based control systems in the astronomical and industrial environment and compares distributed and centralised control system architecture. It describes the design and construction of one such distributed control system for the St Andrews Twin Photometric Telescope.

Keywords : telescopes, automation, microcontroller.

**The Declaration for the Degree of Ph.D.**

I, Richard Gears, hereby certify that this thesis, which is approximately 72300 words in length, has been written by me, that it is the record of work carried out by me and that it has not been submitted in any previous application for a higher degree.

Date 11<sup>th</sup> January 1996..... Signature of candidate .....

I was admitted as a research student under Ordinance No. 12 in October, 1990 and as a candidate for the degree of Ph.D. in October, 1991; the higher study for which this is a record was carried out in the University of St. Andrews between 1990 and 1993.

Date 11<sup>th</sup> January 1996. Signature of candidate .....

I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of Ph.D. in the University of St Andrews and that the candidate is qualified to submit this thesis in application for that degree.

Date 12<sup>th</sup> January 1996 Signature of supervisor .....

In submitting this thesis to the University of St. Andrews I understand that I am giving permission for it to be made available for use in accordance with the regulations of the University Library for the time being in force, subject to any copyright vested in the work not being affected thereby. I also understand that the title and abstract will be published, and that a copy of the work may be made and supplied to any bona fide library or research worker.

Signed ..... Date .....

## Glossary

ATOM	A 16 bit integer array whose entries points to character strings
ATIS	Automatic Telescope Instruction Set
BUS	A generic group of signals connected between two or more structures via a common physical route
C700	C compiler version 7.00. This is the first Microsoft compiler to support the Windows 3.0 / 3.1 operating system
C	Programming language used to create Windows modules
C++	Object oriented version of C
CCW	Counter clockwise rotation
CFX	Control file executor
CLK	Clock
CPU	Central processing unit, e.g. microcontroller IC
CR	Carriage return (0x0D) or capacitor resistor timing circuit
CRC	Cyclic redundancy check. A polynomial generated by the SIO to enabling error checking
CTS	RS232 Protocol : clear to send
CW	Clockwise rotation
DEC	Declination
DCD	RS232 Protocol : data carrier detect
DCS	Manufacturer of IBM PC clone used in TPT
DDE	Dynamic data exchange
DOS	Disk operating system. Standard text based operating system for IBM PC clones; current version is 6.0
DTR	RS232 Protocol : data terminal ready
DX	Non multiplexed data bus
EPROM	Erasable programmable read only memory. An IC that stores a program for use by a microcontroller
FET	Field effect transistor
GAL	Generic array logic device: An IC that can be programmed to emulate logic functions
GND	Local ground
GUI	Graphical user interface
IBM	International business machines. Manufacturer and standard specifier of PC computers
IC	Integrated circuit
IDC	Indirect connector
INT	Interrupt, usually related to a physical line which interrupts the CPU
INTEL	Manufacturer of 80x86 processor ICs
I/O	Input / output
JEDDEC	Jedec code (kernel format) : used in programming GAL devices
LAN	Local area network
LED	Light emitting diode
LSB	Least significant bit

MAX386	A memory management driver from Qualitas required by Microsoft's C700 compiler
MB	Mega byte or mega baud
MSB	Most significant bit
NTT	New technology telescope
/nod	Link switch in Microsoft's C700 compiler that ignores default libraries named in object files
OLE	Object linking and embedding
PC	Personal computer, generic name for IBM type clones
PCB	Printed circuit board
PIO	Parallel input / output unit
PLC	Process logic controller
PWB	Microsoft's integrated environment for the C700 compiler
R.A.	Right ascension
RAM	Random access memory
RDU	Remote driver unit. Used to describe a microcontroller unit
RFI	Radio Frequency Interference
RISC	Reduced instruction set computer
RST	Reset, usually applied to the physical line to reset a CPU
RX	Receive
SCMA	Smart communications macro assembler
SCSI	Small computer systems interface
SDLC	Synchronous data link control. An IBM propriety packet protocol
SIO	Serial input output unit. This allows the computer communicate to the outside world, usually using RS232 protocol
SID	Short for sidereal
SMART	Manufacturer of test equipment
STACK	A type of data storage, usually very fast and memory efficient
SVGA	Super versatile graphics adapter (1028*760 pixels, 256 plane)
SX	Multiplexed data bus
SYNC	A timing pulse to synchronise data transfer
TCL	Telescope control language
TI	Texas instruments
TPT	Twin photometric telescope
TTL	Transistor transistor logic
TX	Transmit
TXREQ	The network access arbitration line
TXRX	The network data transfer line
VCC	Voltage collector collector (power supply +)
VGA	Versatile graphics adapter (640*480, 16 plane)
WDT	Watch dog timer. Used to detect fatal program crashes
WHT	William Herschel Telescope
WIMP	Windows, icons, mouse pointing technology
Z80	The processor family name of which the microcontrollers are the latest generation
.BCL	Filename extension : Batch file for high level commands

<b>.COM</b>	Filename extension : Command program for PC computers
<b>.CUR</b>	Filename extension : Cursor file. Generated by the image editor
<b>.DEF</b>	Filename extension : Definition file for C700 compiler
<b>.EXE</b>	Filename extension : Executable code for DOS and Windows
<b>.H</b>	Filename extension : Include file for definitions
<b>.ICO</b>	Filename extension : Icon file. Generated by the image editor
<b>.JED</b>	Filename extension : Jedec (kernel format) file used by XP6005
<b>.LST</b>	Filename extension : List file used to debug compiled programs
<b>.MAP</b>	Filename extension : Memory map of used locations
<b>.MCL</b>	Filename extension : Label table for module variables
<b>.OBJ</b>	Filename extension : Compiled code which has not been linked
<b>.PRE</b>	Filename extension : Pre-processing directives for the compiler
<b>.PLD</b>	Filename extension : Programmable logic source file: Used as the source code filename extension for GAL devices
<b>.REL</b>	Filename extension : Relative code (source or object)
<b>.SCL</b>	Filename extension : Default configuration file for modules
<b>.SYM</b>	Filename extension : Label names and hexadecimal representation
<b>.RC</b>	Filename extension : Resource script for C700 compiler. Holds data for popup menus, hot keys etc.
<b>.RES</b>	Filename extension : Compiled resource script
<b>.VCL</b>	Filename extension : Volatile disk based stack for modules

## **Chapter 1 : Introduction**

This chapter gives an overview of a generic telescope control system. It reviews the history of telescope control and automation systems with reference to various telescope facilities. It outlines projects where telescope automation is required or preferable. It finally describes the construction and operation of the twin photometric telescope (TPT) and its operational status during October 1990 and October 1991.

## **1.1 : Overview of a Generic Telescope Control System**

A control system is an object that provides an output or response for a given input or stimulus (1). With reference to an astronomical telescope, the control system is the collection of processes that point the telescope to the correct co-ordinates, collect the data from the detector, track the dome to the correct position and much more.

## **1.2 : Review of Telescope Control Systems**

The most popular telescope control system ever designed was the human. A single person can move the telescope to the correct co-ordinates, acquire the object and record the received data onto paper. The flexibility and adeptness of the human in acquiring new skills has yet to be equalled by any artificial invention. The Achilles' heel of the human operator is the inability to be consistent and accurate over periods of time. Images recorded by hand are subject to artistic interpretation. Numerical values can be recorded incorrectly. Integration times may slip from lack of concentration. Two star fields may look similar but one may be incorrect. Much work has been done in the physiological field to try to minimise the inconsistencies of the human operator. However with recent advances in electronic design autonomous solutions can provide a level of consistency which far surpasses that of the human.

The discovery of the photoelectric effect in the 1920's by Einstein provided the astronomer with a detector capable of quantifying the received brightness of a source by recording the output current pulses. The recording could be achieved by

counting individual current pulses for low photon rates or by measuring the integrated charge level across a capacitor.

The first astronomical observations using a photomultiplier based detector required a person to record data from the voltmeter or pulse counter to the log book. While this simple but unreliable method was suitable for integration times exceeding a few seconds, new astronomical techniques called for sub-second integration times and hence automated solutions.

For example, using occultation techniques it is possible to determine the presence of double stars and to calculate the angular diameter and precise position of objects. This is achieved using a short integration time and an accurate time stamp. By correlating the discontinuities in the detector output with the stellar motion, absolute and relative angular displacements can be determined. The equipment required to measure such angular diameters include a microprocessor controlled tape recorder with a time reference of 1 millisecond accuracy (2).

Other techniques requiring a reduction in integration time centre around minimising atmospheric effects which decrease the resolution and increase the variability in the received light intensity.

One example of such an atmospheric effect is high, thin cirrus cloud. The time scale over which this atmospheric variability occurs is in the order of tens of seconds (4). One method of reducing this variability is to use narrow band filters which, if chosen to lie between the strongest of the atmospheric absorption lines, can be relatively insensitive to the atmospheric fluctuations. Another method is to integrate for time scales less than the atmospheric variations. A high speed electronic filter changer was designed to make short integration times possible and hence to obtain worthwhile photometric results.

Other methods employ dual channel differential photometers which record both the test and nearby reference star simultaneously. Knowledge of the reference

magnitude and observed test and reference magnitudes enabled the astronomer to minimise the atmospheric variations (3,5).

Many detector controllers and recording devices for photometry use have been constructed. These have usually been based on a centralised computer photometer acquisition system designed using commercial computers (Apple II (6)), custom built computers (8085 processors (2)) or large scale minicomputers (Raytheon (7)).

The use of photometers to detect stars was not restricted to the measurement of the star magnitude. Guidance systems using photomultipliers, TV and CCD cameras are used to accurately track an object during the observation run. Techniques such as a rotating slit aperture, which illuminates sectors of the sky, or a camera with an image processing package can automatically steer the telescope by adjusting the drives for all of the telescope axes (8, 9).

As the operator was confined to the control room, tertiary automation systems were designed to support the observation. Telescope domes could be aligned to the position of the telescope (10), data could be reduced interactively to show the quality of the observing run, and more importantly to show if an error has occurred. This data display can inform of problems including dome or shutter masking of the object, the presence of cloud cover or detector faults (11).

Integration of all the control systems into one collective unit was first experimented with in the 1960's by Code (15). This PDP-8 controlled telescope examined a set of standard stars to measure the extinction coefficients relieving larger telescopes of this repetitive task.

Many projects are suited to complete automation. Automation of telescopes by Treffers (12), Stirling et al (13), Linnel (7) and Colgate (14) enabled large scale searches for supernovae to be undertaken. To detect approximately 1 supernovae a

week in galaxies brighter than 19 magnitude, approximately 1000-4000 galaxies must be observed twice nightly.

Other projects include the monitoring of long period variable stars and those suspected of being variable (15). For example, 60 suspected variable stars can be observed every clear night with one automatic telescope. This observing run is repeated for several years to detect long term variables such as RS CVn binaries and Zeta Aurigae eclipsing variables. To accurately determine the atmospheric extinction coefficient, and thus provide a quality control on the observed variables, all sky photometry of a set of standard stars throughout the observing run is also undertaken (16).

Techniques for telescope automation are as varied as the host telescopes which receive the control system. For small telescopes single processor solutions are favoured, for example the DFM Engineering 68000 VME based system controller unit (17). As the complexity and physical size increases, remote networked stations are introduced to minimise cabling, increase the reliability and enable multitasking at the sensor level (18). These remote controllers were usually common 8/16 bit processors programmed using FORTH (19). These controllers could be integrated into minicomputer I/O racks, such as the CAMAC crate, to provide remote processing units communicating to both the workstation and the sensor (20). Multi-processor control systems provide various topological solutions to networking. Two widely accepted topologies are currently in use: the star and ring topology. An example of a star topology is the Apache point telescope whose central server transmits commands to distributed controllers using dedicated RS232 lines. Likewise, the New Technology Telescope (NTT) control system uses an ethernet connecting a HPA900 minicomputer to 7 VME based outstations (18).

With the expanding use of networks, a highly integrated processor was designed to take advantage of these links. The transputer developed by Inmos is a reduced instruction set computer (RISC) with memory and four high speed serial links

within one package. This device has been developed to control CCD camera waveforms and various detectors (21,22), but can be applied to a wide range of controlling tasks.

Complete automation is not always required. Remote observing, using a voice and data link has been operated at the Royal Observatory, Edinburgh (23) and at Waimea (24) with success. Remote observatories can be located at premium sites (25) with a cost saving over 90% to the astronomer. Scheduling of observing programs can be more efficient as astronomers can reschedule their own programs in the office and when problems arise, expertise is readily available.

To highlight the differing approaches to telescope automation, two telescope facilities are described. The first example is the William Herschel Telescope. This is an example of a large scale, complex telescope system used by visiting astronomers. The second example is the Autoscope Telescopes Observatory system. This is an example of a small scale, technically simple and compact system for small group research.

#### The William Herschel Telescope.

The William Herschel Telescope (WHT) is a 4.2M telescope mounted on an altitude-azimuth platform. It is part of the Roque de los Muchachos Observatory on La Palma in the Canary Islands (26). It saw 'first light' early summer 1987 and started observing in mid 1988. This large telescope is based on a modular, distributed design incorporating many novel approaches to control system design. A suite of over 8 instruments are proposed for the telescope, including a Prime focus camera, faint object spectrograph infrared photometer and various acquisition and guidance units.

The system comprises of three networks. A DECnet network connects together a Workstation (User Interface), VAX 3600 (Telescope computer) and a MicroVAX II

(telescope control computer). The System computer connects to two more networks: a low speed ethernet 'utility network' used for time independent processes and a high speed bus dedicated to retrieving image data from a large external memory.

Device control is achieved using a distributed array of embedded microprocessors placed near to the controlled unit. Each microprocessor unit can communicate to both the sensors, actuators and RS232 line. Each controller consists of a 6809,68008 or 68020 processor from Motorola executing FORTH. The hardware and software installed enable a target unit to be configured for use, rather than compiled, reducing the development costs.

The control of each device is initiated from the user interface. The instruction for the device is passed to the utility network via the system computer. Taps on the network covert the messages to an RS232 standard, which connects to the microprocessor unit. High speed, high volume data from some detectors is passed to the external memory store by dedicated RS422 lines where multiple images can be stored and retrieved asynchronously by the system computer.

The user interface was designed to provide a clear and consistent presentation to all astronomers, supporting both menus and hot keys for operators with different levels of experience (27). Operators can enter commands interactively through the command language ICL or by using batch files in the VAX ADAM environment. Text batch files are written using TPU in advance of the observing night. The batch file is loaded into the system by starting a interpreter called "control file executor" (CFX). This batch file dynamically links label, action and databases files to provide a high level language which is easy for the astronomer to use (28).

A status display with a consistent graphical user interface (GUI) display format is used to show the detector status to the operator, highlighting by colour the devices which require attention.

The Autoscope Corporation.

Autoscope, an automated telescope manufacturer adopts a different approach. Autoscope are vendors for 0.8M and 0.5M robotic telescopes. A centralised control system, mounted in a 19 inch rack unit controls a horseshoe mounted mirror assembly. Up to four detector modules can be selected. These modules range from photomultipliers, photodiodes, CCD cameras, and spectrometers (29). The complete telescope assembly is less than 2M in height.

Control of each telescope can be real time, across a network, or through a scheduler supporting the Automatic Telescope Instruction Set (ATIS). This facility enables the astronomer to create an observing program in the office, send it to the facility for scheduling and to receive the data when the observation has been completed. Boyd and Genet (30) discuss the cost effectiveness of such a facility, where a observational sequence on a mountain top non automatic telescope costs \$17.40 against an automatic telescope cost of \$8.20. DFM Engineering provide a similar centralised control system (17).

Irrespective of the approach observers use to gather data, the complexity of today's instrumentation require an integrated approach to telescope system design.

### 1.3 : Overview of the St Andrews twin photometric telescope

The Twin Photometric Telescope was built by Grubb and Parsons in 1962 (5) for the Royal Observatory, Edinburgh, where it was used for accurate differential photometric measurement of two objects and monitoring of variable stars.

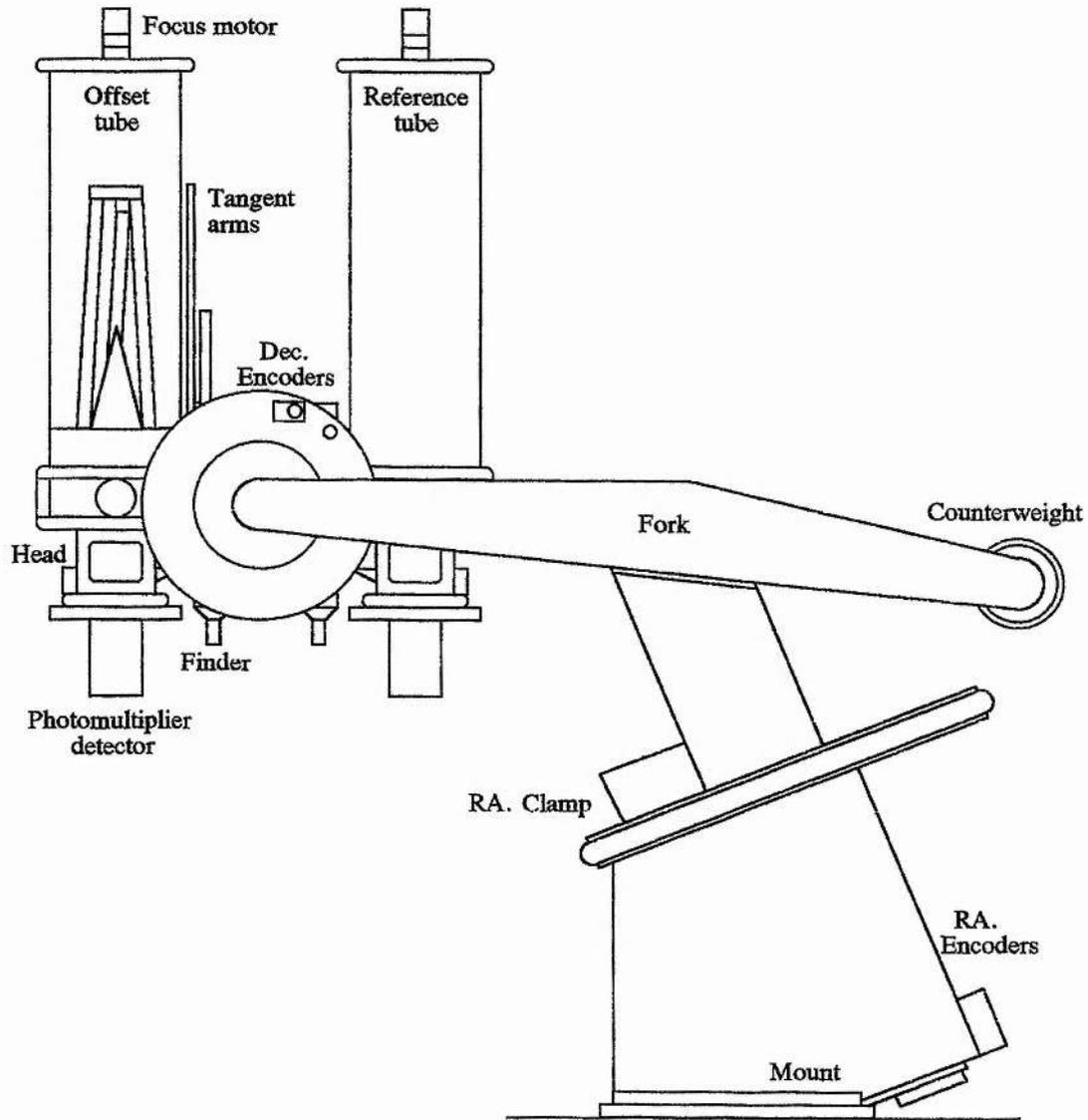


Diagram 1 : Twin photometric telescope

In 1980 the telescope was moved to St Andrews University where some new electronic control systems were installed by David Carr. The telescope is part of the St Andrews Observatory (-0h 2m 48.9s, +56° 20.2') which is situated on the outskirts of the town of St Andrews on the east coast of Scotland. Its low altitude (30ft) and proximity to the North Sea produce extremely variable seeing conditions. Whilst the observing quality of the site is poor it is nevertheless possible to undertake various research programmes such as the photometry of close variable stars (31, 32). Recently the photometric calibration of stars for photographic sky surveys has also been attempted.

The twin photometric telescope (Diagram 1) comprises of two identical 0.4m, f/15 Ritchey Chrétien telescopes mounted on a common equatorial fork. Each telescope is set visually using two finders placed alongside the main tubes. The offset tube is able to point up to 5 degrees distant from the reference tube in both right ascension and declination directions. Differential flexure in both telescopes is known to be present at high zenith angles but is not a serious problem. The photometer head and detector units are mounted directly onto the rear of each mirror cell. The photometer units are controlled by a BBC computer whose keyboard and display is located in a warm room.

Diagram 2 shows the internal construction of the photometer head, detector and light path to both detector and CCD camera. The photometer head (33) comprises of two, six position wheels for the aperture and filter units. Each wheel is positioned by a stepper motor using a 3 bit grey code to identify the object selected. The aperture wheel has an additional opto switch mounted on a gear train for higher positioning accuracy. A CCD camera provides guidance using the off-axis rays reflected from the highly polished aperture disk.

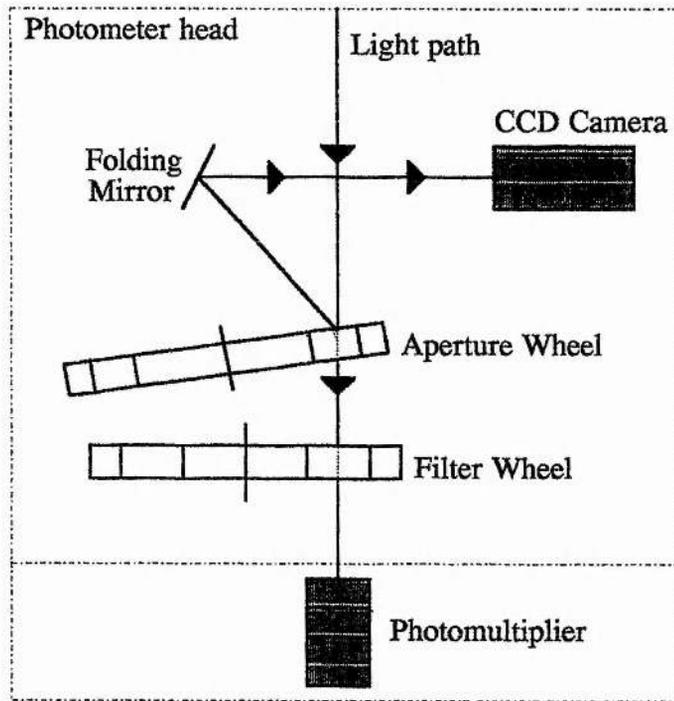


Diagram 2 : Twin photometric telescope photometer head

The photomultiplier detectors are housed in Peltier cooled cold-boxes which incorporate window heaters to prevent condensation. The photomultiplier outputs are discriminated, amplified and counted by dual 24 bit counters connected to the 1Mhz bus of a BBC 'B' data logging computer. Integration times are controlled by a counter, clocked by an accurate time pulse from the observatory's LSI 11 computer. At the end of an observing night, data is down-loaded by Kermit software to the university STARLINK node for subsequent reduction.

A control desk situated in the telescope warm room housed all switchgear and relay units. All operator controls and consoles were located on the front of the desk, giving the user access to most of the telescope functions from a warm environment. Control of each axis was implemented by 'Magslip' synchro resolvers (34). Magslip encoders displayed the present position of all the telescope axes on rotary dials. The required position was set using another Magslip encoder producing an error signal

between the actual and required positions. This error signal was minimised by feeding both the amplitude and phase of the error signal to the axis motor logic to effect a move towards equilibrium. Whilst this feedback algorithm was rudimentary, the ratio of axial inertial torque to output driving shaft torque was low. It can be shown (1) that the solution to this transfer function produces little overshoot and so the telescope required little compensation.

#### **1.4 : Twin photometric telescope : Status October 1990**

The Twin photometric telescope as described above was used for variable star research during the winter of 1990 / 1991 by Steve Bell and Don Pollaco. Insignificant data was realised from this observing season owing to control and data acquisition faults. In parallel with the current system design for the Ph.D. project, time was expended on maintaining the serviceability of the existing control system.

These faults included:

- Repeated failure of low voltage cabling

Various parts of the control system were disabled by the incineration of cabling which distributed the low voltage DC to the console and telescope. The faults were located to various oxidised connector tags which caused high impedance links. The low voltage distribution cabling was replaced.

- Op amp drift

Drift in the Magslip error amplifier caused the telescope to oscillate in R.A. at high hour angles. This was rectified by increasing the error trigger voltage which increased the slewing hysteresis.

- Photomultiplier trigger level

The photomultiplier amplifier trigger level was set too high in one tube, rejecting valid pulses. This was adjusted to pass all pulses above the valid height.

- Photometer counter bit drop out

Various bits in the photometer counter section dropped out, producing very low count rates. This was tracked to a fault in the wire wrapping and was rectified.

During the observing season 91 / 92, observations of the variable RZ Cas by Pierre Maxted (40) and Schmidt plate calibrations by Paul O'Neill (35) using the new EEV camera unit and photometer head caused more system faults.

- EEV Camera unit

The EEV camera unit suffered from condensation problems and had to be returned to the manufacturers for modification.

- CUBE photometer head controller

The CUBE BBC controller developed an intermittent hard reset which corrupted the photometer head configuration during telescope operation. The fault was eventually traced to the joint effect of a low logic supply voltage and an incorrectly set reset divider network. The divider chain was modified for the lower supply.

The photometer head program was stored in battery backed random access memory (RAM). Following infrequent telescope use the stored program was found to be corrupted. This indicated that the RAM battery was not holding charge.

The IDC cables which connected the power to both the stepper motors and logic inputs were found to be of inadequate wire gauge thereby producing a voltage potential across their length. This voltage potential decreased the stepper motor power level and caused the received logic levels to lie in the invalid region of the input GAL stepper controller. Higher gauge wire of lower resistance was installed to the stepper motor supply. TTL to CMOS buffers were inserted at the photometer head to increase the logic level voltage range and to isolate the cable from the internal electronics.

- Broken / worn Magslips

Some of the Magslips had faults, ranging from pointer mis-alignment to complete failures. In the latter half of the observing season the reference telescope positioning was extremely unreliable and required checking manually with the hour angle, situated on the base of the telescope fork, and visually with reference to a known constellation. Pointing the offset telescope was carried out entirely by eye.

- Worn control system hardware

Most limit switches were corroded and failed to stop the telescope when activated. Many of the status indicators were inoperative.

Electrical interference from the hydraulic pump frequently corrupted discrete digital devices including the photometer counters and handsets.

It was clear that a major overhaul of the twin photometric telescope was required.

### **1.5 : Astronomical projects proposed for the twin photometric telescope**

The twin telescope is used on a nightly basis to observe known and suspected variables. Alongside these standard observing runs, another project is proposed.

The Mt. Palomar all sky survey is a major tool for the astronomer in identification and classification of stellar objects. The survey consists of 1146 plates each covering a  $6^{\circ} \times 6^{\circ}$  field. The variation in processing of each plate brought about the requirement for plate to plate calibration. An on-going project proposed by Dr. P. W. Hill with preliminary testing by Paul O'Neill involved the use of the twin photometric telescope to identify and measure four objects present in adjacent plates (Diagram 3). The major problem with the project was the large amount of observing time required. One plate observation consisting of five objects integrated in both B and V filters for 3 minutes per observation gives an observing time of 30 minutes.

Calibration for 50% of the sky would therefore require approximately 170 hours, excluding telescope pointing and acquisition (35).

To minimise these two operations it was proposed to use an automated object acquisition algorithm, correlating a reference map stored in CDROM to a visual image from the telescope CCD cameras. The return value would control the setting of the telescope.

The aim of the project was to observe a large number of stars in as short a time period as possible. This could be achieved by maximising the observing time and minimising the acquisition time. To achieve this, all functions of the telescope (movement of the dome and each axis) should be optimised, operating in parallel with each other. The telescope pointing accuracy should be increased to minimise the setting time required. The system had to be reliable and accept observing programs in the form of batch files to minimise the time taken to enter data at the telescope.

Whilst it was possible to build an automatic acquisition system based around the CCD and current control system, either the faults described earlier had to be rectified or the entire control system had to be upgraded. Taking into consideration the current failure rate of the present system and the estimated performance and operator flexibility of a distributed control system, the second option was chosen. This forms the basis of the project described in this thesis - the assessment of the existing twin telescope and its control system, the design of a robust and intelligent replacement system, the implementation of this enhanced system and finally its testing.

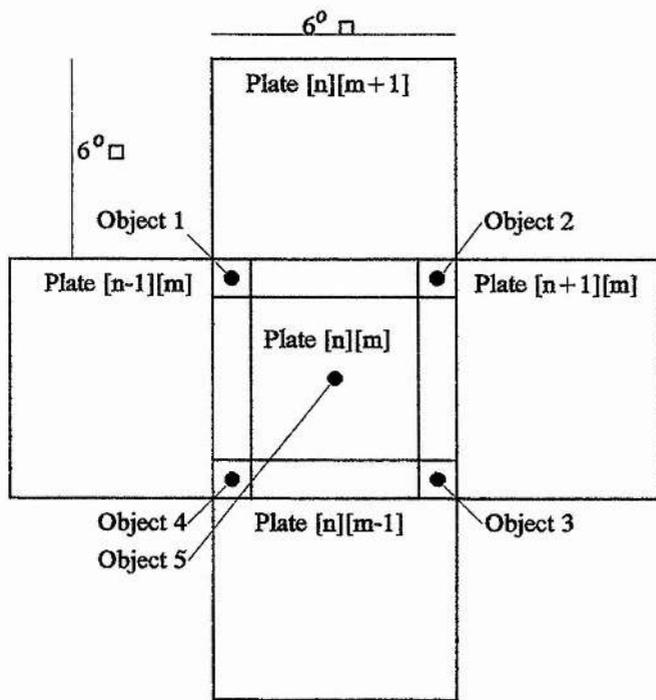


Diagram 3 : Proposed search diagram

## **Chapter 2 : Distributed control system concept**

This chapter outlines the recent developments in control systems. It describes the different types of control system architectures that are available and compares the centralised and distributed control system architectures. Finally it outlines the implementation of both the network and protocol used in this project.

## **2.1 : History of process controllers**

In the industrial workplace, the demand for mass-produced items created a requirement for intelligent autonomous controllers which could work throughout the day and night with little or no human intervention.

The first form of process control used banks of relays and mechanical timers as logic gates. Groups of these logic gates were combined to perform a control algorithm programmed by the designer using a ladder diagram. This controller was connected to various actuators and sensors around the site, providing a rudimentary autonomous control system. Whilst each control algorithm was relatively simple, extensive switchgear was required to implement the total process. This was due to the limited switching capacity of the relays, each switching up to 10 isolated poles per relay (41).

Implementation of relay logic in industrial plants revealed serious limitations:

- High initial cost
- High rate of mechanical and electrical wear due to 'hot' switching
- Restrictive potential for expansion due to the direct wiring of relays and the time required to produce a new ladder diagram
- High power consumption
- Large physical size
- Limitation to basic logic functions

During the late 1950s digital designers realised that much of the relay based logic could be replaced using digital techniques. Allen-Bradley responded in 1959 with a unit called the "PDQ". This was an elementary logic controller that replaced the old relay logic banks with a semiconductor based processing unit.

Continuing processor development in the late 1960's and early 1970's produced the basic programmable logic controller (PLC) that is in use today. Gould (previously Modicon, previously Bedford Associates), Allen-Bradley and Texas Instruments developed dedicated modular industrial processor cards with a kernel program to interpret the ladder diagrams and actuate relays. These cards were slotted onto a rack alongside application specific modules (42) such as digital and analogue input and output buffers, counters, timers and PID (Proportional Integral, Derivative) units.

Initial developments from this PLC design focused on improving the program scan speed and diversifying potential applications. While the Allen-Bradley PLC-3 used dual processors to relieve the central processing unit (CPU) under heavy loads, other PLC manufacturers relied on the semiconductor industry to increase the processor instruction speed and also for end user designers to use conditional ladder programming to enable large processes to be successfully executed.

The varied requirements of industry have led to an increasing number of new applications for the PLC, many of which the original designers could not have conceived.

An example of this expansion is the 5TL PLC, the mainstay process controller of Texas Instruments, one of the original PLC designers. The 5TL was first launched in 1975 and is still being produced today. The 5TL consists of a rack based processor with software counters, timers and I/O facilities. Whilst this is adequate for process line control, the basic system cannot use analogue position and velocity sensors to model transfer functions.

An example of a transfer function is the equation governing the dynamics of a robotic arm. The arm response is a conglomeration of rotational, transitional and electromechanical systems and can be modelled by a composite transfer function (1). This transfer function can control the arm with more accuracy in both speed and position control than direct actuator control from discrete states generated by

limit sensors. This is due to the introduction of a continuous, closed feedback loop which can record and account for changes in environment. Early control units that interfaced the PLC to such an arm using this transfer function used either an analogue computer (49) or a dedicated digital computer (46), purpose built by the end user. Both methods required substantial interfacing and programming, increasing the development time. To provide a standard interface unit Texas Instruments built the PID unit (41) providing the application designer with a standard method of programming transfer functions for each application. Presently, state of the art PLC's use functional block programming on high speed processors to implement PID, ramp, S-ramp and more transfer functions.

At the same time as these developments were being made with PLCs, advances were made within the general computing field. Computer communication techniques provided the operator with access to information databases and control of peripherals. Serial communication, such as the widely used RS232 standard, gave the user low speed point to point data transfer for modem links, linking two remote computers together. Parallel communication gave the operator a short, fast point to point data transfer for use with graphical display units and printers. Finally, Local Area Networks (LAN's) provided the user with high speed, multi-node communications enabling access to many databases, distributed workstations and control of processes from remote terminals. PLC manufacturers realised that they could adopt many of these applications by installing a communications link to the PLC. These links were primarily implemented using I/O cards, superseded by a dedicated communications card. Applications included status display, data logging, control and printing.

As higher capacity, more complex computer networks were constructed (50) it was realised that a fully networked site could integrate all processes, from accounts and

stores to robots and process lines (41). Full integration was completed by storing all designs on databases, allowing all nodes access to the relevant design information (43). The ultimate goal of the systems engineer is to provide a high level of automation, enabling the manufactured quantity to quickly respond to changes in market conditions.

As the production configuration evolved, each PLC manufacturer adapted by inserting a new card into the reliable and proven system providing the process control manager with a greater sense of security than using a new custom made controller. Modern large industrial plants now use numerous distributed PLCs monitored by a workstation and overseen by just a few technicians.

Total integration requires the monitor and control of a large number of sensors and actuators, more than can be serviced by a single PLC. The logical progression would appear to be a scaling of the PLC module to encompass all external devices, but in practice multiple PLC modules are used, each processing a fraction of the total tasks. An analogy of this is the transputer. The computer designer aims at a higher processing power that would logically require a faster processor. While many designers choose to scale the processor size accordingly, 'Next' used a distributed architecture using multiple transputers to achieve this aim.

Between the plant and computer environments there is a little explored area where a process consisting of multiple fragmented sub-processes interfacing over a wide geographical area must be automated but is too complex for a single controller.

The twin telescope is such a device. The fragmented processes are its axes. Each axis requires high speed data acquisition and control from a physically wide footprint whilst the physical sensor and actuator sizes are small.

It was proposed to embed into the twin photometric telescope an architecture based on a distributed industrial PLC scenario. Each actuator and sensor was to be

managed by its own microcontroller. The control implementation was embedded in each microcontroller, enabling the observer to interface directly to the network, requiring little pre-processing by external computers. The high level of process automation embedded in each microcontroller separates the telescope control system from other distributed controllers that merely relay the sensor and actuator state through the network to a central computer after completing rudimentary processing.

## **2.2 : System architecture**

To clarify the reasoning behind the choice of the distributed control system, it is helpful to compare and contrast the two major control architectures available:

1. Centralised control architecture
2. Distributed control architecture

## **2.3 : Centralised control architecture**

A centralised control system is characterised by a single processing unit providing both low level control routines and high level data reduction and display (Diagram 4). The processing unit is usually a high speed microcomputer or a mini computer, such as a Sun SPARC station or a Intel 80486. Multiple tasks are executed by the processor to service each environment. These tasks include:

- Operating system
- Graphical display
- Operator interfacing
- Data reduction

- Controller algorithm interpreter
- Low level interrupt handlers

This list illustrates the diverse requirements of the processing unit. To ensure valid process control, both the low level interrupt handlers and the controlling algorithm interpreter must be adequately serviced by the operating system. The low level interrupt routines bypass the operating system and so, providing that the interrupt routines are small, the routines appear transparent to both the operating system and to the user. The priority of the task selector must first be to service the controlling algorithm interpreter, then the operating system and finally any other packages. However, since the operating system determines the task selection, it is impossible to ensure valid process control in all situations. This is due to the operating system memory requirements. Selection of each task is governed by a complex algorithm that usually requires memory swapping from disk to RAM and kernel memory allocation for the task selection parameters. For each task there is a memory, time and processing penalty which must be subtracted from the task time slice. Increasing the number of tasks decreases the processing time allocated to each task. When the processing time is negligible, 'thrashing' occurs. Thrashing describes the condition where the computer spends all the time slice period configuring the processor environment for the task. The solution to this diminishing time slice is to choose a computer with processing power far in excess of the predicted demand.

The centralised control system gives the user a simple process controller which is easy to understand, easy to program and control, but which has several inherent limitations:

1. Following system expansion the demand from the tasks could lead to thrashing.  
The only solution is to upgrade to a larger, more powerful computer.

2. The computer price tag is highly dependant on the processor speed. An increase in the control complexity will require a disproportionate increase in the capital outlay.
3. Selection of a host computer ties the user to vendor specific software and hardware.

Hence the centralised control system approach is suited for small scale, static, stand-alone control applications that can be easily modified by many programmers.

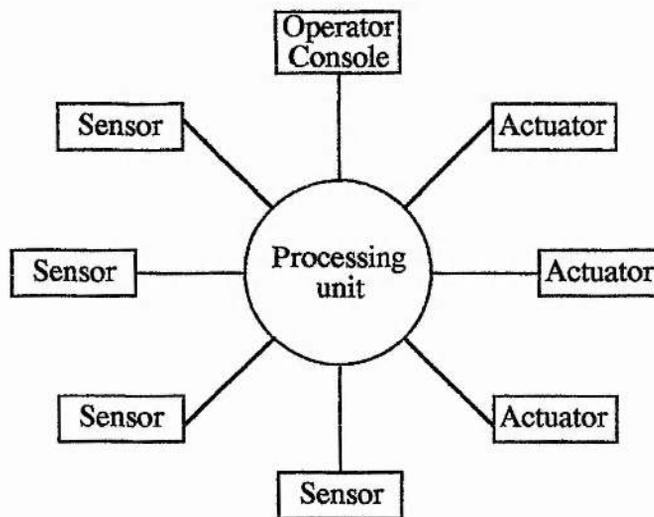


Diagram 4 : Centralised control system

#### 2.4 : Distributed control architecture

A distributed control system is characterised by multiple processing nodes situated on one or more networks communicating to other nodes and to an operator (Diagram 5). Each node interfaces to the network and to physical devices such as the sensors and actuators. Embedded into the node are controller routines, similar to the centralised control system:

- Network interfacing routines
- Process control interpreter
- Interrupt handler routines

Unlike the centralised control system, there is no task selection to degrade the process control. The node is dedicated to process control and to nothing else. This means that the routines that control the process will execute much faster than a similar centralised task, providing the user with security in the knowledge that thrashing and incorrect task selection will never occur.

An inexpensive microcontroller can be used, dedicated to simple bit wise manipulation. Since it does not have to cope with complex instruction commands, it is considerably cheaper than the centralised alternative.

The operator controls the system by sending network messages to one or more controllers through a remote console running a user interface program. The received message updates a node variable or selects a new algorithm for execution.

If the node cannot physically process the statement, further network conversations are initiated with an on-line database to resolve its situation. The database may initiate further messages to other nodes, download new statements to the initiator node or request user action. It can be seen that the operation of a distributed control system is complex to implement and analyse, but it lends itself to an open ended architecture with great potential for expansion, limited only by the network type and protocol.

The major limitation of a distributed control system is the use of network messages. Although single node execution is extremely fast, multi-node task execution is limited by the network media and protocol. For example, a process implemented by a distributed control system, linked by an ethernet running at 10MBaud using anti-collision algorithms will provide each node with a 99% chance that it can hope to

achieve 480 packets per second (50). Under heavy network loading where the anti-collision algorithms indicate corruption, the probability of success drops dramatically. To relieve the network load, multiple networks must be constructed using an internet router with zone list or alternative protocol such as a token ring. If simpler network media are used, the congestion problem increases. As the network demand increases either from a reduction in usable bandwidth or an increase in communication demand, process assignment must be carefully selected to minimise inter-node communication.

From the considerations above, various guidelines can be noted. The total system must be divided into units, each unit executing a specified task.

Inter-node communication must be treated as an asynchronous, time independent routine. For example, the interpreter must not 'hang' whilst awaiting a response, but continue monitoring until receiving a message or until an acknowledgement time-out is reached.

Great care must be taken when providing a hierarchical network structure as inter-node communication is just as important to the process completion as is the operator input.

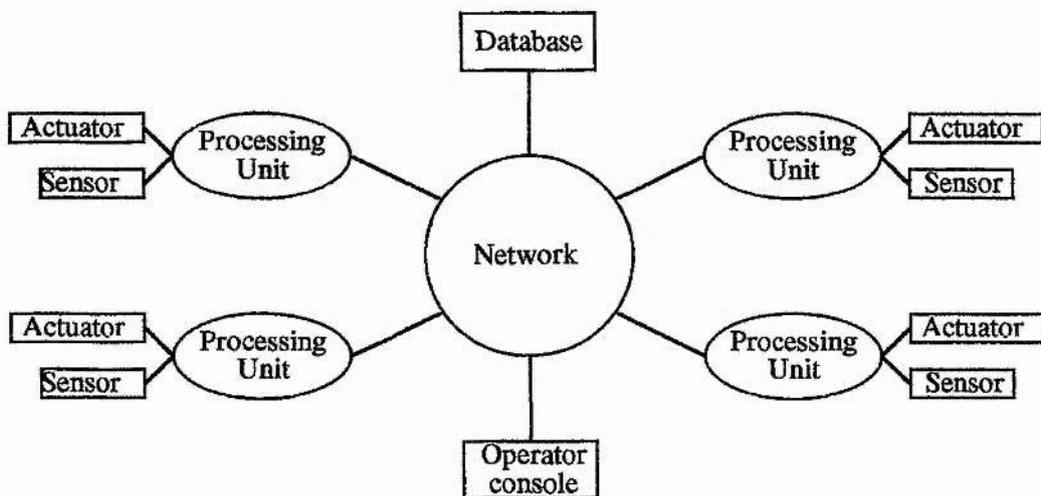


Diagram 5 : Distributed control system

Table 1 shows the characteristics of each type of control system architecture. Diagram 6 and Diagram 7 display the processing and data transfer requirements of both centralised and distributed control system architectures.

Parameter	Control system architecture	
	Centralised	Distributed
Processor type	80486DX @ 66 MHz	8031 @ 25 MHz
Processor application	Numerical, graphics	Process control
Processor cost	High (> £1000)	Low (£20)
External devices	High (48 I/O lines)	High (48 I/O Lines)
Initial development time	Medium	V. long
Controller Language	Assembler C/C++	C, Assembler, Occam
Console Language	C/C++ (DOS/UNIX)	C/C++ (WINDOWS/UNIX)
	Software / Hardware	Software / Hardware
Initial Installation	Easy / Difficult	Medium / Easy
Slight modification	Easy / Easy	Medium / Medium
Serious expansion	V. Difficult / V. Difficult	Medium / Medium
Ideal programming paradigm	Sequential / single task	Multitasking, parallel processing
Ideal operation system	DOS / CP\M	UNIX / WINDOWS
Process loop time	Fast	V. Fast
System speed	Fast	slow (Network bandwidth limited)
Applications	High speed robots, Small PLCs, standalone dedicated processes	Physically large site control. Intelligent co-operative automatons. Generic control block

Table 1 : System architecture characteristics

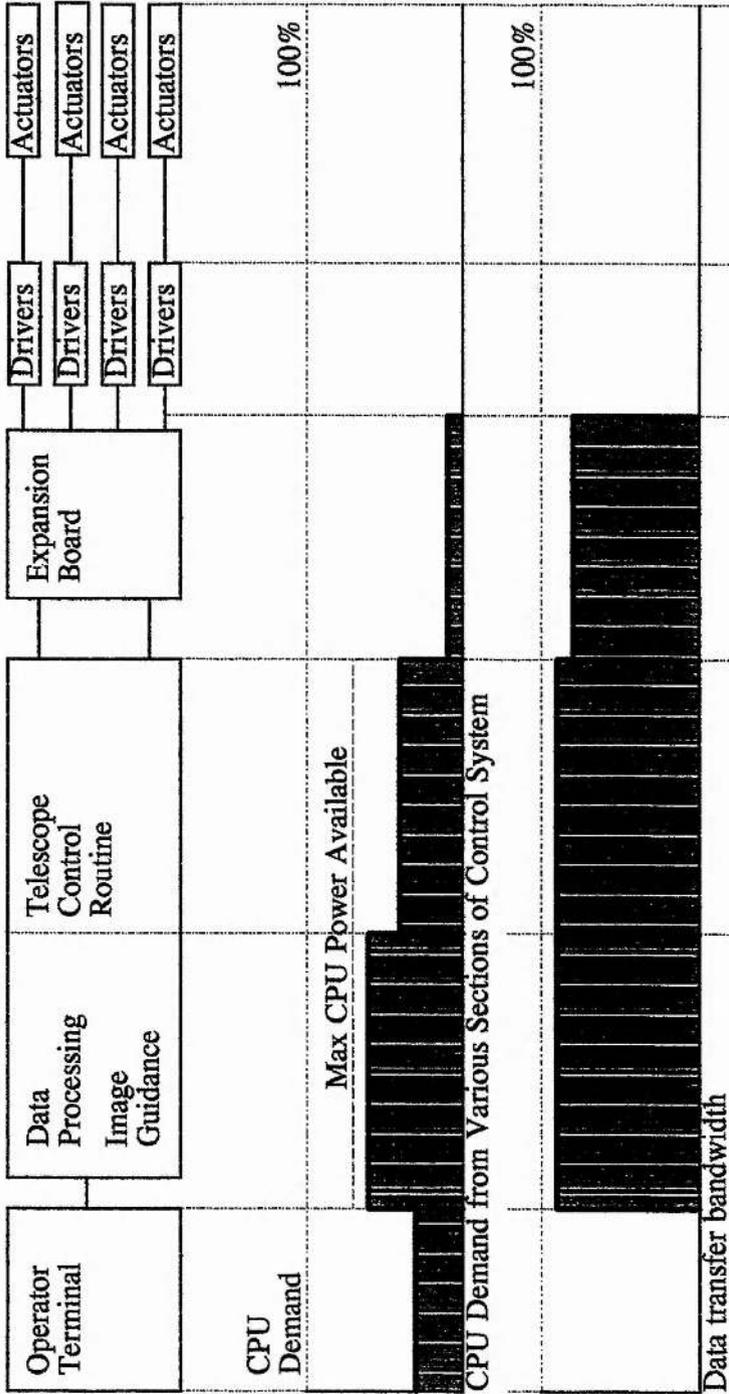


Diagram 6 : CPU Demand and data transfer bandwidth from a centralised control system

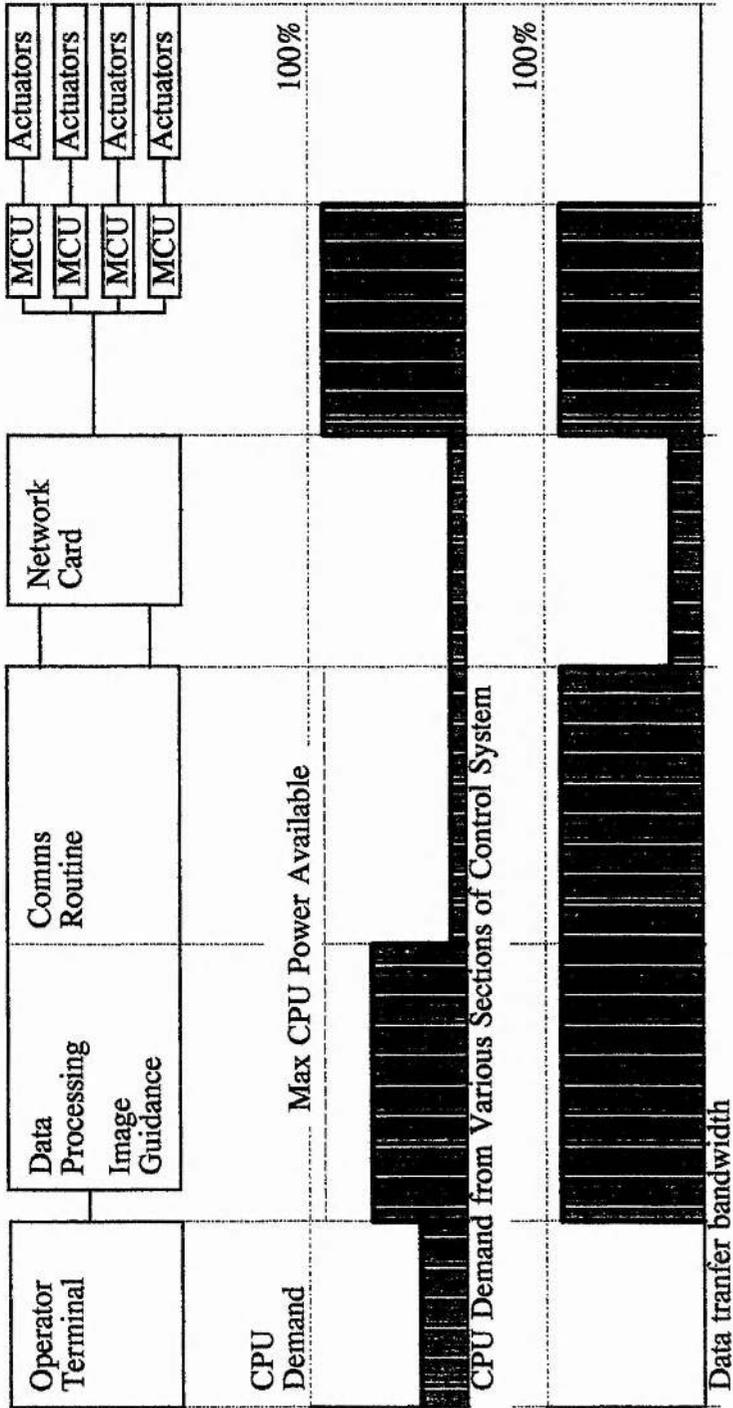


Diagram 7 : CPU Demand and data transfer bandwidth from a distributed control system

## **2.5 : Networks used for distributed architectures**

The change in architectures from single processor to multiprocessor environments has significantly influenced network design. The current demand is for a reliable, high bandwidth multinode network which can withstand operation in an electrically hostile environment. The twin photometric telescope is an example of an electrically hostile environment suffering from mains borne spikes, long term voltage fluctuations, radio frequency interference, inductive crosstalk and variable earth potentials around the telescope. Since the telescope commands are distributed to each microcontroller unit through a network, the system integrity and performance are directly related to the level of security of the network from interference. Thus the network must be chosen to suit the environment.

There are many different networks currently available to the designer, the most common of these are the 20mA current loop, RS232, RS422, RS423, RS485, IEE488, SCSI and ethernet. Table 2 summarises their properties.

Network	Max. nodes.	Attributes *	Bandwidth B / M	Max. Line B / M	Collision Avoidance	Mode	Use
20mA	-	S			Software	Half Duplex	Teleprinters
RS 232/V.34	2	IS	19.2K@15	1.2K@1.2K	-	Full Duplex	Standard serial link.
RS 422	10	BS	10M@10	100@1.2K	-	Simplex	Remote display monitors
RS 423	2	USD	100K@10	3@1K	-	Full Duplex	Now upgraded to RS 449
RS 485	32	DBS	10M@40	100K@1100	Software	Half Duplex	Microcontroller networks
GPIO/IEE 488	256	DP			Hardware	Half duplex	Hewlett Packard systems
SCSI	8/256	DP	4M@100	100K@1.1K	Hardware	Half Duplex	Peripheral I/O, Hard disk
CAMAC/IEEE-583	224	PS	1M	-NA-	Hardware	Half Duplex	Minicomputer I/O standard
Ethernet/IEEE-802.3	216	S	10M	≈ 1M@3K	Software	Half Duplex	LANs, Multiple workstations
Fibre optic	2	S	40M	1K	Hardware	Half Duplex	Hostile environments

\* S = Serial, P = Parallel, U = Unbalanced, B = Balanced, I = Bipolar, D = Differential

Table 2 : Network types

The twin photometric telescope network was required to exceed the following specifications. A 7 node, duplex network was required to operate over a length of 40 meters with nodes distributed unevenly along its length. The bandwidth must be sufficient for 32 characters to be transmitted in under 0.1s to give real time response. This gives a bandwidth of 3Kbaud. The network must have high noise immunity from radio frequency interference, mains borne glitches, fluctuations and inductive coupling from nearby motors. A high common mode rejection ratio was required to overcome the differing earth potentials around the system. The networks which fitted these parameters were the 20mA, RS485, IEE488, SCSI and fibre optic.

The IEE488, commonly used for the Hewlett Packard instrumentation network, and the SCSI, used for computer interfacing to memory devices are best suited to high level computer peripheral links and were eliminated. 20mA has inadequate protection against noise and was also eliminated.

RS 485 link was considered, and would have been the ideal network to use, but due to the high noise levels in the telescope, interference would still be present in the network.

A fibre optic network was considered ideal for the twin photometric telescope, with each unit being linked by a fibre optic transceiver (Appendix F.37). The electrical isolation provided by the fibre optic network eliminates interference from external sources and from non zero ground levels. Applying the fibre optic network to a SCSI based serial protocol would enable a fast, error free network with a tested collision avoidance algorithm.

Each message is propagated around the network by the re-transmission of the received message using the point to point construction of the fibre optic network (Diagram 8). When the message returns back to the source node, it is blocked from re-transmission. This stops any line clamping that may occur.

Each unit receives all the network messages. Upon receiving the packet terminator signifying the end of message, the message target field is compared to the internal unit address. If equal, the message is for the unit.

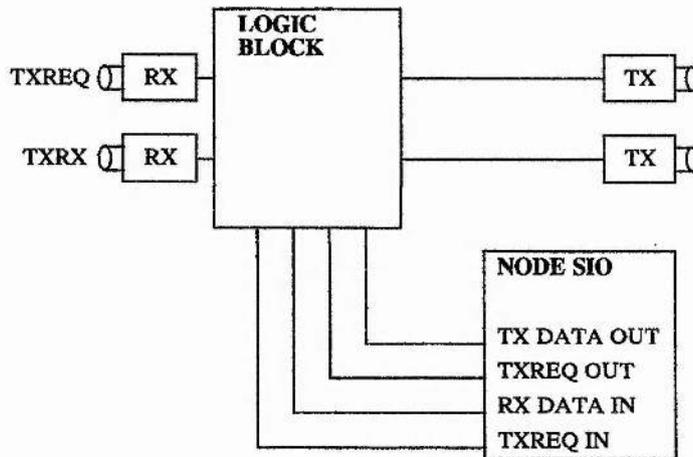


Diagram 8 : Network node controller block diagram.

After a review of the costs of the fibre optic network (£200) it was rejected in favour of the 20mA current loop. Whilst it degrades network performance and dynamic control response, the following protocol is still valid.

## 2.6 : Design of network

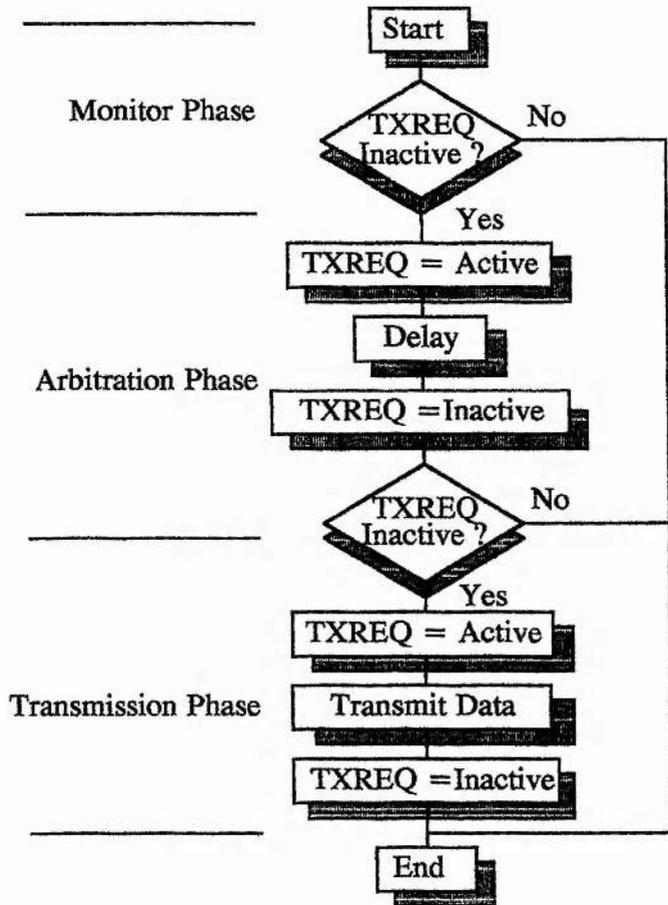
The network designed comprised of 2 independent tri-state lines (Table 3).

<b>Line</b>	<b>Tag</b>	<b>Description.</b>
Data	TXRX	Data transmit and receive
Request for transmit	TXREQ	Data line status (request transmission / transmission pending)

Table 3 : Network lines

Message transmission follows the following protocol (Flowchart 1, Diagram 9):

Both DATA and TXREQ lines are initially inactive. When a message is requested for transmission, TXREQ is monitored for inactivity (monitor phase). If TXREQ is active, the request is aborted, and should be resubmitted later. The node then asserts TXREQ for a pre-defined period unique to the node (arbitration phase). Any other requests for transmission from other nodes will abort at the monitor phase, giving the transmitting node priority over later requests for transmission. After the pre-defined period TXREQ is released, and is monitored for activity. If active, transmission from a higher priority unit has started and the (lower priority) routine must be aborted. If inactive, TXREQ is asserted and transmission can be started. After the last bit of the last character has been shifted out from the serial input /output (SIO) shift register, the TXREQ line is released signifying that the transmission is completed and has been successful.



Flowchart 1 : Network transmission protocol

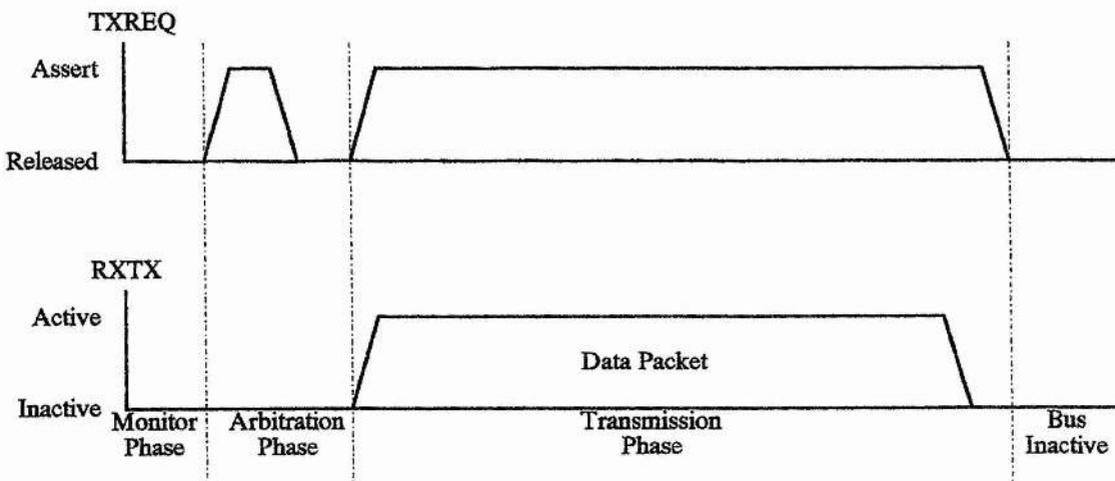


Diagram 9 : Arbitration state

## 2.7 : Network capacity

The amount of dynamic system control is limited by the available network capacity.

The maximum number of messages per second is:

$$\text{Number instructions/Sec} = \left\{ \frac{(\text{Bits/Character} + \text{Parity}) * \text{Message Length}}{\text{Baud Rate}} + \text{Arbitration time} \right\}^{-1}$$

Equation 1 : Instruction capacity of a network

## 2.8 : Network packet protocol

The network packet should contain the following information:

Target address field, host address field, transmission start time, message creation time, command type and associated data and terminator fields. Since the processor clocks are not synchronised and since one of the design parameters specified time independence across the network (See Page 38), the two time fields are ignored. The resultant packet adhered to the following protocol (Table 4).

< TARGET >	< HOST >	< COMMAND >	< DATA >	< TERMINATOR >
8 Bit ASCII 'A'-'Y'	8 Bit ASCII 'A'-'Y'	8 Bit ASCII 'A'-'Y'	n * 8 Bit ASCII n < 27	ASCII(0x5A) 'Z'

Table 4 : Network packet protocol used on the twin photometric telescope

The target address is a 8 bit ASCII coded field which defines the recipient address.

This address is compared to both the node and wild card addresses stored in the

erasable, programmable read only memory (EPROM) and if equal to either of the EPROM stored values the message is for that processor.

The host address, similar to the target address, is an 8 bit ASCII coded field but defines the sender address. It is used to specify the call-back address when replying to a message.

Valid addresses can be any character between ASCII(0x41) ('A') and ASCII(0x59) ('Y'). The addresses are specified as follows:

Address	Process description	Address Type	Macro Set
A	R.A. main	Physical	
B	DEC main	Physical	
C	R.A. offset	Physical	
D	DEC offset	Physical	
E	Counters	Physical	
F	Dome	Physical	
G	Operator control	Physical	
H	Log	Virtual	
I	Error	Virtual	
W	All Virtual devices	Macro	{ H,I }
X	All devices	Macro	{ A,B,C,D,E,F,G,H,I }
Y	All Physical devices	Macro	{ A,B,C,D,E,F,G }

Table 5 : Network addresses

As well as the physical addresses (A-G) there are also virtual, wild card and macro addresses. These enable compact network wide instructions to be sent quickly to physical and emulated nodes using one command. For example sending a RESET

instruction inserting 'Y' in the target field resets all physical devices, including the operators console.

## **Chapter 3 : Design of the distributed control system for the St Andrews twin photometric telescope**

This chapter describes in detail the design and construction of the system hardware used on the twin photometric telescope.

### **3.1 : Overview**

The control system design is subjected to constraints governed by the concept of a generic distributed control system as outlined in chapter 2.

#### **Parameters relating to the distributed architecture**

1. Each physical process must be modelled by a software algorithm.

The physical process must be suitable for embedding into software for use in the given microcontroller. For example, controlling the angular position of an axis using actuators and encoders lends itself easily to embedding into small microcontrollers, but CCD image reduction clearly would not be applicable due to the memory and floating point calculations required.

2. Each algorithm must be time independent of other physical processes.

If mutually dependant algorithms were embedded into separate microcontrollers linked only by a common network, the efficiency of both algorithms would be dependant on the network packet transmission rate. Since the network protocol used cannot guarantee immediate transmission, both algorithms must be time independent. Since there is a 95% chance that a message can be successfully transmitted in 1 second, this is not strictly true, but for physical processes required to control the telescope, where the process loop time is in the order of milliseconds, the network propagation rate is insufficient.

3. The controlling algorithm for each physical process must interface to a time independent network.

As above, commands sourced from the operator to the microcontroller pass through the network. Therefore this algorithm is time independent.

4. All microcontrollers are linked to the operator's console by one common multinode, duplex, non-hierarchical network.

Each controller has the ability to communicate to any other node, so only one network is used. If multiple networks were to be used, a suitable gateway must be installed, transparent to each microcontroller. Each node must be able to transmit and receive data packets without being overridden by a higher priority node. The only exception to this is the system failure scenario, where the console node should have a higher priority to enable system restart. This can be implemented by the console software as shown later.

These four parameters enable the distributed architecture to perform without a severe error occurring. Further constructional constraints were applied during system design in response to the problems encountered while developing the prototype.

1. The telescope system is required to plug into COM 1-4 of any 32 bit PC clone.

This enables the operator to update the console with ease, without disconnecting vendor specific boards or modifying complex configurations.

2. All network data characters shall be printable and readable on a VT52 standard terminal.

While the network was being installed, ambiguity existed over hot key functions from various consoles. For example, some dumb terminals send ASCII(0) and ASCII(10) with the carriage return character ASCII(13). This would cause a packet error in each controller, invalidating the previous message. Also, to analyse network messages, a dumb terminal was used as a probe, but the display was limited to printable characters. By changing the packet terminator to ASCII(90), 'Z' both ambiguity and network display conditions were satisfied. All other characters lay within the ranges 'A'-'Z', '1','0'.

After analysis of the telescope operation, it was found that there were six independent physical processes acting on the telescope that could be integrated into a distributed control system architecture. These were:

1. Reference right ascension

This consists of coarse encoder, fine encoder, sidereal drive, clutch solenoid, clamp motor, slew motor and limit switches.

2. Reference declination

This consists of coarse encoder, fine encoder, tangent arm, clamp motor, slew motor and limit switches.

3. Offset right ascension

This consists of encoder, tangent arm motor, centre detent and limit switches.

4. Offset declination

This consists of encoder, tangent arm motor, centre detent and limit switches.

5. Dome azimuth

This consists of encoder, hydraulic valves.

6. Photomultiplier counter

This consists of two photomultiplier pulse inputs and 1Mhz time signal.

The photomultiplier counter process required the recorded data to be transferred to the host console at regular intervals to stop data stacking in the microcontroller RAM. This would be negligible for long period integrations, but would invalidate the time independent criteria when undertaking high speed photometry and was discontinued.

### **3.2 : Electrical considerations**

When the design was conceived, a firm step was taken to use the nearest state of the art technology available to the department. This included new ICs such as generic array logic devices (GAL's), HCT series logic and integrated process controllers (IPC's). This drive for modern technology enabled multiple functions to be integrated onto one generalised device. This reduced board space, power requirements and noise levels, whilst returning constant propagation delays across logic gates and faster processing speeds. With reference to an earlier design based on the Z80 processor it was shown that a physical reduction in board space by a factor of 3 coupled with a power reduction by a factor of 10 could be achieved.

Surface mount packaging was used during the development of the prototype but it was found that the board manufacturing process was too unreliable for direct soldering of packages at the required pin resolution. Surface mount formats would have reduced the board size further, but no other advantages would be gained.

The main disadvantage of using high speed processor cards was a lower input noise immunity and higher crosstalk on the printed circuit board due to the lower gate hysteresis levels and higher clock rates used. Careful track placement, input filtering and device decoupling lowered the noise level of the card. It was also found that the initial expense of using new devices was returned on the reduced board size and board production cost.

### **3.3 : Printed circuit board manufacture considerations**

All microcontroller boards were identical in design and lead to 'mass production' runs, reducing the tooling and labour time. Boards were developed in house using

the facilities of the physics department electronics workshop. It took 8 weeks to produce 30 drilled, plated double sided boards ready for population.

During the construction of the prototype it was found that the board quality was poor and inconsistent; a fault tracked down to the opacity of the acetate and the developer / etchant fluid quality. A minimum track width of 0.2mm was realised with an average error rate of 10 cut tracks per board.

### **3.4 : Design of microcontroller boards**

The microcontroller design was split into three distinct areas; the microcontroller board and support devices, input / output boards to interface it to the outside environment and a power conditioning unit.

Each board was designed using the most applicable devices, but changes were introduced due to financial constraint. These modifications related to the functionality and reliability of each unit.

### **3.5 : The CPU board**

The CPU board comprised of a microcontroller unit, ROM, RAM, I/O buffer and memory logic to map the ROM, RAM and I/O buffer into the correct location in memory.

#### **Ideal Processor : IMS T414**

The ideal processor available was the Inmos T414 transputer, a 32 bit transputer unit with 2Kbytes of on chip RAM (Diagram 10).

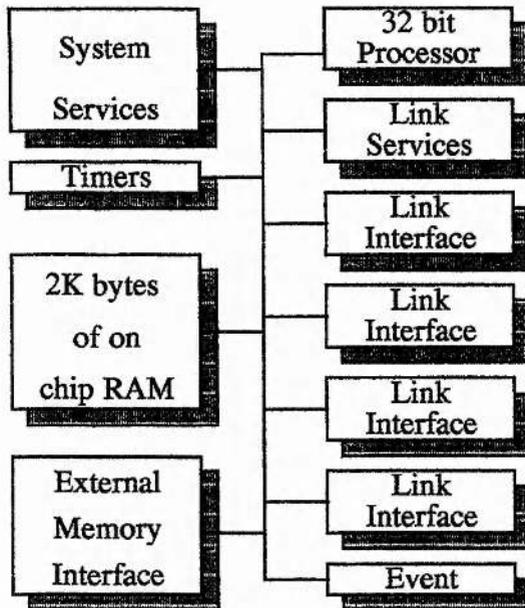


Diagram 10 : IMS T414 transputer block diagram

The transputer combines both system services and limited program / data memory in one integrated package, where the operating program can be bootstrapped into the transputer at run time through a link. This would significantly decrease the upgrade / development time which is normally limited by the EPROM erase time per EPROM. The T414 is clocked by an external 5Mhz clock to minimise radio frequency interference (RFI) across the motherboard which is then phase locked to an internal oscillator to produce a device wide 25Mhz clock, giving on average instruction throughput of 10MIPS (20MIPS peak). Each of the four Inmos links support 1.6Mbytes (47.2Mbaud) data transfer rates which minimise the second design criteria : "Each algorithm must be time independent of other physical processes.". Combined with the IMS C011 link adapter connecting the operator's console to the network this device would have minimised the complexity of the CPU board design and fully exploited the system architecture.

At the time of design (1990 / 1991) each transputer cost £60, with the Occam compiler costing £2000. This was considered too expensive and another microcontroller was chosen.

**Z84013 Microcontroller**

The Z84013 is a cheap, high performance microcontroller based on the long established Z80 processor which has seen service in the Z80, Z81, ZX spectrum, Research Machines computer units in the mid 1980s.

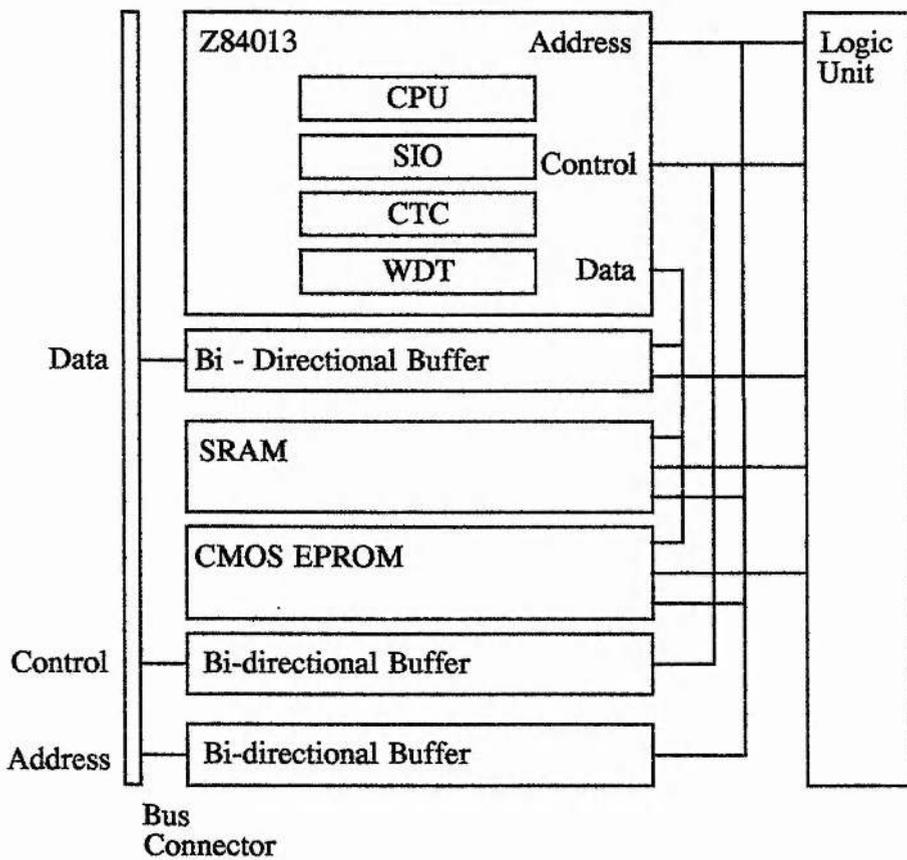


Diagram 11 : Block diagram of CPU board

It was chosen due to both cost and the range of integrated peripherals (Diagram 11). It comprises of a watchdog timer, two comprehensive serial ports, four timers, on

board clock generator and a 10Mhz Z80 processor and cost £10. While the Z80 has now been surpassed by various microcontrollers (i.e. 8051, H8), the processor is powerful enough to model the above physical processes. The Z80 processor has a wide range of cheap compilers, logic analysis decoders and in circuit emulators which enable fast development times. The drawback was that the Z84x series was designed for a single task, single processor environment, and application to a multiprocessor, multitasking environment limited the software to relatively rudimentary code generation (Appendix F.35).

Code execution on a processor of this type requires more memory per instruction, and so 8/16K\*8 zero wait state EPROM and 2K\*8 zero wait state RAM were used (Appendix F.36). All I/O was buffered by a 74HCT245 to prevent damage to the Z84013 after destroying the first prototype Z84015 from I/O glitches. Memory and I/O buffer logic was compressed into one 25nS 16V8 GAL to provide constant, clock limited, propagation delay. Using a GAL for logic enables the designer to quickly change the memory mapping of the ROM, RAM and I/O (Appendix A.1, E.3).

### **3.6 : The power board**

The power board was required to produce +5VDC, 5% regulated 1A logic rail and a +12VDC, 10% regulated 0.5A relay drive rail to each board with reset and glitch detect circuitry from a +20VDC unregulated input. It also acted as a backplane for mounting the CPU and I/O boards.

The ideal design would be to use a mains powered switch mode regulator alongside a comprehensive power monitor such as the MAX 696. A switched mode regulator sourcing power from a filtered mains outlet would provide a suppressed high current output which would adjust to any voltage fluctuations which could occur.

The power monitor would provide the system with a comprehensive range of functions, such as power fail, reset, watch dog timers and battery back up of CPU RAM.

The only disadvantage of this design was the cost, therefore a design was used based on discrete devices.

### **Power board design used**

Discrete regulators for the +5V and +12V supplies were used to control the board voltages, fed from a common power supply in conjunction with two LM311 comparators for the generation of RESET and NMI pulses (Appendix F.28). This design was cheap, but is susceptible to mains spikes and power fluctuations from inductive motors and to RFI on the lower voltage supply lines. The power loss through the regulators, dropping from 20V to 12VDC and then to 5VDC was excessive, and the regulators had to be secured onto the metal cabinet to provide a heat sink thereby lowering the working temperature. This was considered to be a very inefficient use of the regulators and reduced the maximum power level to each board. The printed circuit board size increased with the amount of extra components.

### **3.7 : I/O board design**

The I/O board was to provide a reliable and robust interface to the other functions. It must buffer and insulate all inputs from encoders, limit switches, sensors and centring arms from the I/O bus, latch output data to give TTL outputs and drive relay, lamp and inductive actuators. It should also decode the address logic from the CPU board and map each input bank onto the appropriate address space as seen from the Z84013 microcontroller (Diagram 12).

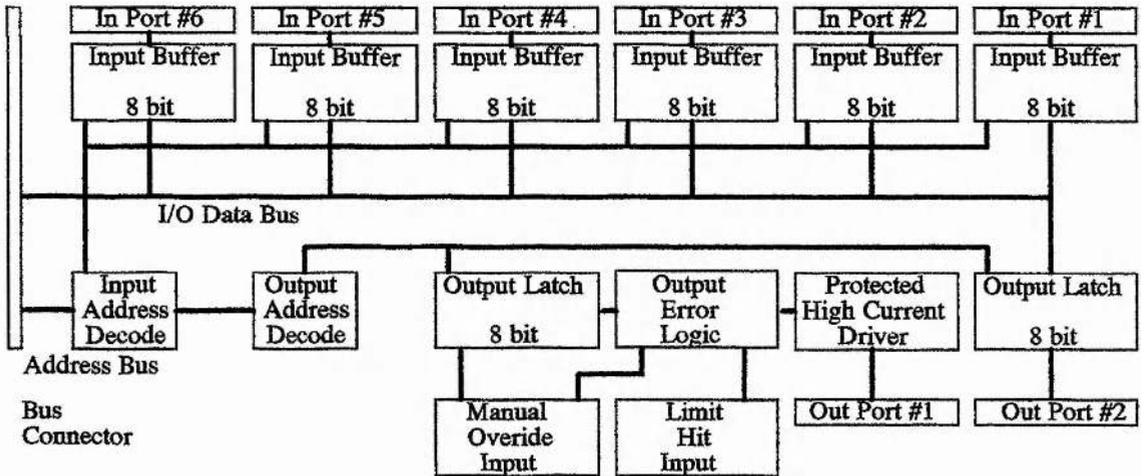


Diagram 12 : I/O block diagram

All inputs are buffered and mapped using 74HCT244 (Appendix F.34), and output mapping and latching uses 74HCT374 (Appendix F.33). The output control logic using inputs from both CPU and limit switches are programmed into a 16V8 GAL and output to a CA3262 high current inductive buffer / driver (Appendix A.2, E.4). This driver is able to power mechanical / solid state relays, small DC motors and lamps.

### 3.8 : Solid state relay design

All slew motors operate on an isolated 3 phase, 240VAC supply. The motors were previously activated by suppressed mechanical relays producing 'hot starts'. This is where arcing occurs across the tongues of the relay while contact is being made producing a wide spectrum of radio frequency interference. This welding effect degrades the relay contact area until the contact sticks.

Recently, optoisolated triacs which switch on and off near the zero voltage crossing point coupled with an integral snubber network have substantially reduced the

electrical and RFI noise, increasing relay lifetime. With features such as no contacts to wear or bounce, higher surge current capacity, lower power consumption, higher isolation voltages and intrinsically safe construction these devices are very desirable (51).

Owing to the cost of the solid state relays the design was modified to use a combination of both mechanical and solid state relays. Two solid state relays were used to enable each motor, with phase switching for direction changing achieved using a double throw mechanical relay (Appendix F.24).

The disadvantage with combining relay types is that mechanical relays have a longer undefined switching period than solid state relays, so arcing can occur during this period. This was overcome using software routines in the microcontroller which disabled the power to the given actuator while the mechanical relay switch was undefined (Flowchart 2).



Flowchart 2 : Flowchart of typical actuator direction change

### **3.9 : Pulse width modulated (PWM) DC motor driver board**

The telescope has three tangent arms which provide fine movement of the telescope. These are each controlled by a low voltage DC motor (Appendix E.9). After various designs it was decided to build a high current, high voltage PWM driver based around the UDN-2954W H bridge motor controller (Appendix F.25). This enables PWM control of motor armature / fields of up to 2A at 45V. Device features include regenerative braking, over current limiting and fast direction

changing. Logic to the driver was sourced from the field voltage and TTL inputs were optoisolated to reduce return line interference from the worn brushes and any inductive spikes generated from the motor.

### **3.10 : Board construction**

Owing to the high component density, double sided board was used. Density was restricted by both the CAD package resolution and the minimum reliable track width. Both the schematics and the PCB layout were designed using EasyPC, a software CAD package on an Elonex 33Mhz 80386DX computer. The PCB layout was printed full size using a Hewlett Packard Deskjet 500 printer and sent to the University photographic department for conversion to 20 $\mu$ M thick positive resist acetate. This process produced high quality acetates with small line edge effects which could potentially produce fine quality etched PCB.

### **3.11 : Cube unit**

Both photometer heads are controlled by an industrial BBC computer, called the CUBE. It consists of four 19" boards rack mounted, which in turn is mounted on the telescope counterweight struts. The computer controls filter and aperture wheels of both photometer heads, camera selection, camera integration and readout times, and infra - red field illumination of the main telescopes.

Photometer head control was effected from a keyboard in the warm room, with a separate teletext status display alongside the keyboard. The photometer heads interfaced to the computer by 4, 6522 versatile interface adapters (VIA).

In the early stage of the project it was decided to control the CCD camera functions from the CUBE. The functions were:

1. Camera select
2. Camera integration / readout time
3. Tube illumination

A wire wrapped board was designed and constructed. A multiplexer controlled the camera select, a software interrupt driven pulse width modulated timer controlled the camera integration / readout time, and a digital to analogue converter with a non-linear driver output controlled a set of 4 infra-red LEDs per tube.

It was found that the non-linear output driver stage provided inadequate resolution for the range of objects observed. This would require more resolution from the converter or a different LED brightness response curve.

The CUBE unit was integrated into the console by connecting the RS422 serial line to an additional SIO card and assigning the CUBE an address to support the existing telescope network protocol. The CUBE appears to be a another node on the telescope network.

### **3.12 : Construction of the sidereal drive**

The twin photometric telescope sidereal drive unit was common to each major telescope at the observatory. A radio clock downloaded universal time (UT) from the Rugby time service into an LSI 11 computer alongside the local sidereal time which was derived from a temperature controlled crystal oscillator. The Rugby time service and the local sidereal time (LST) are both superimposed onto a 1 MHz pulse and distributed to all domes around the observatory.

The 1 MHz signal is optoisolated and terminated at each receiver end and is decoded to display either U.T. or L.S.T. on seven LED displays. A byte wide output bus is available to other computers for synchronising time services. Also

available as part of the decoding board services is a clock output at 1 MHz (sidereal). This reference frequency is used to drive the sidereal unit (Diagram 13). The function of the sidereal unit is to accurately control the telescope right ascension (R.A.) at the nominal sidereal rate allowing the operator to guide and set in R.A. by changing the speed and direction of the sidereal drive. The sidereal drive uses three boards to effect this control from the 1MHz sidereal frequency.

1. Divider and rate selector unit
2. Dual analogue to digital converters (ADC) to produce high quality triangular waveforms
3. Power amplifier driver

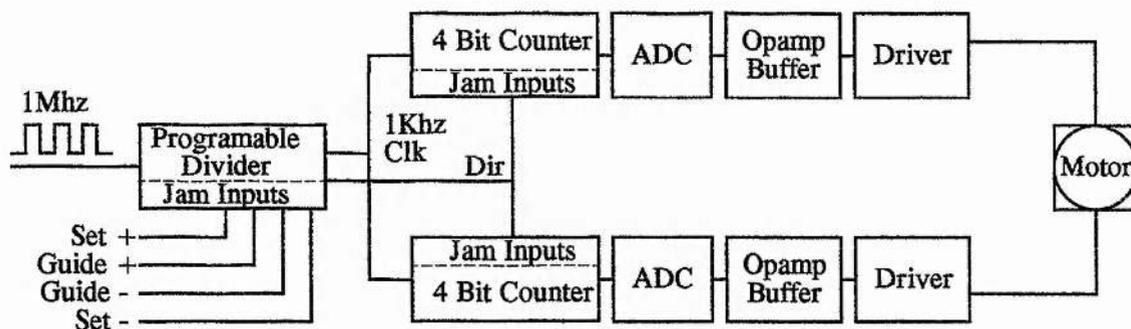


Diagram 13 : Block diagram of sidereal drive unit

The 1Mhz input frequency is fed into a divider circuit which is pre-set by one of four jam inputs, Set +, Set -, Guide +, Guide -. The output consists of both a master clock nominally at 1KHz and direction selector. This output frequency is then fed into two 4 bit counters and ADCs which with the direction selector produce two triangular 2.5V waveforms in phase quadrature. These two waveforms are then amplified by a totem pole drive stage and then fed into both windings of the stepper motor. Table 6 shows the phase quadrature relationship to the direction.

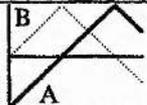
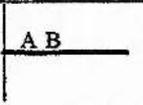
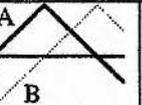
Mode	Set -	Guide +	Track	Guide +	Set +
Shaft rotation	-ve	Off	+ve	+ve	+ve
Phase quadrature	$A < B$	Undefined	$A > B$	$A > B$	$A > B$
Waveform					

Table 6 : Relationship between mode selected and shaft rotation

Before decommission, it was found that the sidereal drive was faulty. The nature of the fault was characterised by an intermittent jitter on the stepper motor output which indicated a discontinuous phase relationship in the drive to the stepper motor windings. This caused the telescope to lose track over a short integration period. Subsequent failures in other units during 1993 around the site gave cause for concern for the ageing drive design. It was decided to build a dedicated processor to control the stepper motor which could interface onto a standard RS 485 network or to the previous system. Since the design was more recent, two proposed controllers were designed, using the H8 and 8051 microcontrollers but due to the cost of the design (£200) it was decided to retain the old unit and to interface directly to the mode inputs through an opto-isolator (Appendix F.26). As a consequence of retaining the original driver unit the problems with the sidereal drive unit still remain.

### **3.13 : Construction of the dome controller**

Dome control required two functions:

1. Raise and lower the shutter
2. Rotate the dome

The shutter operation used a 3 phase 230VAC motor which connected to a mains lead situated on the wall through a Plessey multiway connector mounted on the side of the dome. Another Plessey connector allowed the operator to close the dome in emergencies independently using two 12 volt batteries and a 12 volt motor running in parallel with the mains operated motor.

The motor control was modified and simplified by only using one enable switch mounted on the dome connector assembly.

The dome rotation is controlled by a hydraulic control system (Diagram 14). The entire dome section is mounted on six wheels. Three wheels lift and rotate the dome at a given height in both directions from hydraulic control lines from the hydraulic pump motor. The other three wheels act as air springs to give an even dome ride.

The direction is set by enabling one of two hydraulic valves corresponding to clockwise and counter clockwise directions (Appendix E.30). The current to each valve is pre-set to give different speed levels (oil flow rates). To disable the dome rotation while the shutter lead is connected to the dome, a signal line is fed to a relay which in turn activates a 'dump valve', disabling any rotation.

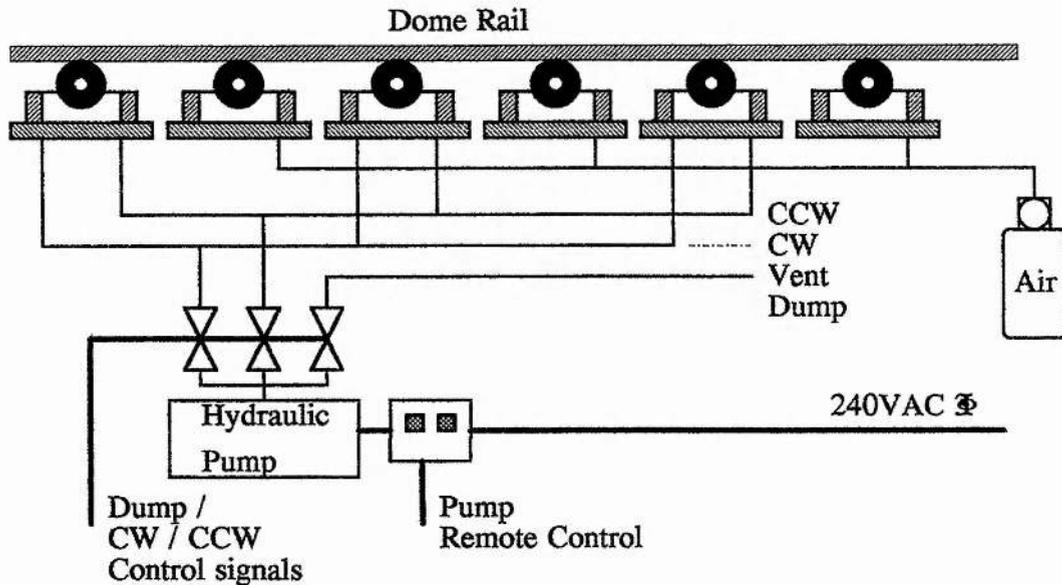


Diagram 14 : Dome rotation schematic

After testing processor boards in situ it was found that an unacceptable level of interference was present when the dome hydraulic pump was enabled and disabled using the mechanical actuator. The interference was characterised by invalid reset pulses being generated by the LM311N comparators on the power boards by a high voltage glitch which was propagated through the network and power lines. After examination of the hydraulic pump and actuator, it was found that a pre-set current of 15A was needed to be dialled into the actuator to enable the pump to operate, far higher than the stated rating.

Two methods were proposed to overcome this problem.

1. Enable the pump to run continuously
2. Eliminate the voltage glitch

The pump emitted a loud and annoying noise when in operation, and so the pump room walls and ceiling were lined with 2" polystyrene to reduce the noise levels.

The majority of the noise was transmitted through the suspended wooden floor, so the pump was mounted on its own custom built concrete plinth directly in contact with the foundations. The noise now emitted from the pump is comparable to a computer fan.

When the pump was tested, it was found to cut out after 1.5 hours. This fault was traced to the high temperature of the motor, which was very hot to touch. Considered in conjunction with the dialled 15A current, 7A over the motor's specified operating current, gave cause for concern (Appendix E.29). The pump control gear was opened and was found to be wired for delta. This would give a high start torque, but operate at a higher temperature. It was decided to wire the pump for star, giving a softer start-up torque, but lower running temperature. Dome operation did not change as a consequence.

While the pump room was being altered, proposals were put forward to remove the voltage glitch. Three were considered.

1. Low pass filters
2. Soft start modules
3. Zero crossing switches

Low pass filters are used to minimise the glitches on low power inductive actuators. Switching 15 amps would require expensive passive components, and would only reduce the effect.

A soft start module would lower the initial transient currents using a zero crossing 3 phase switch and a programmable current ramp. This proposal was discontinued because of the cost (£300).

As used on the other AC actuators, the action of a zero crossing solid state relay with an integral snubber network would bypass the initial voltage glitch, and so a 50A three phase 'CRYDOM' with fail-safe logic was designed (Appendix F.31).

### **3.14 : Construction of focus motors**

The focus motors for the main tubes required the operator to view the CCD camera monitor for object definition, and to alter the focus accordingly. The precise nature of the focus motors invalidated the design parameter "The controlling algorithm for each physical process must interface to a time independent network." so was not integrated into the control system.

For each telescope tube a three position switch was situated on the power panel and in conjunction with a motor run capacitor, operated the focus motors (Appendix F.21).

## **Chapter 4 : System software**

This chapter describes in detail the software used for both the microcontrollers and the console modules. It also confirms the use of the Windows operating system as the selected program environment for the operator interface.

#### **4.1 : Software overview**

Two types of software code were developed for the control system.

1. Low level code to embed each physical process into its own microcontroller
2. High level code for interfacing the system to the operator

As both software packages were to be connected through the network, great care was required to adhere to the defined network protocol, as the telescope system is highly dependent on the network packet capacity.

#### **4.2 : Low level microcontroller software for process control**

The assembly language software for the microcontrollers was written on an Elonex 80386DX under notepad, and assembled using the Smart Communications SCMA macro compiler program in a command shell. Common routines and definitions were stored in individual files, linked at compile time to provide clarity when writing main code.

When the prototype microcontroller board was built, three approaches to the microcontroller programming were considered.

1. Kernel operating system in EPROM, downloading program at run time into RAM

A small program stored in EPROM downloads the main machine code program from the operator's console disk at run time into RAM, which is then executed when an execute command is sent to the microcontroller.

This gives enormous flexibility during the initial development of the program and for user modification. If the T414 transputer was used, this would be specified by

the **BootFromRom** pin, enabling downloading from the link or from EPROM. The advantages are short development times, easy user modification and fault replacement, but there is one major disadvantage: programs stored in RAM are more susceptible to corruption than one in EPROM. For example, if a glitch changed one bit in RAM, the program would try to execute an erroneous op-code and abnormal program termination would occur. Whilst loading from network is very attractive to the system programmer, the hostile electromagnetic environment such as the twin photometric telescope produces glitches which compromise the reliability, therefore this approach must be discarded.

## 2. Kernel interpreter in EPROM, loading application specific statements at run time

A general purpose interpreter resident in EPROM loads statements from the network which define the physical characteristics of the telescope.

This gives the operator flexibility when further modifications are required, but restricts the system designer. As the statements define the physical characteristics of the telescope but not the processing algorithm, the programmer cannot modify the interpreter paradigms, relying on the initial development programmer to provide a comprehensive set of commands for any eventualities. The high version numbers on any commercial compiler brings this reliance into question. However, for simple modifications, it is easier for the technician to understand and modify text statements which describe observable physical parameters.

This interpreter was developed for the Z84015 prototype microcontroller and worked well apart from the limited interpreter commands owing to the limited RAM size. Since each statement was written in text, there was a high level of redundancy in RAM, and a maximum of 18 statements could be stored in memory. This was insufficient for the multiple actuator microcontrollers, and so discontinued.

## 3. Program and data stored in EPROM, downloading modified data at run time

The program stored in EPROM copies data stored in EPROM to RAM allowing the user to download new data to RAM on request.

Although the flexibility is limited to changing the variables in RAM the advantage of this is that it is fool proof to inexperienced technicians and is frugal in RAM usage. This is the current program methodology used in the microcontrollers.

#### 4. Program in EPROM using RAM for run time data.

This program is identical to the above approach without user modifications. It is too limited for this application and cannot be considered save for system testing.

Programming for a multiprocessor environment raises new objectives for the programmer. Should there be a common program, used in all microcontrollers with only a unique address byte specifying each unit or should individual programs be tailored to the specific process requirements?

Examining the programs from a system view revealed that even if different peripherals are used, such as clamp and clutch actuators or absolute and incremental encoders, they are all processing the same physical algorithm. The only exception to this is the dome controller, which treats the axis as a cyclic number instead of a linear 1-1 mapping. This indicated that common programming would reduce development time by producing well tested low error rate programming.

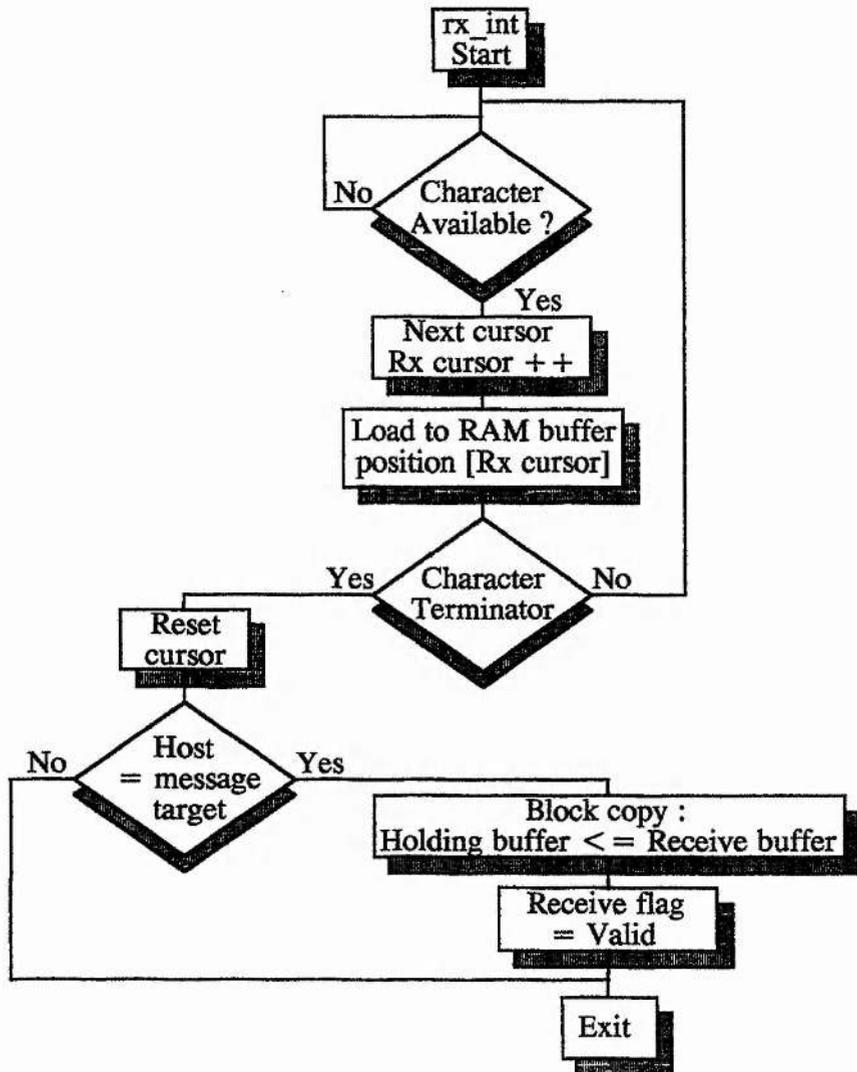
The disadvantage was that the common programming could not be tailored for individual scenarios. For example, the use of a 3 phase overload sensor on the R.A. clamp to indicate that it has applied the correct pressure on the clamp pad. This requires a routine to monitor the clamp overload sensor, and when active wait to see if the overload was generated from the transient inertial load of the clamp pad contacting the drive plate, sampling again for activation. There were two solutions to this problem:

1. Generalise the kernel program, using a simplified version of the statements loaded into RAM to specify what external devices were present on the



new character provided the interrupt flip flop is enabled. The host interrupt flip flop is disabled during packet transmission, to stop the transmitted characters being received. The available characters stored in the SIO buffer are transferred to a receiver buffer in RAM, cycling every 256 bytes, limiting damage that could occur on network error. Message synchronisation is achieved using the terminator character ASCII(0x51), 'Z'.

Upon receiving this character the receiver cursor is reset and the message target address is compared to the host address stored in EPROM. If equal, the contents from the RAM receive buffer are copied into a RAM holding register, enabling subsequent messages to use the RAM receive buffer without corrupting the previous valid message (Flowchart 3).



Flowchart 3 : Receive interrupt routine

#### 4.5 : Encoder interrupt routine : ctc\_int

This interrupt routine reads the encoders and updates both present and offset fields in memory. For each of the two reference axes a coarse and fine encoder are used (Table 7).

Axis:	Coarse encoder resolution:	Fine encoder resolution:
R.A. REFERENCE	24	2048 (2 <sup>11</sup> )
DEC REFERENCE	36	2048 (2 <sup>11</sup> )

Table 7 : Encoder detail

Since the coarse encoder is geared down, the fine encoder governs the pointing accuracy of the telescope. Thus splicing both encoders together will produce an erroneous result (Diagram 15). This will occur when pointing near the state change region of the coarse encoder. Within this region, backlash in the gearbox will produce two different codes for the same location, dependant on the slewing direction.

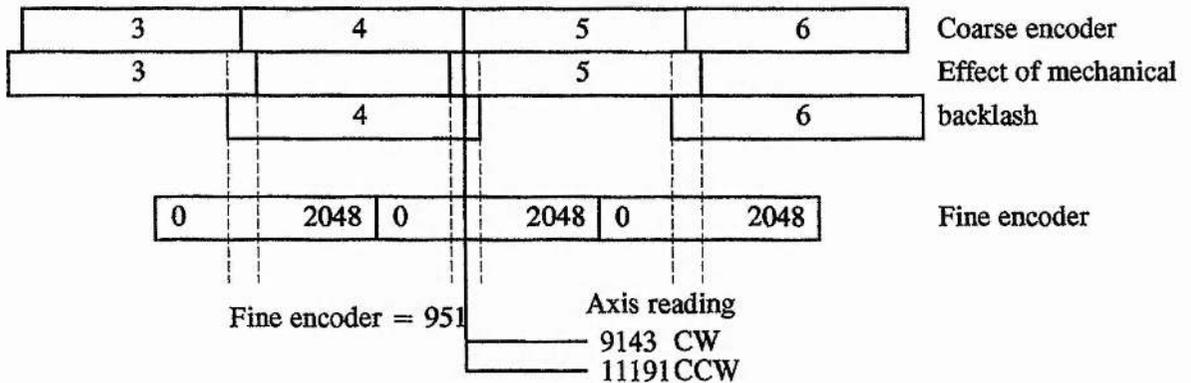


Diagram 15 : Encoder error from backlash.

Using a software interrupt running at 30 kHz, over twice the maximum state change rate of the fine encoder, this problem can be bypassed by reading only the fine encoder (Diagram 16). The maximum change in fine encoder reading between each interrupt time period must be 1. The only exception to this rule is when the encoder cycles through the zero point. At this point the change in encoder reading will be  $2^{11}$  or  $(1-2^{11})$ . When this change occurs the coarse encoder is read only if the axis is rotating CW, producing a fine encoder change of  $(1-2^{11})$ . Otherwise the coarse encoder variable in RAM is decremented.

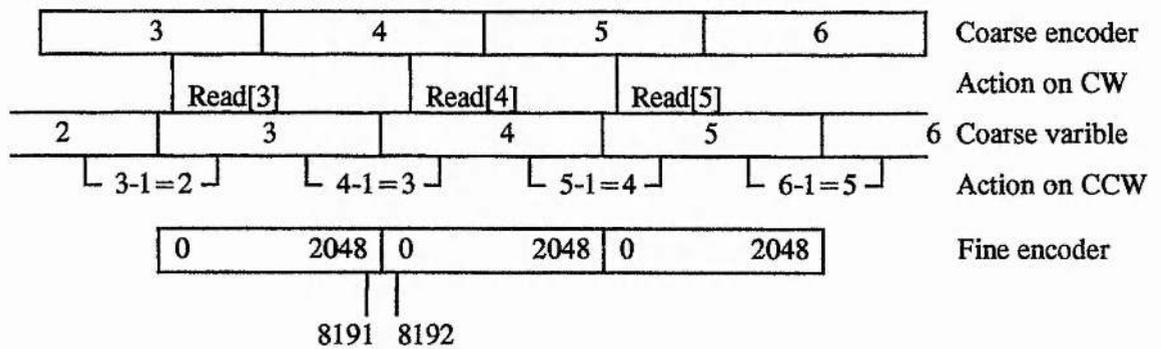
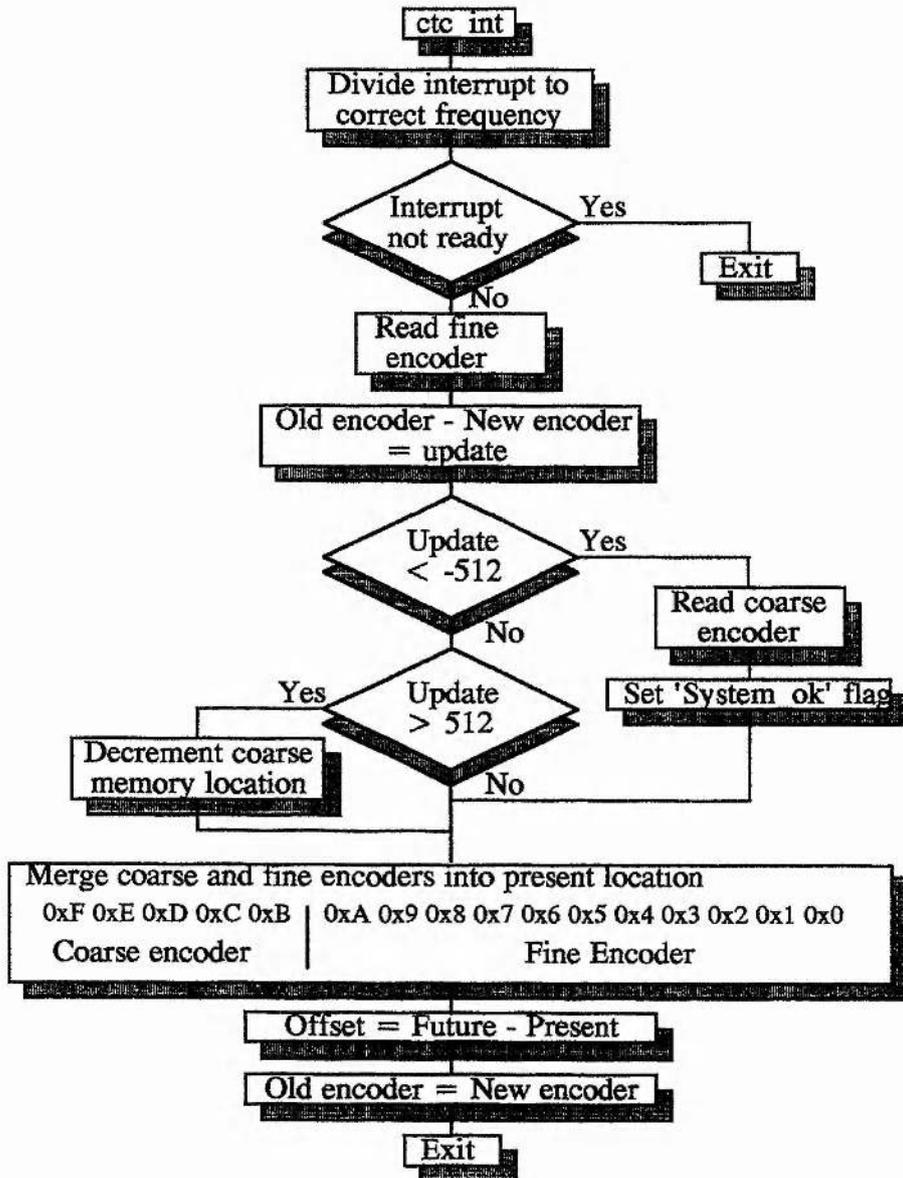


Diagram 16 : Modified encoder read routine

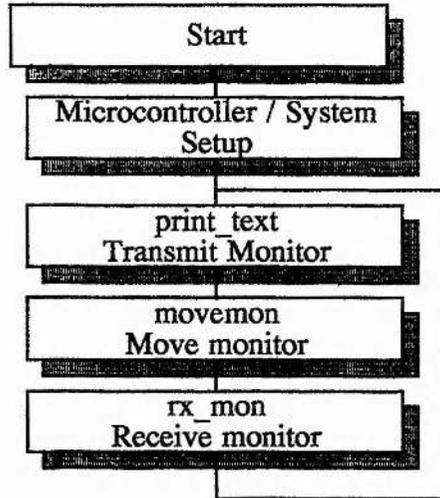
The disadvantage of using this routine lies in the difference calculation. The interrupt routine requires the RAM to hold the previous data for the coarse and fine fields. Since the RAM is invalid during power down, the microcontroller does not know the initial axis position at start-up. To reset the encoder interrupt variables, the fine encoder must be cycled through one zero point positively, so reading both fine and coarse encoders. When this occurs, the encoder flag is set (valid) and pointing can begin.

The details of the routine are shown below (Flowchart 4).



Flowchart 4 : Encoder merging routine

#### 4.6 : Main loop routines



Flowchart 5 : Main microcontroller loop

At runtime the microcontroller initialises both the internal peripherals and axis through `io_setup`, and then starts an infinite loop polling three routines `print_text`, `movemon` and `tx_mon` (Flowchart 5).

Each of these three monitor routines enable the microcontroller to interface to the network and act on the physical process embedded in the memory. Loop times are in the order of 50ms.

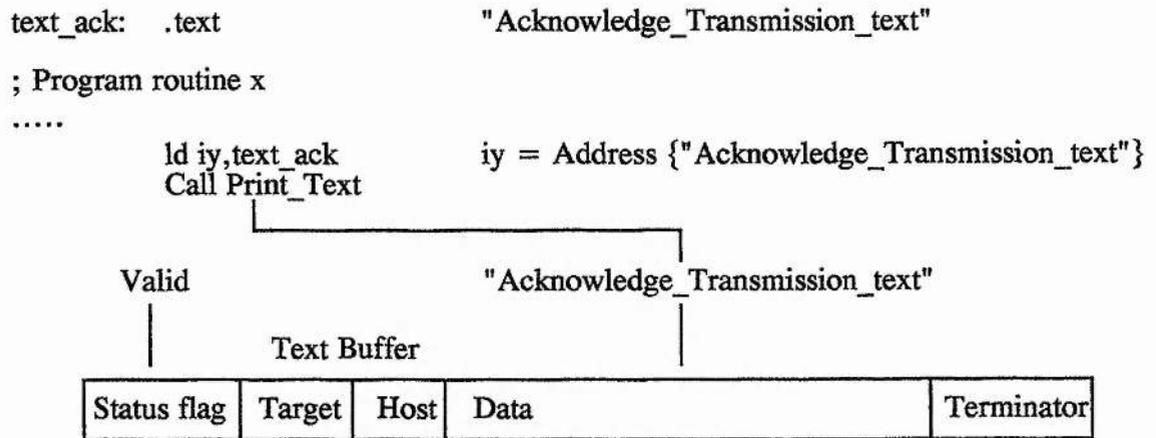
#### 4.7 : Transmit monitor : print\_text

This routine monitors the transmit buffer for a valid message. If one is present, transmission is started from this routine, using the specified transmission protocol.

To transmit a message packet, two routines are used:

1. `tx_request` Loading of message into transmission buffer
2. `print_text` Transmitting of packet across network

## tx\_request

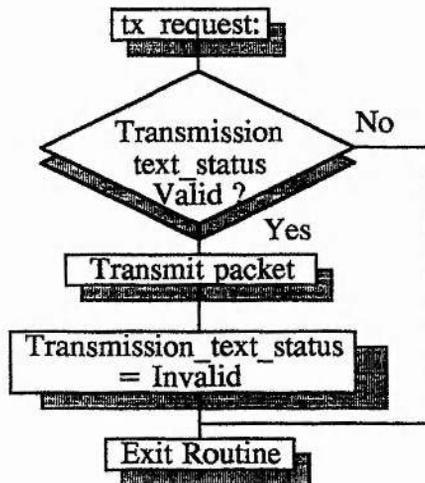


Flowchart 6 : Loading of transmission buffer

Messages to be transmitted are loaded into the transmission text buffer as simple text or data strings superimposed onto a text template previously loaded into the buffer. Host and target addresses are then added and the status flag is validated (Flowchart 6). Control is then returned to the main loop where the transmission is serviced by `print_text`.

## print\_text

When the `print_text` is called, it checks the transmit text status flag and if valid transmits the packet, clearing the transmit text status flag after successful transmission (Flowchart 7). If transmission was not successful, the process would be repeated until success occurs. If another message was requested for transmission during the arbitration delay period, the previous message would be overwritten.

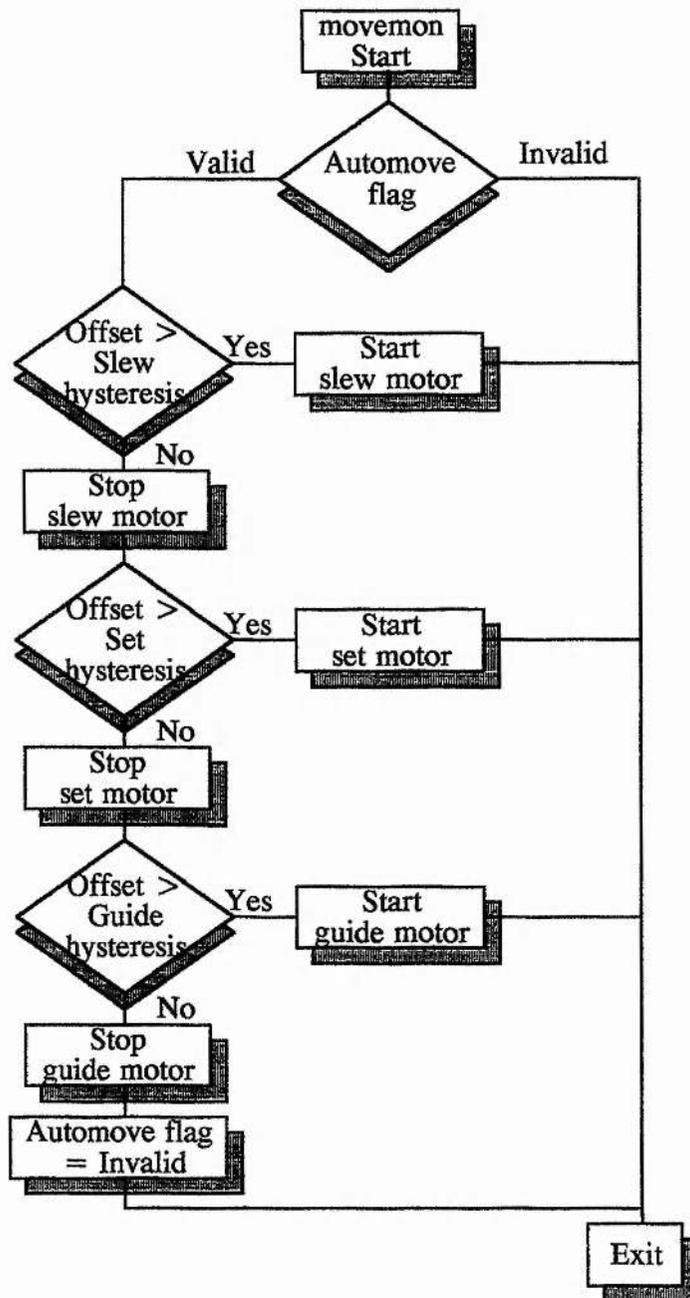


Flowchart 7 : Transmission check

#### 4.8 : Move monitor : movemon

To enable the microcontroller to move the axes, the **automove** flag must be validated by receiving an **\_MG\_** instruction from the network. The routine **movemon** enables the CPU to move under operator control or to set to a pre-defined position stored in memory address **future**, under control of the microcontroller (Flowchart 8).

The **movemon** routine compares the offset (calculated in the timer interrupt routine **etc\_int**) to three successively smaller hysteresis levels, corresponding to the physical hysteresis of each type of motor available (slew, set, guide). On overflow, the routine breaks to check or initiate motor start and exits the routine. If no overflow is present a call is made to stop the motor. If no overflow is present in all hysteresis comparisons, the **automove** flag is invalidated, locking out the microcontroller from initiating a further move. This allows the operator to request a manual move or to load new co-ordinates.



Flowchart 8 : Move monitor routine

## **4.9 : SIO overview**

The receiving of commands from the network comprises of two routines:

### **1. rx\_int**

An interrupt driven routine initiated from each received character successfully read into the SIO port A (but not during packet transmission from the same node).

### **2. rx\_mon**

A monitor routine interpreting received packets when made available by rx\_int.

The limiting number of data packets to each microcontroller is governed by the main loop time. Using the present code, the loop time is of the order of 50ms, corresponding to a maximum number of 20 packets. Thus the minimum data rate required to update packets using 8 bits per character and 24 characters in width is 3840, or 4800 baud.

## **4.10 : Receive monitor : rx\_mon**

The receive monitor polls the receiver holding register to see if any new messages from other processors have been received. If the receive flag is valid, it splits the data field into command, command qualifier and data sections. It then compares the received command byte with an internal command list, breaking out to the specific routine when equal.

As part of the network protocol, two call-back messages must be used to signify that the processor has received the message and that a task has ended (Diagram 17). When the message has been decoded into the valid individual fields, a **RX** message is returned to the host signifying that the processor has received the message and that it has been decoded. After the command has been completed an

\_ACK\_ message is sent to signify the successful termination of the individual command routine. An \_ERR\_ message can also be used in case of an error during the command execution.

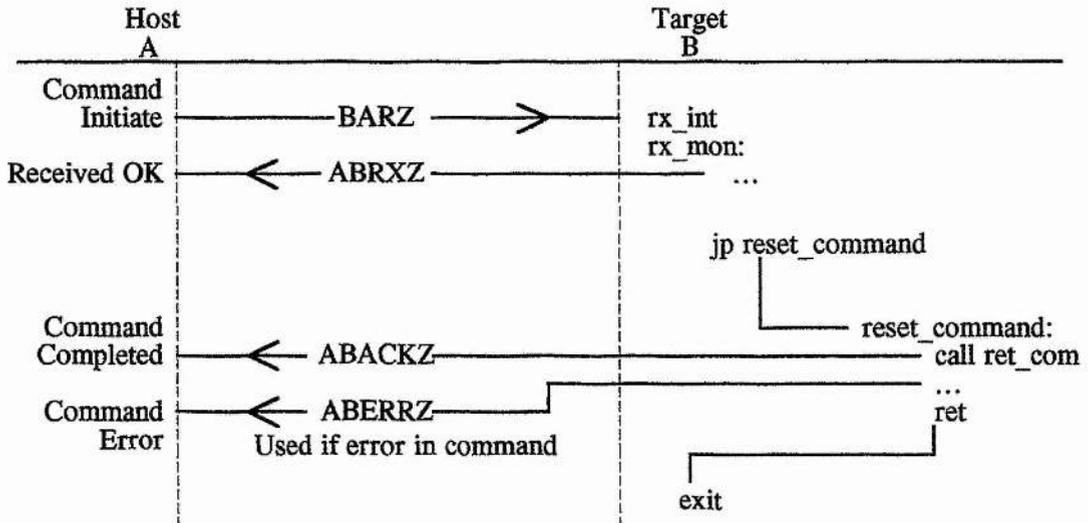


Diagram 17 : Network view of command processing

#### 4.11 : Commands used in remote microcontrollers

The commands used in the microcontrollers were expanded as and when they were required. Since most of the commands are common in all microcontrollers differing only in execution, the same network commands could be applied to different microcontrollers by changing just the target address.

Table 8 show the network commands. Using these instructions the operator can use and modify the telescope parameters.

Command	Description
M	Manual / automatic move
D	Read 16 bit data value from memory
P	Program 16 bit data value to RAM
H	Hard reset
R	Reset encoders
C	Centre axis using centring detent

Table 8 : Microcontroller commands

Command: manual / automatic move	Command qualifier:	Telescope axis	Dome axis
M	0	Stop axis normally	
M	1	Guide : CCW	CW slow
M	2	Guide : CW	CW medium
M	3	Set : CCW	CW fast
M	4	Set : CW	CCW slow
M	5	Not used	CCW medium
M	6	Not used	CCW fast
M	7	Not used.	Not used
M	8	Slew : CCW	Not used
M	9	Slew : CW	Not used
M	G	Select microcontroller automove	

Table 9 : Microcontroller actuator parameters

#### **4.12 : Manual / automatic move**

Each axis can be requested to move under control of either the operator or the computer. Manual operation of the axis will override the microcontroller at any stage in the program execution. Table 9 displays the command qualifiers.

This command enables the operator to control the telescope without knowledge of the operations logic. It is from the routines hidden behind these commands that the device specific code is compiled.

#### **4.13 : Software operation of the actuators**

Device dependent routines are embedded in the start and stop axis control routines separating the common control program from the devices' individual requirements.

Table 10 shows the three types of routines used.

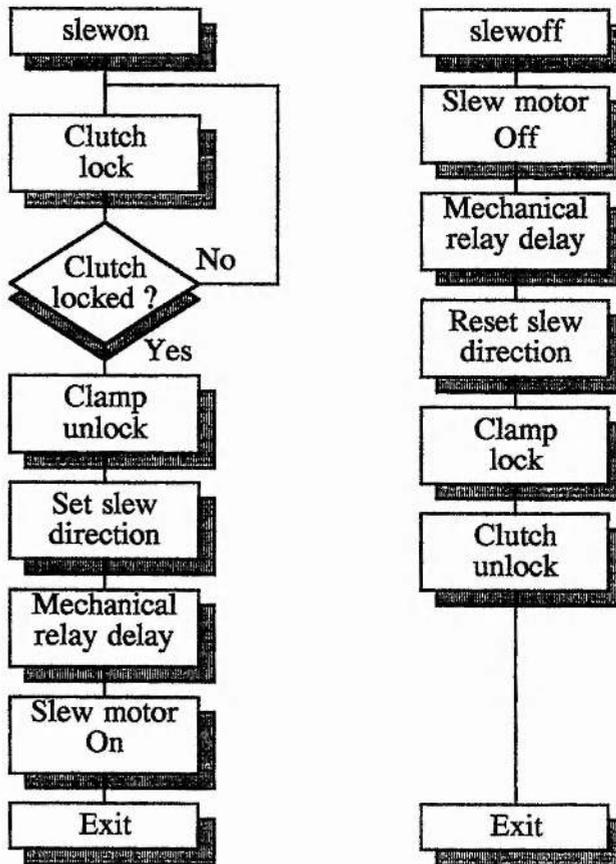
<b>MCU axis</b>	<b>Main actuators used</b>
R. A. reference:	Clamp, clutch, slew motor
DEC reference:	Clamp , slew motor
Other:	TTL logic levels

Table 10 : Actuators used for each axis

#### **4.14 : R.A. reference : slewon and slewoff**

The **slewon** routine starts the R.A. slew motor with regard to other actuators present on the axis (Flowchart 9). This routine locks the clutch in, unwinds the clamp and turns on the motor after waiting for the direction contacts to close.

The **slewoff** routine is the same of **slewon**, except reversed.



Flowchart 9 : R.A. reference slew routine

#### 4.15 : Clamp routines : clampon, clampoff

The clamp unit comprises of two distinct parts, the motor actuator and clamp plate (Diagram 18). The motor unit comprises of a three phase motor feeding through a worm drive gearbox to an arm which can freely rotate.

The clamp plate comprises of a pad mounted on a helical thread. At the top of the thread two lugs block the movement of the motor arm every half turn. When the

clamp is activated the motor arm rotates, building up inertia and hitting the clamp. The motor then unwinds the clamp allowing the telescope to move in R.A. The clamp uses two types of sensor for control. When the clamp unwinds, a magnet mounted on the side of the motor arm comes into close proximity with a reed relay mounted on the clamp base. When the reed relay breaks the clamp is sufficiently unwound and the motor can be stopped.

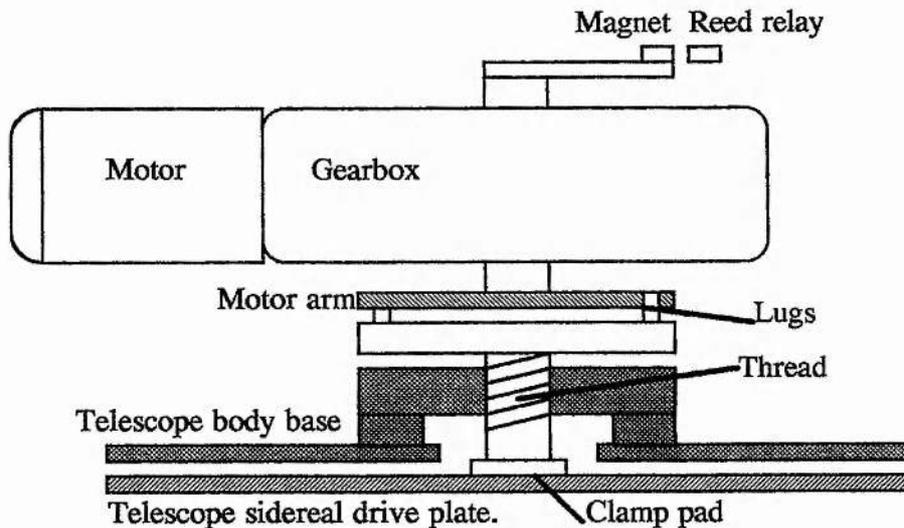


Diagram 18 : Clamp unit

A three phase overload sensor is used to disable the clamp motor on locking the telescope body to the sidereal drive plate. As the clamping action increases, the clamp motor stalls, increasing the current to the windings. When the stall current exceeds the pre-set limiting current, the overload sensor activates, disabling the clamp motor.

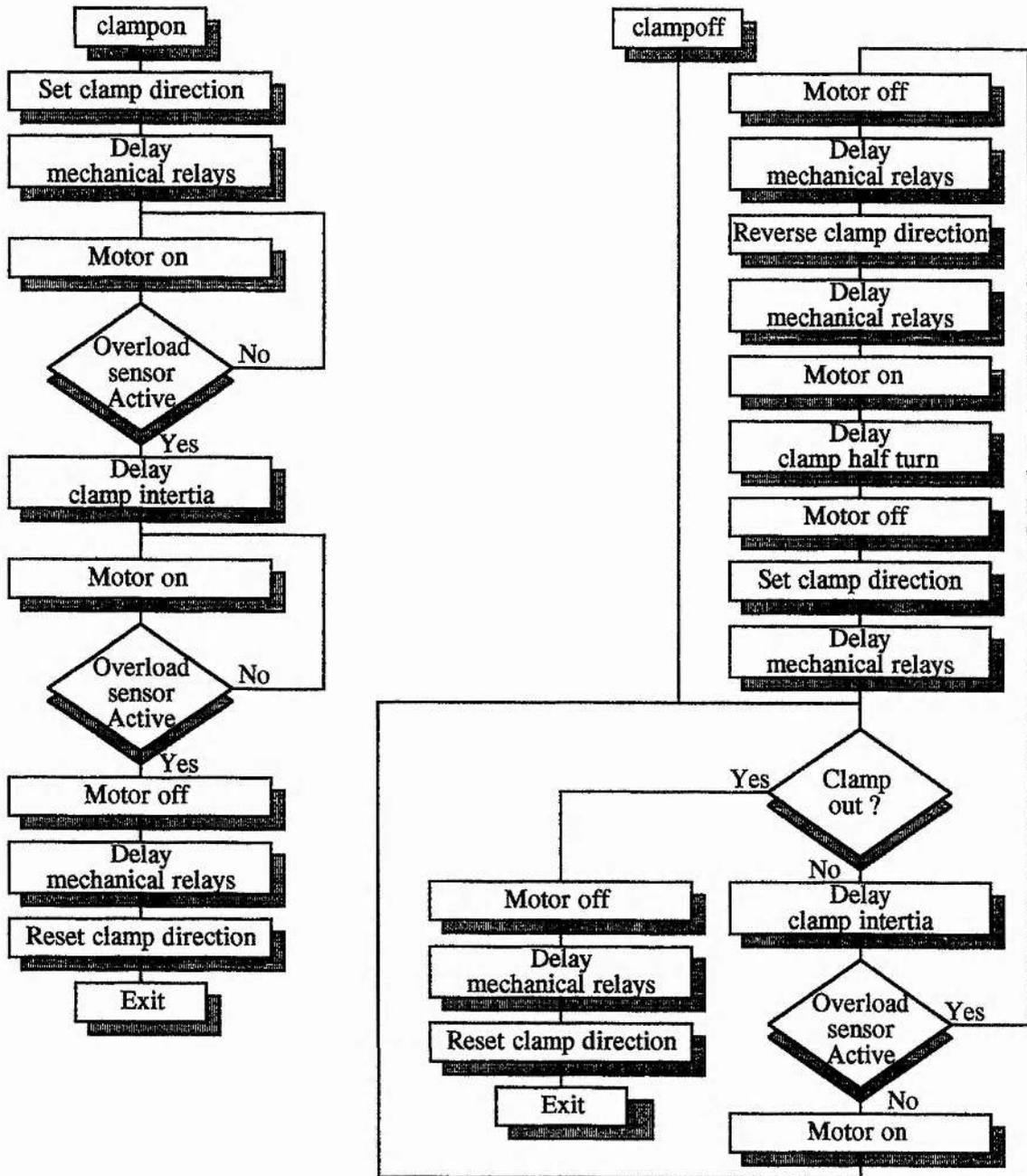
Whilst it was used for sensing when the clamp was locked, it is also active when the motor arm makes contact with the clamp lugs from both directions, causing a transient overload, creating an active pulse from the overload sensor. This is utilised in the new control system in event of the pad sticking onto the sidereal drive plate, a

problem that used to occur after infrequent use. When the motor arm struck the lugs, the overload sensor produced an active pulse equal to the inertial acceleration period of the drive plate. If the pad is stuck then the overload sensor will remain active. Both routines monitor the overload sensor for activity. When active it waits for a time greater than the inertial delay of the pad assembly and then samples the sensor again. If it is still active, it reverses the clamp back approximately half a turn (controlled by a timing loop) and strikes it again. Repeating the process enables the clamp to be unlocked from high pressure settings (Flowchart 10).

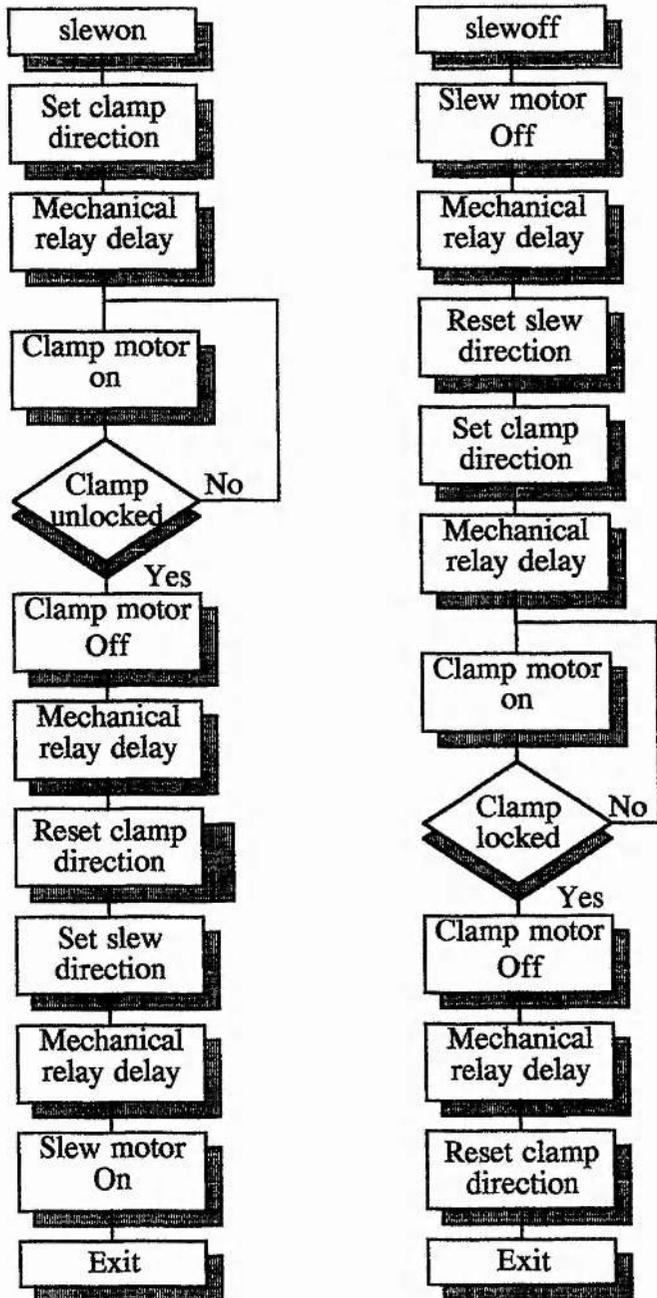
#### **4.16 : DEC. reference : slewon / slewoff**

The DEC slewing routine is a simplified version of the R.A. slewing routine. The DEC axis does not have a clutch unit and the clamp operates on two micro switches denoting 'clamped' and 'unclamped' states.

To start slewing in DEC, the clamp is unlocked and the slew motor is switched on. To stop slewing, the reverse operation is followed (Flowchart 11).



Flowchart 10 : R.A. reference clamp routines



Flowchart 11 : DEC. axis slewing routines

#### **4.17 : Other microcontroller axis routines**

All other routines output a logic state to an optically isolated independently controlled device.

<b>Command Qualifier:</b>	<b>Location</b>	<b>Read Write</b>	<b>Notes:</b>
P	Present axis position	R	Updated every 30Khz
F	Future axis position	RW	Destination from operator
O	Present - Future offset	R	Updated every 30Khz
S	Slew motor hysteresis	RW	Slew > Set
T	Set motor hysteresis	RW	Set > Guide
G	Guide motor hysteresis	RW	Guide > 0
N	I/O Port 1,2	R	0xFFFF0,0xFFFF1
W	I/O Port 3,4	R	0xFFFF2,0xFFFF3
H	I/O Port 5,6	R	0xFFFF4,0xFFFF5
M	Mech. - SSR relay delay	RW	Pre-set 50mS
I	Axial inertial delay	RW	Set at test
A	Arbitration delay	RW	DO NOT CHANGE
O	Tx inactive delay	RW	DO NOT CHANGE
K	System OK flag	R	Displays MCU status
D	Interrupt divider chain	RW	DO NOT CHAGE

Table 11 : Available memory addresses

#### 4.18 : Memory utilities

The memory commands **P** and **D** enable the operator to update and read fields in the microcontroller memory. The format for these routines are:

< Command > < Location > < Data string >

Where the < Command > is either **P** or **D**, programming or displaying the location, < Location > is the relevant section in memory (Table 11) and < Data string > is the data to be loaded, using the format 16 bytes containing ASCII(0x30) {'0'} or ASCII(0x31) {'1'} depending on the set bit.

#### 4.19 : Reset commands

Command : Hard reset      **H**

The hard reset command restarts an individual microcontroller by a

jp 0000h      ;jp to start of program

This will reset all variables in RAM to the default state in EPROM and clear any flags set. The axis will be invalid until a reset encoder command is sent to the processor.

Command : Reset encoders    **R**

The reset encoder command validates the position of the axis by moving the axis positively until the system OK flag has been set. This means that all encoders on that axis have passed through a reference sync. point. Resetting the encoders will invalidate the current telescope position.

#### 4.20 : PC clone control program

The physical processes embedded into the telescope system decrease the operator skills required to control the telescope. The high level message passing through the telescope network enables the controlling software to map each specific operator input to a corresponding output. This mapping confirms that the control system is distributed around the telescope network, rather than being centralised at the operators console.

The program used to interface the operator to the telescope system is viewed by the operator as an application for the computer. This application uses devices such as disk, video and keyboard. For these to be compatible with other programs and computers, a common operating system must be used.

The choice of operating system defines how the application interacts with the host computer, and defines how the application is constructed, displayed and executed.

Table 12 lists the major operating systems currently used for the PC market.

<b>Operating system</b>	<b>Task / user restrictions</b>
OS2	Pre-emptive multitasking, single user
Windows	Non pre-emptive multitasking, single user
DOS	Single task, single user
UNIX	Pre-emptive multitasking, Multi user

Table 12 : Common operating systems

Each operating system is designed for a different environment. The correct choice of the operating system enables the programmer to concentrate on the task involved, rather than the limitations of the operating system.

### **Disk Operating system (DOS)**

DOS was the first operating system developed for the Personal Computer (PC) market, when the PC user relied on 64K RAM, 8086 and one floppy disk. The fundamental design parameter declared by IBM was to enable PC hardware to progress while retaining the ability to execute any program / data file written for an earlier PC.

This design parameter assured the user of a open ended, upgradeable system and governed the enormous success of the PC but due to the inherent restrictions has limited the software. These limitations include single tasking environments, no windows, icon, mouse and pointing (WIMP) technology, front end graphical user interfaces (GUI) and multimedia support.

### **Windows**

In response to the growing demand for WIMP / GUI / multimedia support alongside the advantages of multitasking, Microsoft produced Windows, currently version 3.1. This non-pre-emptive multitasking environment with WIMP / GUI / multimedia support enabled programmers to harness the processing power of the modern PC. This was achieved by treating each peripheral as an abstract object, enabling powerful modern languages such as C/C++ to realise the full potential of the computer system.

Cheap, modern applications with a common structured presentation enabled the user to minimise their application learning curve allowing more efficient working practices. Front end GUI programming overheads are supported by Windows, maximising the programmer's application development time. The advantages of

multiple instances, multi-threaded code could be exploited without incurring the corresponding memory deficit by DISCARDABLE, LOADONCALL memory management commands.

The combined use of globally locked memory, dynamic data exchanges (DDE) and object linking and embedding (OLE) gives fast interprocess communication, allowing simple modules to coalesce and execute a complex task which would otherwise require a long and complex structure.

Sophisticated memory management and message queues confirmed the move away from linear programming used in DOS to a message (event) driven architecture, loading small portions of code only when required.

## OS2

IBM realised the limitations of DOS and developed the operating system 2 (OS2), which was similar to Windows but was a true pre-emptive multitasking environment. Whilst more rigorous in its design, it has currently failed to gain significant portions of the market due to its late product launch relative to Windows 3.0, and the rare and expensive compilers. Nevertheless, it is currently gaining a good reputation as an operating system for large, structured modules and will be a position to rival Windows NT for the preferred PC operating system.

## UNIX

UNIX (PC versions are called XENIX) is a text based multi-user, multitasking environment. The operating system is used for workstations where multiple users are present. In this single user context, it requires an excessive use of memory and hard disk and is not applicable.

Thus the Windows environment was the chosen operating system for the telescope. The prototype modules were supported by Windows 3.0, but it was found to be unstable and so was upgraded to Windows 3.1. This version operates without significant system faults.

Windows 3.1 has one major problem. A non pre-emptive multitasking environment does not employ a task switching algorithm to define processing time slices for each task, but relies on the currently active module to relinquish control and return processing to the Windows environment. In the majority of instances the module procedure will be executed only when a message is present in the module message queue. This message is processed by the module and on completion, the procedure ends, returning control to the Windows environment. In the case of a programming error such as the execution of an infinite loop, the Windows environment has no way to regain control, and the entire system 'hangs'. This problem is bypassed in Windows NT, which supports a pre-emptive multitasking environment.

Other problems associated with the Windows operating system are minor. The early compilers with Windows support compiled under DOS, and so increased the development time, as each module version required a change in operating system. Changing from a linear programming to a reactive message oriented architecture required a very steep learning curve by the programmer. These problems have been minimised by modern, Windows based compilers with extensive context sensitive on-line help.

#### **4.21 : Design of the telescope control program**

The telescope control program is the only interface the operator has with the telescope. Since the operator is an astronomer, not a computer programmer, the interface must be seen to be minimal.

General design parameters constricting the module development were outlined.

- Ease of use

The operator shall understand all data that is on the screen. The presentation of the system variables shall be in a format that requires no further processing by the operator.

An example of the is the filter command. The network message 'F6' actually means 'blank reference filter', so the modules must interpret and modify the requested and received data to the desired format.

- Large and comprehensive help index

The Windows operating system has a context sensitive help module, which can be embedded into other modules by the WinHelp(...) function call. This help file shall give quick solutions to common problems which can be fixed by the operator.

- Enable batch files

The operator shall be able to run batch files to minimise standard telescope configuration time.

- User control at all levels

The operator shall have control of the telescope at all levels. Messages at all levels shall be sourced from the command line input, and displayed when applicable.

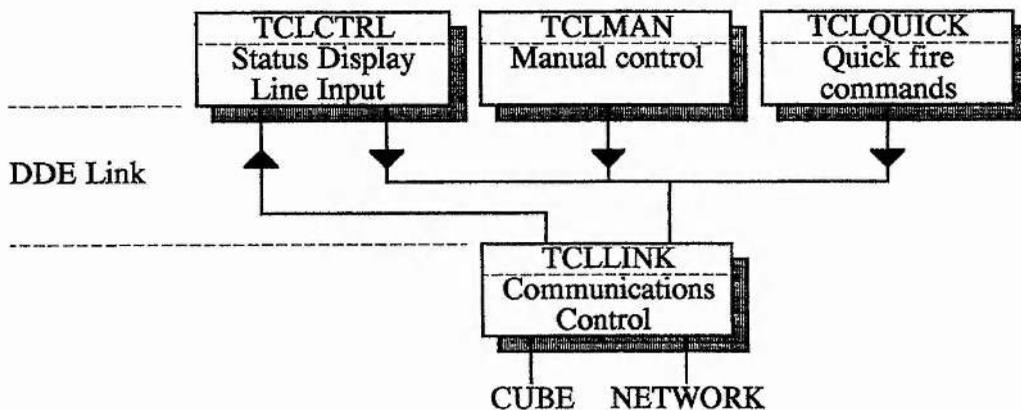
- Fast fault finding time

The operator shall be responsible for minor fault finding. As an extension of the help file, solutions to common faults shall be installed.

The program can be spilt into different distinct tasks.

- Communication with both the telescope and CUBE networks
- Status display
- Command line input
- Manual control input

A module was designated to each task. Communications throughout the modules were implemented by DDE links, enabling modules to link together and work cooperatively (Flowchart 12). This methodology produced small modules that were simple to debug and fast to compile. Various techniques were used to simplify the programming of each module. Maximum use of WIMP / GUI support limited the operator to valid instruction commands, producing robust modules which were very stable. If a batch file was to be executed, drag-drop functions bypassed the need for error and file checking by each module. A logical file list relating user modification files to module formats gave disk independence and multi-user support.



Flowchart 12 : *Windows 3.1* DDE link map

## 4.22 : Module TCLQUICK

### Application:

The module **TCLQUICK** displays up to 15 user defined buttons which can be activated to initiate system messages to **TCLINK** through the DDE link and finally to the networks.

### Parameters:

The button labels and associated system messages are defined in the logical file **SCL\_TCLQUICK\_CONFIG** using the following protocol:

```
< Button text [40] > < Whitespace > < Network text [40] > Z < cr >
```

This configuration file is loaded at module execution. An extra terminator ('Z') is appended to the network text string to signify the end of each network statement.

### Errors:

A logical file, **SCL\_TCLQUICK\_LOG**, is opened for recording module errors. This can be used to analyse run time errors such as incorrect start-up or file non-existence. Associating **SCL\_TCLQUICK\_LOG** extensions (default is \*.ERR) files with the Windows **NOTEPAD.EXE** module enables the user to view this file whilst controlling the telescope. Table 13 lists the error statements encountered from **TCLQUICK**.

<b>FILE : SCL TCLQUICK LOG</b>			
<b>Error text</b>	<b>Error type</b>	<b>Description</b>	<b>Solution</b>
Executing File	Information	Started TCLLINK Module	
Executing command < Command >	Information	Command requested	
Terminating module	Information	Module terminating (end of program)	
Invalid initialisation file	Fatal	Cannot read <b>SCL_TCLQUICK_CONFIG</b>	Check : INITIATE.TCL for <b>SCL_TCLQUICK_CONFIG</b> entry, path existence.
Module interlock unavailable	Fatal	Cannot execute TCLLINK module	Check : INITIATE.TCL for <b>SCL_TCLLINK_EXE</b> entry, path existence.
System fragmented	Fatal	Cannot find DDE address of TCLLINK module	Commonly linked with Module Interlock unavailable. If ONLY error, Windows environment unstable. Restart.

Table 13 : TCLQUICK error codes

#### 4.23: Module TCLMAN

##### Application:

The multi-instance module **TCLMAN** enables the interactive user to manually control any axis on the telescope using an array of push buttons organised as a 3\*3 matrix. Configuration of both axis selection and speed control for each individual instance is selected by the pull-down menu (Table 14).

<b>Axis selected:</b>	<b>Reference tube</b>	<b>Offset tube</b>	<b>Dome</b>
Available speeds:	Slew Set Guide	Set Guide	Selected by push- button

Table 14 : Available speed modes for different axes

By defining a different axis for each created instance gives the operator manual control over the entire telescope system. The front push-button display has been standardised for reference and offset tubes, but owing to the nature of the dome axis, a separate display format was used (Diagram 19), allowing the user to control the speed by the front push-buttons. Activating a push-button will request the microcontroller to start moving the required axis, performing all electro-mechanical logic. Other axes are not affected. Depressing 'STOP' will request the microcontroller to stop moving all axes.

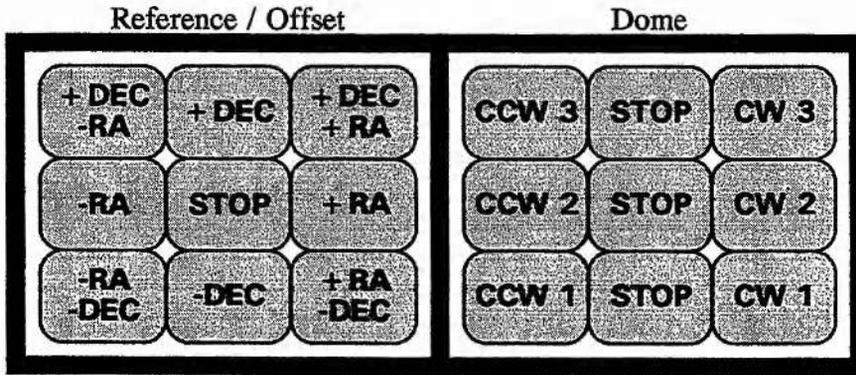


Diagram 19 : Reference, Offset and Dome push-button layouts

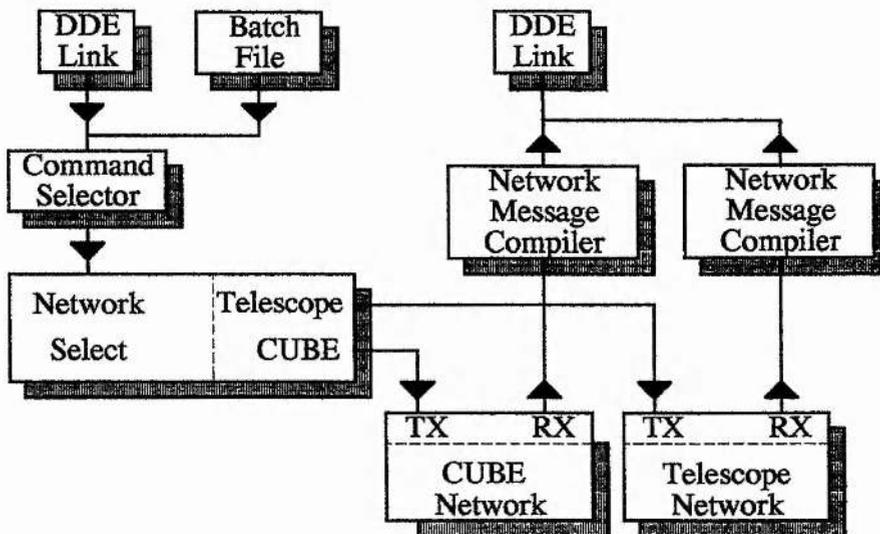
As with the **TCLQUICK** module, the network messages are sent by the DDE link to the **TCLLINK** module, and then onto the system network. The only difference is that the network messages are embedded into the program code, not in a \*\_CONFIG file.

#### 4.24 : Module TCLLINK

##### Application:

The module **TCLLINK** controls all the communication to and from telescope, CUBE networks and controlling computer. Multiple network commands sourced from individual modules or batch files are compressed into one network text ATOM and sent to the **TCLLINK** module from the DDE link for transmission. The network text received is then split into individual commands and transmitted to the network specified by its target address. (Flowchart 13).

Upon receiving characters from the network, the **TCLLINK** module compiles a network message. When a terminator character is received, the network message is sent to the **TCLCTRL** module, where it is decoded.



Flowchart 13 : TCLLINK routines

The **TCLLINK** module executes as a background task, and should be left in its minimised state. The title text bar is used to display the current status of the module (Table 15).

Title text	Description
TCLLINK:INTERLINK	Module idle
TCLLINK:INTERLINK[..]	Transmitting statement [..] from DDE link
TCLLINK:UPDATE	Initiating batch file
TCLLINK:..	Executing batch file ..

Table 15 : TCLLINK title text

Batch files are executed by dragging a file icon from the file manager over the module icon.

Errors:

Upon execution, the logical file **SCL\_TCLLINK\_LOG** is opened. Table 16 lists the error statements available.

FILE:SCL TCLLINK LOG				
Error text	Error type	Description	Solution	
Set Timer Failed	Fatal	Cannot find a free timer	Kill a tertiary module which uses a timer.	
OpenComm:COMx failed	Non-Fatal	Port unable to be opened	Check port existence :Kill modules using specified port.	
BuildCommDCB:COMx failed	Non-fatal	DCB string invalid	Reset DCB string to initial state.	
SetCommState:COMx failed	Non-fatal	Requested port state invalid	Check: Modified: port hardware, Comms BIOS,DDL,DCB string.	
Terminating Module	Information	Module terminating		
WriteCom failed	Warning	Cannot write data to port	Check hardware fault : Comms BIOS / DLL modifications. Displayed if port initialisation failed.	
UNKNOWN TARGET	Warning	Target specified does not exist	Network statement address invalid. Track to source.	
DROFFILE:FILE ..	Information	Batch file has been started		
INTERLINK:MESSAGE ..	Information	DDE message received		
INTERLINK RESUMED	Information	Batch file has ended normally		

Table 16 : TCLLINK error codes

Command Field 1	Command Field 2	Command Field3	Description
Point	Ref	<Hrs> <Min> <Sec> <Deg> <Min> <Sec>	Points telescope to given location
	Off	<Hrs> <Min> <Sec> <Deg> <Min> <Sec>	Offsets offset telescope to location relative to reference telescope
	Dome	<Degrees> 0 <Degrees> <359	Sets dome to <Degrees> azimuth
Delay	<Time>		Delays execution of next command for <Time> Seconds
Wait			Suspends batch file and prompts user to resume
Filter	Ref	<UMF_REFERENCE_FILTER> <Position> 0 <Position> <7	Sets the reference filter
	Off	<UMF_OFFSET_FILTER> <Position> 0 <Position> <7	Sets the Offset filter

	<b>Both</b>	<UMF_BOTH_FILTER> <Position> 0 <Position < 7	Sets both filters
<b>Aperture</b>	<b>Ref</b>	<UMF_REFERENCE_APERTURE> <Position> 0 <Position < 7	Sets reference aperture
	<b>Off</b>	<UMF_OFFSET_APERTURE> <Position> 0 <Position < 7	Sets offset aperture
	<b>Both</b>	<UMF_BOTH_APERTURE> <Position> 0 <Position < 7	Sets both apertures
<b>Centre</b>	<b>Dec_Ref</b>		Centres Reference DEC tangent arm
	<b>Ra_Off</b>		Centres Offset R.A. tangent arm
	<b>Dec_Off</b>		Centres Offset DEC tangent arm
<b>Reset</b>	<b>Ra_Ref</b>		Locks Reference R.A. encoders
	<b>Dec_Ref</b>		Locks Reference DEC encoders

	<b>Ra_Off</b>			Locks Offset R.A. encoder
	<b>Dec_Off</b>			Locks Offset DEC encoder
	<b>Dome</b>			Locks DOME encoder
	<b>All</b>			Locks all encoders
<b>Camera</b>	<b>Ref</b>			Selects Reference camera
	<b>Off</b>			Selects Offset camera
	<b>Finder_Ref</b>			Selects Reference finder camera
	<b>Finder_Off</b>			Selects Offset finder camera
	<b>&lt;Time&gt;</b>			Sets camera exposure time
	<b>0 &lt;Time &lt; 5000</b>			
<b>System</b>	<b>Ra_Ref</b>		<b>&lt;Encoder Offset&gt;</b>	Sets reference R.A. composite encoder offset
			<b>0 ≤ Encoder Offset ≤ 65535</b>	
	<b>Ra_Off</b>		<b>&lt;Encoder Offset&gt;</b>	Sets reference DEC composite encoder offset
			<b>0 ≤ Encoder Offset ≤ 65535</b>	
	<b>Dec_Ref</b>		<b>&lt;Encoder Offset&gt;</b>	Sets offset R.A. encoder offset
			<b>0 ≤ Encoder Offset ≤ 65535</b>	

<b>Dec_Off</b>	<i>&lt; Encoder Offset &gt;</i> $0 \leq \text{Encoder Offset} \leq 65535$	Sets offset DEC encoder offset
<b>Command</b>	<i>&lt; Network Command &gt;</i>	Sends a low level command to the network
<b>LED</b>	<i>&lt; Level &gt;</i> $0 \leq \text{Level} \leq 15$	Sets tube I.R. brightness
<b>Sky</b>	<i>&lt; Step &gt;</i> $0 < \text{Step} < 200$	Offsets aperture to sky position Sets sky position
<b>Star</b>		Resets aperture to star position

Table 17 : TCLCTRL commands

## 4.25 : Module TCLCTRL

The module **TCLCTRL** allows the user to enter high level commands (Table 17) from a line editor, execute high level batch files and display the telescope status in the module window.

Input commands can be sourced from 3 locations:

1. Interactive user input

The command line edit box at the top of the module workspace enables the operator to enter high level commands interactively as and when required.

2. Batch file input

The high level batch file consists of one or more of the above commands (Table 17). This file can be created by using **NOTEPAD.EXE**, with one command per line. It is loaded into **TCLCTRL** by the drag-drop function identical to **TCLLINK**. The **TCLCTRL** batch file is for use with the high level command set, and so must not be confused with the **TCLLINK** batch files, which consist entirely of network messages.

3. Network input

The module **TCLLINK** transfers network commands originating from the telescope back into **TCLCTRL** to be analysed by the module. This is transparent to the operator.

Other inputs used.

The three push buttons (Table 18) at the top of the module give the operator quick commands to both the module and the telescope in case of malfunction.

Button	Description
Abort Batch	Terminates the current batch file
Abort Source	Clears current commands line edit box
Stop	Stops the telescope. Identical to Stop on TCLMAN

Table 18 : TCLCTRL push buttons

### Module display

The module displays all data required for the correct running of the telescope (Diagram 20, Diagram 21). All data displays at this level require no interpretation for the operator, and enable them to recognise errors quickly. Each variable entry, including the descriptive text is assigned a child window superimposed onto the module workspace. This enables the operator to quickly update a field by selecting its child window. This action displays the current options by linking the selection address to the logical file or internal data range. The use of colour gave the operator quick identification of the variable status (Table 19).

Colour	Status
BLACK	Variable valid and accepted
RED	Network error in processing last variable update
GREEN	Transmitting variable update. Waiting for request acknowledge
BLUE	Processing variable update. Waiting for process acknowledge

Table 19 : Variable colour status

STARTUP		
	REFERENCE	OFFSET
R.A.	UNDEFINED	UNDEFINED
DEC	UNDEFINED	UNDEFINED
DOME	0	
FILTER	UNDEFINED	UNDEFINED
APERTURE	UNDEFINED	UNDEFINED
STAR/SKY	STEPS	0
CAMERA	UNDEFINED	
INTEGRATION TIME	0	
SIDEREAL	0	

Diagram 20 : TCLCTRL Initial display

FILTER BOTH RED		
	REFERENCE	OFFSET
R.A.	12 06 45	01 23 44
DEC	+67 56 20	-00 55 13
DOME	23	
FILTER	BLUE	BLUE
APERTURE	25ARC_SEC	25ARC_SEC
SKY	STEPS	20
CAMERA	REF	
INTEGRATION TIME	3000	
SIDEREAL	01 46 52	

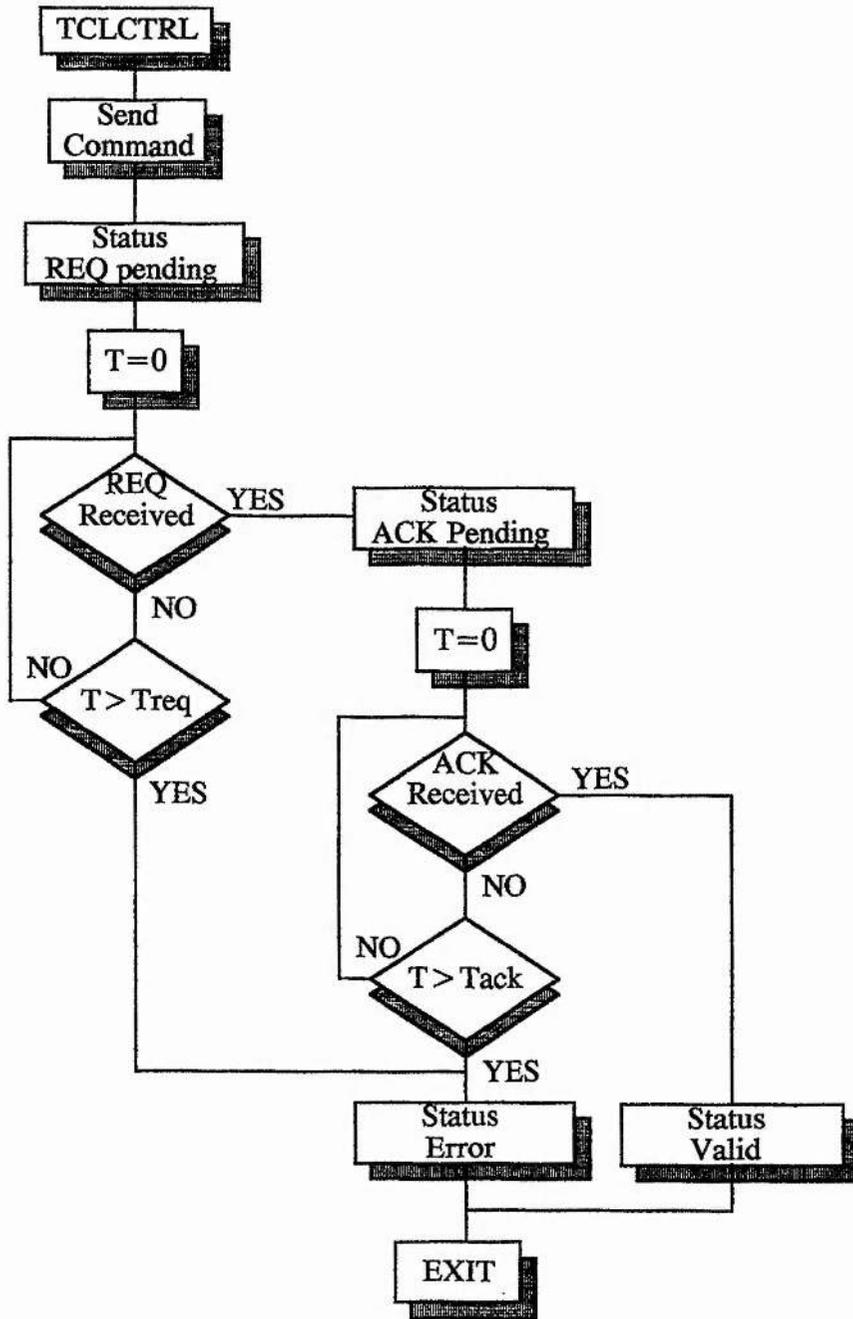
Diagram 21 : A TCLCTRL operational display

#### Module Operation.

The module itself does no operation as it utilises message generation from both networks and operator to initiate further messages to both the operator and the network.

An input from the edit box is filtered to extract each command field and associated data. Upon correct identification of command field/s the data is checked for validity and is encoded onto a network message template. This message is then sent to **TCLLINK** via the DDE link.

After the message is transmitted, one or more of the telescope microcontrollers processes the message, and executes the command responding with the **\_RX\_** and **\_ACK\_** network messages at the appropriate processing stage. While the network message is sent to **TCLLINK**, **TCLCTRL** initiates two sequential watchdog time-out sequences for both request acknowledge and process acknowledge handshaking messages. These sequences can be interpreted (for clarity) using a sequential flowchart (Flowchart 14).



Flowchart 14 : Sequential interpretation of message acknowledgement

**Errors:**

The logical file **SCL\_TCLCTRL\_LOG** is opened upon execution to record events occurring in **TCLCTRL** (Table 20).

Error text	Error Type	Description	Solution
System fragmented	Fatal	Cannot find DDE address of TCLLINK module.	See TCLQUICK
Module Interlock Unavailable	Fatal	Cannot execute TCLLINK module.	See TCLQUICK
Executing module	Information	Executing TCLCTRL.	
Interactive	Information	Batch file has ended. Control returned to the operator.	
Executing drop file ..	Information	Batch file .. started.	
Terminating module	Information	Module terminating (end of program).	
Interlink Message <..>	Information	DDE message .. received from TCLLINK.	
Abort Source	Information	'Abort Source' Push-button pressed. Batch file terminated.	
Abort Command	Information	'Abort Command' push-button pressed. Edit box cleared.	
Stop	Information	'Stop' push-button pressed. Telescope stopped.	
Interactive Command <..>	Information	Processing command ..	

Table 20 : TCLCTRL error messages

## **Chapter 5 : Conclusion**

This chapter describes the present status of the St Andrews twin photometric telescope. It outlines the improvements that are required to fully commission the telescope for use by observers. It presents an overall view of the project.

## **5.1 : Overview**

The aim of this project was to design, develop and implement a new control system for the St Andrews twin photometric telescope. A distributed control system architecture was selected with regard to cost and performance factors (Section 2.4). This architecture was implemented using multiple microcontrollers linked to a central console by a common network. A collection of Windows modules were designed for use by the operator at the central console. Windows was chosen because of its graphical multitasking environment, whose event driven architecture corresponded with the requirements of the control system architecture (Section 4.20).

The control system was validated in August 1993. A set of tests, initiated from various modules, confirmed that the telescope functions operated correctly. The telescope control system was not fully commissioned by the project completion date. Several improvements are required before the telescope can be used by observers. This chapter discusses the status of the telescope control system as of 1st October 1993 and outlines the improvements required to complete the commissioning of the telescope.

## **5.2 : Board design analysis**

All boards were constructed and tested in the laboratory. Minor corrections were required for some boards as a result of design modifications during production. All boards were subjected to a 24 hour burn in period and no failures were detected in this time. Each interface unit was then assembled, and the composite unit was then subjected to another 24 hour burn in period. Again no failures occurred, but

excessive power was dissipated by both power board regulators. These regulators were therefore mounted on the steel enclosure to sink the generated heat.

Each unit was installed at the telescope and subjected to an 8 hour burn in period. Faults developed during this period and were located to mains borne spikes and multiple earthing levels present around the telescope interacting with each unit. Board modifications to minimise these problems were undertaken using opto isolators on all input and output devices. Analogue input hysteresis levels were also increased to reject false triggering.

Suggested board improvements are listed below.

1. Input buffers should be exchanged for edge triggered devices to provide low noise levels across the I/O board and a more precisely defined input capture time.
2. All input buffers should be latched by a common read pulse to minimise skew across the inputs of the I/O board. This would also provide a more precise input capture time.
3. All relay drive circuitry should be protected by a CPU fail signal, driven from a watch dog timer. When CPU power fail occurs, the relay power should be disconnected by a relay.
4. A hard reset should occur when a CPU board is plugged into an active power unit. This must be independent of the power board status.
5. The power board must be adequately filtered to stop mains borne spikes reaching and damaging the microcontroller. This was a regular occurrence which was sourced to the hydraulic pump. Inadequate filtering produced a false /NMI interrupt signal from the LM311 comparator, resetting all the microcontrollers. This comparator was disconnected pending the construction of the solid state actuator.

6. The manual override should be buffered before connecting to the I/O board.
7. Better pointing accuracy could be found by ramping the motors by using the unit software. This option is present on the CPU and I/O boards, but would require additional software.
8. An extra backplane connector should be installed as standard to enable logic analysers easy access to the I/O bus.

### **5.3 : Board software analysis**

The board software was compiled quickly using the techniques described in chapter 4. The use of the logic analysers and promulators significantly reduced development time and revealed rogue transient glitches which caused program failure in the prototype. Function implementation was correctly embedded into the unit and each unit was successfully tested in all conditions. The common kernel program with device specific drivers produced robust code that minimised axis development time. The only limitation of the microcontrollers was the processor environment. Programming overheads were inevitable to support multiple tasks, and in a few routines CPU capture was present. The transputer environment would have been more favourable.

Various controller software improvements are listed below.

1. CPU capture was present in tangent arm centring and in low level motor control logic. These two routines should be integrated into the main program loop allowing error escape commands to disable all functions.
2. The `rx_int` receiver interrupt routine required a synchronising network command to clear any previous error in the network. The routine should automatically clear after receiving a packet terminator character.

3. A RST 0x38H should be called on stack overflow. At 0x38H the TXREQ line should be asserted, resetting all microcontrollers. Although stack overflow did not occur in the final version, this facility would enhance the program integrity. Since the op code for 0x38H is 0xFFH which corresponds to an invalid memory space, this is very easy to implement.

#### **5.4 : Tertiary system analysis**

The dome hydraulic motor actuator was tested on a small motor and was found to operate reliably, indicating a good glitch free operation. The installation is still to be completed.

#### **5.5 : Telescope control analysis**

The telescope controls are affected by both sensors and actuators. The new control system has overhauled all sensors and actuators, and improvements from the previous control system have been made in all areas. Visible improvements include the clamp unwinding, position setting and dynamic stability of each axis. System improvements include reductions in electromagnetic interference (EMI), reductions in RFI noise generation and lower power requirements for control implementation.

One major improvement to the fine setting motors is required.

These motors now operate at a slower maximum speed than in the previous control system. This is due to a lower armature voltage. While the PWM motor driver can handle higher armature voltages, the driver unit was designed to operate up to the maximum specified voltage of the motor. I was hesitant to exceed the manufacturers recommendations as this could reduce the motor lifetime.

## **5.6 : Windows modules**

The Windows modules were designed and developed to a rudimentary level.

Each module was developed and tested on both development and console computers. Under all situations each module worked well and was stable with regard to the operating environment throughout the development period. When multiple modules (or multiple instances) were executed, clean DDE message passing and correct telescope control were observed.

The Windows operating system provided benefits for both programmer and operator. The module programming was simplified by relieving the programmer of the GUI programming overheads. The event driven architecture of the Windows environment corresponded and interfaced well to the message driven architecture of the distributed control system.

A test operator found the system easy to use by providing user entry through push buttons and pull down menus laid out in a structured form. All processing was transparent to the user and is only interrupted when an error occurs. The comprehensive on line help file with keyword search and error referencing gives the user quick solutions to complicated error conditions.

Further module improvements are required to eliminate various fault conditions and to provide support for other modules.

The development of the Windows modules were the last to be programmed. Various software developments would be required to bring the telescope to a fully operational condition.

1. Create a comprehensive DDE link routine to enable the modules to link with other commercial software, such as databases and spreadsheets. This requires each module to support multiple handles specifying different module instances. Enhancing the use of the DDE link would bypass the WinExec function which was used to determine the module handle. Since the telescope module title bars are dynamic, the WinExec function occasionally fails.
2. The general layout of the display could be improved to provide a clearer user interface. While type face and layout are secondary to the telescope control, one of the project's aims is to provide the operator with a friendly environment.
3. While the status of each microcontroller could be inferred by the network messages, a management module should be included to monitor the status of each node. This would give the user an indication of a failed node as the fault occurred, rather than when a requested command time-out expired.
4. A timer message should be constructed to provide the TCLCTRL module with an automatic update of the telescope position and CUBE status. This message would initiate a global network message to read the present position of each axis. Upon reply a Windows message would then decode the position and display it in the status box.

### **5.7 : Concluding remarks**

The major restriction on the project was financial. Product selection was based primarily on cost rather than a consideration of various factors including task application, design standards, compatibility and product lifetime. Examples of this can be found in chapters 3 and 4 where the application has been modified to reduce the capital outlay to such an extent that the project aim was compromised.

To minimise the effect of this constraint the bulk of the software, test equipment and hardware was lent or donated to the project by other research groups and companies. Since all construction from board level through to installation was undertaken by myself, this restricted the amount of time available for the development of the telescope control system.

This thesis describes the design of a control system which is cheap, intelligent and open ended. It can correspond well with modern operating environments and hardware. It takes an alternative view of co-operative interaction between intelligent objects and tries to emulate that interaction in a physical device. This project was not driven by an abstract concept, but by repeated observation of people in motion : the efficiency of teamwork overcoming the orders of the one.

Distributed environments are becoming commonplace for telescope control systems. Different topologies are used for differing classes of telescope based on control requirements, physical size, performance and cost. This thesis describes the novel design of a distributed control system for the twin photometric telescope and how the design can easily include unique requirements such as a secondary telescope.

## References

- (1) Nise N. S., 1992, Control Systems Engineering, Blackwell, Redwood City USA
- (2) Chen P. C., 1983, PASP, 95, 332
- (3) Grauer A. D., Bond H. E., 1981, PASP, 93, 388
- (4) Sorvari J. M., 1975, PASP, 89, 443
- (5) Reddish V. C., September 1966, Sky and Telescope, 124
- (6) Elston R., Zeilik M., 1982, PASP, 94, 729
- (7) Linnel A. P., Hill S. J., Brandt E. F., Sorvari J. M., 1974, PASP, 87, 273
- (8) Rambold W. N., Stilburn J. R., 1987, PASP, 100, 393
- (9) Golyas Yu. E., Tikhonov V. A., 1983, UDC, 681, 139
- (10) Honeycutt R. K., Kephard K. E., 1982, PASP, 24, 605
- (11) Boyd L. J., 1984, IAPPP symposium
- (12) Treffers R. T., 1985, PASP, 97, 446
- (13) Colgate S. A., Moore E. P., Carlson R., 1975, PASP, 87, 565

- (14) Colgate S. A., 1987, Santa Cruz Summer Workshop, Springer-Verlag, New York, 653
- (15) Genet R. M., Boyd L. J., 1985, Sky and Telescope, July, 16
- (16) Hayes D. S., Genet R. M., Crawford. D. L., 1990, Unpublished
- (17) Melsheimer F., 1987, Santa Cruz Summer Workshop, Springer-Verlag, New York, 732
- (18) Raffi G., Biereichel P., Gilli B., Gustafsson B., Roche J., Wirenstrand K., 1990, SPIE, 1235, 428
- (19) van Breda I. G., Parker N. M., 1983, Microprocessors and Microsystems, Butterworth & Co Ltd, vol. 7 no 5, 203
- (20) Nees W., 1987, Santa Cruz Summer Workshop, Springer-Verlag, New York, 612
- (21) van Breda I. G., Newton G. M., Johnson A. N., Waltham N R., 1990, SPIE 1235, 438
- (22) Waltham N R., van Breda I. G., Newton G. M., 1990, SPIE 1235, 328
- (23) Sinnott R. W., 1983, Sky and Telescope, April, 311

- (24) Lelièvre G., Waddell P., Sovka J., 1987, Santa Cruz Summer Workshop, Springer-Verlag, New York, 707
- (25) Smith D. H., 1985, Sky and Telescope, July, 23
- (26) Parker N. M., van Breda I., Martin R., 1983, SPIE, 445, 505
- (27) Jones L. R., 1990, SPIE, 1235, 374
- (28) Jones L. R., Branston D. M., 1990, SPIE, 1235, 383
- (29) Autoscope Corporation Newsletter, October 1989
- (30) Boyd L. J., Genet R. M., 1986, PASP, 98, 618
- (31) Bell S. W., Hilditch R. W., 1984, MNRAS, 211, 229
- (32) Bell S. W., Rainger P., Hilditch, R. W., 1990, MNRAS, 249, 632
- (33) Edwin R. P., Gears R. T., 1992, PASP, 104, 1234
- (34) Chubb B. A., 1967, Modern analytical design of instrument servomechanisms, Addison-Wesley, Reading Mass.
- (35) Hill P.W., O'Neill P., Lasker B., 1992, in Workshop on Robotic Observatories, Kilkenny
- (36) Linnel P. L., 1981, PASP, 93, 661

- (37) Claudius M., Florentin-Nielsen R., 1981, AA, 100, 186
  
- (38) Kielkopf J., Hinkle S., 1987, PASP, 99, 442
  
- (39) Proc SARA/IAPPP/Fairborn Symposium, 1993, Robotic Observatories: The first decade, AIP Conference Proceedings
  
- (40) Maxted P. F. L., Hill G., Hilditch R. W., 1994, A&A, 282, 821
  
- (41) Warnock I.G., 1989, Programmable Controllers: Operation and Application, Prentice Hall, Englewood Cliffs NJ.
  
- (42) Kissell T.E., 1986, Understanding and using Programmable Controllers, Prentice Hall, Englewood Cliffs NJ.
  
- (43) Swainston F., 1991, Systems Approach to Programmable Controllers, Nelson (Thomas), Melbourne
  
- (44) Chesmond C. J., Wilson P. A., Lepa M. R., 1991, Advanced Control System Technology, Whitaker
  
- (45) Bennett S., 1994, Real-Time Computer Control: An Introduction, Peregrinus, London
  
- (46) Virk G. S., 1991, Digital Computer Control Systems, Macmillan, London

- (47) Forsythe W., Goodall R. M., 1991, Digital Control, Macmillan education, Basingstoke
- (48) Jovic F., 1991, Process Control Systems : Principles of design and operation, Chapman and Hall, New York
- (49) Hartley M. G., 1962, An Introduction to Electronic Analogue Computers
- (50) Pabrai U., Dowat C., Carlson J., 1990, Understanding and Using Computer Networks, Fermi National Accelerator Laboratory, Illinois
- (51) Farnell Catalogue 1991 Page 568.

## **Acknowledgements**

I would like to thank the following people for contributing to this project:

Drs Reg Killean, Dave Robb and Matt Emberson for test equipment and good advice. The electronics workshop (Ian, Jim and Mike) for use of the plant, books and more helpful suggestions. The astronomy staff (Dr R. Hilditch, Dr P. Hill) for astronomical matters, Dr R. Edwin for supervision and SERC for financial support.

## **Appendix A : Source code for GAL devices**

The Generic Array Logic (GAL) ICs are located on the processor, I/O, sequential and bus controller boards. The ICs reduce the amount of packages required for sequential and clocked logic. All GAL devices were programmed using a Xender XP6005 universal programmer attached to a DCS80286 borrowed from Ealanta Technologies Ltd.

## A.1 : CPU memory and I/O logic decoder

Filename:	CPU.PLD
Application:	CPU memory and I/O logic decoder
Device used:	GAL 16V8 25nS / Replacement code RS 655-745

```
/* */
/* This is the GAL source program for the */
/* TWIN TELESCOPE CPU module */
/* */
```

```
DEVICE 16v8;
TITLE TWIN_CPU;
REVISION 001;
AUTHOR RICHARD GEARS;
COMPANY ST. ANDREWS UNIVERSITY;
DATE 05/05/92;
```

```
/* */
/* Now define the inputs from the Z84013 CPU */
/* */
/* System clock input ( 10Mhz) */
PIN 1 = CLK ; /* 10MHz CPU 0 deg clock */
```

```
/* Cpu addresses out - ACTIVE HIGH */
PIN 2 = A11 ; /* CPU address 11 */
PIN 3 = A12 ; /* CPU address 12 */
PIN 4 = A13 ; /* CPU address 13 */
PIN 5 = A14 ; /* CPU address 14 */
PIN 6 = A15 ; /* CPU address 15 */
```

```
/* CPU bus control out - ACTIVE LOW */

PIN 7 = MREQ ; /* Memory request out */
PIN 8 = WR ; /* CPU write out */
PIN 9 = RD ; /* CPU read out */
```

```
/* */
/* Now define the outputs to external devices and memory */
/* */
```

```
/* All outputs are ACTIVE LOW */

PIN 11 = GEN ; /* GAL chip enable */
PIN 12 = REN ; /* RAM chip enable */
PIN 13 = ROU ; /* RAM output enable */
PIN 14 = RWR ; /* RAM write into memory */
```

```

PIN 15=    EEN  ;    /*    EPROM chip enable    */
PIN 16=    EOU  ;    /*    EPROM output enable    */
PIN 17=    IWR  ;    /*    IO port write pulse    */
PIN 18=    IRD  ;    /*    IO port read pulse     */
PIN 19=    BEN  ;    /*    Device to data bus en.  */

/*
/*    Now define the power pins
/*
PIN 20=    VCC  ;    /*    +5V
PIN 10=    GND  ;    /*    GROUND (0V)

/*
/*    Now define the equations
/*
/*    first define the EPROM equations, EEn and EOU
/*    The EPROM used is a 16K * 8, 150nS, 0 wait state,
/*    Vpp = 12.5V, mnfr = TI, Farnell order code
/*    TMS27C128-15JL, compatible with the 27C64 type
/*    PROVIDED that the access times et are as high or higher
/*
/*    The EPROM is mapped initially to direct address space
/*    0000h - 3FFFh as a continuous, non-paged mapping

/*    Enable EPROM over memory range with memory
/*    request active low

!EEN = !A15 & !A14 & !MREQ      ;

/*    Enable EPROM output over same memory range but
/*    with Cpu reading

!EOU = !A15 & !A14 & !MREQ & !RD ;

/*    now define the RAM equations, REN, ROU, RWR

/*    The RAM used is a 2K * 8 6116A 120ns 0 wait state,
/*    mnfr = UMC, low power static RAM Farnell order code
/*    UM61162L, but is NOT compatible with anything else
/*    I know of (I think)
/*
/*    the RAM is mapped initially to direct memory space
/*    A000h - A7FFh as a continuous, non-paged memory
/*    Enable RAM over memory range with the memory
/*    request active low

```

```

!REN = A15 & !A14 & !A13 & A12 & A11 & !MREQ ;

/*      Enable RAM output over same range but with the      */
/*      CPU reading (low)                                     */

!ROU = A15 & !A14 & !A13 & A12 & A11 & !MREQ & !RD;

/*      enable memory in RAM to be written while CPU      */
/*      is writing (low)                                     */

!RWR = A15 & !A14 & !A13 & A12 & A11 & !MREQ & !WR ;

/*      The output pins IOREAD, IO WRITE are selected by this */
/*      chip as well. All outputs are buffered, so the bus select */
/*      output places the external bus on the CPU data bus for */
/*      reading as you've probably guessed, do this at the wrong */
/*      time and its fish supper for the CPU i.e. BANG goes the */
/*      CPU so if in doubt, leave it high ( all data moves from */
/*      CPU to BUS                                           */

/*      Enable external devices to read the data bus, IWR      */

!IWR =      A15 & A14 & A13 & A12 & A11 & !MREQ & !WR ;

/*      Enable external devices to write onto the data bus,IRD */

!IRD =      A15 & A14 & A13 & A12 & A11 & !MREQ & !RD ;

/*      Enable the CPU to read data from external data bus      */
/*      yep - this is the one that you DON'T want to screw up */

/*      The bus flow is controlled by a 74HCT245, The ports are */
/*      port  A      =      External bus                        */
/*      port  B      =      CPU data bus                       */
/*                                                                */
/*      BEN ( bus enable ) defines the flow direction. This is: */
/*      BEN LOW : PORT B => PORT A                            */
/*                                                                */
/*      { CPU => EXTERNAL } */
/*      BEN HIGH: PORT A => PORT B                            */
/*                                                                */
/*      { EXTERNAL => CPU } */
BEN =      A15 & A14 & A13 & A12 & A11 & !MREQ & !RD ;
/*      End of TWINCPU.PLD source code                        */

```

## A.2 : Motor enable and direction encoder

Filename:	MOTOR.PLD
Application:	Motor enable / direction encoder limit cut-out
Device used:	GAL 16V8 25nS / Replacement code RS 655-745

```
/* */
/* This is the GAL source program for the */
/* TWIN TELESCOPE MOTOR SELECT */
/* */
```

```
DEVICE 16v8;
TITLE MOTOR_SELECT;
REVISION 001;
AUTHOR RICHARD GEARS;
COMPANY ST. ANDREWS UNIVERSITY;
DATE 27/05/92;
```

```
/* */
/* Now define the inputs from the IO BOARD */
/* */
```

```
PIN 1 = CLK ; /* Not used - combinational */
PIN 2 = MEA ; /* Motor Enable A */
PIN 3 = MEB ; /* Motor Enable B */
PIN 4 = LAL ; /* Limit A low hit */
PIN 5 = LAH ; /* Limit A High hit */
PIN 6 = MDA ; /* Motor Direction A */
PIN 7 = MDB ; /* Motor Direction B */
PIN 8 = LBL ; /* Limit B low hit */
PIN 9 = LBH ; /* Limit B high hit */
```

```
/* */
/* Now define the outputs to the CA3262E Relay driver chip */
/* */
```

```
PIN 12= AEOU ; /* Motor A driver out */
PIN 13= ADOU ; /* Motor A direction out */
PIN 14= NCA ; /* Not connected */
PIN 15= NCB ; /* Not connected */
PIN 16= BEOU ; /* Motor B driver out */
PIN 17= BDOU ; /* Motor B direction out */
PIN 18= NCC ; /* Not connected */
PIN 19= NCD ; /* Not connected */
```

```
/* */
/* Finally, define the power supply and the control */
/* */
```

```

/*      signals for GAL      */
/*      */
PIN 10=   GND ;      /*      0V GND      */
PIN 20=   VCC ;      /*      5V VCC      */

/*      */
/*      Define the equations for the GAL according to      */
/*      the scenario below      */
/*      */
/*      First, Motor A and Motor B are Identical in nature,      */
/*      so the equations are identical. This is also true for both      */
/*      GAL's on the IO board, so to produce a new board, two      */
/*      identical GAL's need to be programmed      */
/*      */
/*      Conditions for operation:      */
/*      While both limits are low, the motor can be enabled in      */
/*      both directions      */
/*      If one OR other limits is set, the motor can only be      */
/*      in the opposite direction, enabling a reverse from error      */
/*      If both limits are set disable the motor completely.      */
/*      An error has occurred - physically, both limits cannot be      */
/*      set at the same time      */

ADOUT =   MEA & MDA & !LAH      ;

AEOUT =   MEA & !LAL & !LAH
          |   MEA & MDA & !LAH
          |   MEA & !MDA & !LAL      ;

BDOUT =   MEB & MDB & !LBH      ;

BEOUT =   MEB & !LBL & !LBH
          |   MEB & MDB & !LBH
          |   MEB & !MDB & !LBL      ;

/*      End of MOTOR.PLD source code      */

```

### A.3 : Network controller and driver

Filename:	BUS_CTRL.PLD
Application:	Network monitor and driver
Device used:	GAL 16V8 25nS / Replacement code RS 655-745

```

/* */
/* This is the GAL source program for the */
/* TWIN TELESCOPE bus control */
/* */

```

```

DEVICE 16v8;
TITLE BUS_CTRL;
REVISION 002;
AUTHOR RICHARD GEARS;
COMPANY ST. ANDREWS UNIVERSITY;
DATE 20/05/92;

```

```

/* */
/* Define the inputs from lines, external inputs */
/* clock input is unused */
/* */

```

```

PIN 1 = SOK ; /* Sensor OK */
PIN 2 = CPU ; /* CPU power ONLINE */
PIN 3 = STBY ; /* Standby power ON (mains) */
PIN 4 = PWR ; /* 415V ONLINE */
PIN 5 = RST ; /* Reset button (Act. low) */
PIN 6 = STOP ; /* Stop (Low), Resume (High) */
PIN 7 = LINE ; /* Interface power line OK */
PIN 8 = TXRX ; /* TX RX data */
PIN 9 = TXREQ ; /* TXREQ line (Act. high) */
PIN 11 = REQERR ; /* error request (Act. High) */

```

```

/* */
/* Define the outputs */
/* */

```

```

PIN 12 = LINERR ; /* Excessive voltage drop on */
/* CPU power line */
PIN 13 = SYSRST ; /* Initiate system reset */
PIN 14 = SYSERR ; /* Display this led if things */
/* aren't OK */
PIN 15 = BUSERR ; /* Protocol error - tx'ing */
/* whilst TX REQ inactive high */
PIN 16 = CPUERR ; /* CPU requests shutdown */
PIN 17 = CPURLY ; /* Logic to CPU line relay */

```

```

PIN 18=    PWRRLY ; /* (active low) */
/* Logic to 415V relay */
PIN 19=    SYSOK ; /* (Active low) */
/* System OK and running */

/*
/* Now define the power pins
/*

PIN 20=    VCC ; /* +5V */
PIN 10=    GND ; /* GROUND (0V) */

/*
/* Now define the equations
/*

/* BUS error. this error signal is set when the TXRX line is
/* active low but the TXREQ line is inactive high. This is a
/* minor error as a noisy line could produce glitches on the
/* txrx line. BUT... I would not expect this to be set.
/* Show it anyway
*/

BUSERR = TXRX & !TXREQ & CPU ;

/* LINE error. this signal is not that important, as again
/* I can see it not affecting the running of the telescope in
/* some special circumstances.
/* But, again, I would not expect it to light.
*/

LINERR = !LINE & CPU ;

/* system error. This error signal shows when one or more
/* errors occur and that the operator should really find out
/* what is wrong. Link in any error conditions to this signal,
/* as it acts as a catch all
*/

SYSERR = TXRX & !TXREQ & CPU
        | !LINE & CPU
        | !REQERR & CPU ;

/* CPU error. This error signal indicates that one or more of
/* the interface units has crashed or is requesting a system
/* shutdown. This signal is generated by a watchdog timer
/* on each Cpu timing the looping of the statement interpreter.
/* If the loop exceeds the Specified delay, the TXREQ line
/* is held low until the watchdog timer is reset, generating
/* an unusually long active low pulse on the TXREQ line,
*/
*/

```

```
/*      much longer than any of the transmissions used. A CR      */
/*      circuit on the BUS board times out, generating this signal */
```

```
CPUERR = !REQERR & CPU      ;
```

```
/*      System reset: This LED tells the user that the system is  */
/*      shutdown and that, to get the telescope working again,    */
/*      they must press the reset button on the bus housing.      */
/*      Pressing the reset button starts the CPU power line, and  */
/*      the CPU power line should (all being well) clear whatever */
/*      set the system into reset mode. Some hope!!!             */
```

```
SYSRST = !CPU      ;
```

```
/*      CPU relay. Right, if things are playing up system wide  */
/*      and you can't understand what is going on, understand   */
/*      what this does, otherwise your stuck up a large creek  */
/*      without alot of hope.                                    */
/*      The system works like this, under these rules:          */
/*      1) if there is a serious error, shutdown the 415V motor */
/*      relays, that way they don't screw up the system more than */
/*      it has already has...                                    */
/*      2) if there is a very serious error, switch off the interface. */
/*      3) if the system clears it's error's,                   */
/*          DONT restart the interfaces automatically            */
/*      4) switch on the interfaces by pressing reset           */
/*      5) Switch on the 415V motor power when the CPU power */
/*          is stable                                           */
```

```
/*      Note that there is NO registered logic present in the GAL, */
/*      for good reason too, the stability of the system is      */
/*      generated by the integrity of the system dynamically, so it */
/*      will automatically shut down if an error occurs.        */
/*      Better safe than sorry...                                */
```

```
CPURLY = RST & STBY & !CPUERR ;
```

```
PWRRLY = RST & CPU & STOP      ;
```

```
/*      system OK: Yep this signal tells everyone that the system */
/*      is up and running and that there should be no cause for   */
/*      concern in terms of system integrity                       */
```

```
SYSOK = CPU & PWR & STBY      ;
/*      End of BUS_CTRL.PLD source code                          */
```

## Appendix B : Files used in the compilation of the microcontroller software

The Z80 include files hold data, common routines and compiler directives for all Z80 source files. These are linked at compile time through the #include statement. The filename extension denotes the file type:

*.H	Header files including definitions, macro declarations, excluding all source code
*.PRE	Compiler pre-processing directives
*.REL	Z80 source code which has been compiled separately, but is common to all programs

## **B.1 : Include file for character definition**

Filename:	character.h
Application:	Include file for use in Z80 microcontrollers

;This file sets up the Symbolic labels for comparisons for the main program

;set up the ASCII character set

```
chr_null      .equ  00h  ;
chr_bell      .equ  07h  ;
chr_txt_st    .equ  02h  ;
chr_tx_st     .equ  01h  ;
chr_tx_end    .equ  04h  ;
chr_text_end  .equ  03h  ;
chr_cr        .equ  13   ;
chr_lf        .equ  0ah  ;
chr_space     .equ  ' '  ;
chr_I         .equ  'I'  ;
```

;now define alternative codes for rdu units

```
ra_main       .equ  'A'  ;
dec_main      .equ  'B'  ;
ra_offset     .equ  'C'  ;
dec_offset    .equ  'D'  ;
counter       .equ  'E'  ;
dome          .equ  'F'  ;
```

;End of character.h include file

## **B.2 : Include file for mapping of external I/O**

Filename:	map_io.h
Application:	Include file for use in Z80 microcontrollers

```
;This file sets up the io locations and offsets;
base_io      .equ  0fff0h ;define the base address of the io channels
encoder_io   .equ  0fff2h ;encoder input
motor        .equ  00h   ;motors 1 to 8 address = FFF0h (write)
direction    .equ  01h   ;direction address = FFF1h (write)

;End of map-io.h include file
```

### **B.3 : Include file for mapping internal memory variables**

Filename:	memory.h
Application:	Include file for use in Z80 microcontrollers

```
;This file sets up the locations in ram (and ROM) for variable storage,  
;buffers etc.
```

```
; set up the ROM locations for the encoder and statements  
encoder_def .equ 03000h
```

```
; Now specify the ram locations
```

```
ram_top .equ 09FFFh  
ram .equ 09800h  
stack .equ 09D00h ;stack goes up memory!!
```

```
; specify buffers
```

```
tx_buffer .equ (ram + 0000h) ;256 byte for text  
rx_buffer .equ (ram + 0101h) ;255 byte for text  
rx_cursor .equ (ram + 0100h) ;cursor to get rx position  
rx_holding .equ (ram + 0201h) ;holding register  
holding_flag .equ (ram + 0200h) ;holding flag
```

```
; specify ram locations for system variables
```

```
present .equ (ram + 0300h) ;16 bit (2 byte) Present encoder position  
future .equ (ram + 0302h) ;16 bit (2 byte) future encoder position  
offset .equ (ram + 0304h) ;16 bit (2 byte) offset (future-present)  
h1 .equ (ram + 0306h) ;16 bit (2 byte) slew encoder hysteresis  
h2 .equ (ram + 0308h) ;16 bit (2 byte) set encoder hysteresis  
h3 .equ (ram + 030Ah) ;16 bit (2 byte) guide encoder hysteresis  
offsetfine .equ (ram + 030Ch) ;16 bit (2 byte) cyclic fine 'coder offset  
delay_mech .equ (ram + 030Eh) ;16 bit (2 byte) Mechanical relay delay  
delay_inert .equ (ram + 0310h) ;16 bit (2 byte) telescope inertia delay  
delay_arb .equ (ram + 0312h) ;16 bit (2 byte) arbitration time delay  
delay_out .equ (ram + 0314h) ;16 bit (2 byte) TXREQ released delay  
delay_over .equ (ram + 0316h) ;16 bit (2 byte) overload sensor delay  
delay_turn .equ (ram + 0318h) ;16 bit (2 byte) delay for clamp rewind  
delay_cap .equ (ram + 031Ah) ;16 bit (2 byte) delay for capacitive line  
sid_cont .equ (ram + 031Ch) ;16 bit (2 byte) sidereal interrupt divider  
newencoder .equ (ram + 031Eh) ;16 bit (2 byte) sidereal interrupt divider  
oldencoder .equ (ram + 0320h) ;16 bit (2 byte) sidereal interrupt divider  
intcount .equ (ram + 0322h) ;16 bit (2 byte) interrupt divider entry  
intcountcpu .equ (ram + 0324h) ;16 bit (2 byte) CPU usage intcount  
coarse .equ (ram + 0326h) ;16 bit (2 byte) sidereal interrupt divider  
slewflag .equ (ram + 0328h) ;slew flag 1=slewing, 0=stopped  
okflag .equ (ram + 032Ch) ;ok=1
```

```
moveflag      .equ  (ram + 032Ah)      ;automove (computer moving scope=1)
fineflag      .equ  (ram + 032Eh)      ;fine flag 1 = fine motors on
offset_dir    .equ  (ram + 0330h)      ;direction of error
slew_dir      .equ  (ram + 0332h)      ;direction of slewing motor
limitflag     .equ  (ram + 0333h)      ;limit status
dome_direction.equ  (ram + 0334h)      ;incremental axis direction
```

```
;End of memory.h include file
```

## **B.4 : Include file for statement field definition (prototype only)**

Filename:	statement.h
Application:	Include file for use in Z80 microcontrollers

```
;This file sets up the offset locations for the buffers;
valid_stat .equ 0 ; 1st byte holds S if statement valid
limita_off .equ 1 ; holds valid if limits off (bank a)
limita_on .equ 2 ; holds valid if limits on (bank a)
limitb_off .equ 3 ; holds valid if limits off (bank b)
limitb_on .equ 4 ; holds valid if limits on (bank b)
presentvl .equ 5 ; holds valid if present > presentvl
presentvh .equ 9 ; holds valid if presentvh > present
offsetvl .equ 13 ; holds valid if offset > offsetvl
offsetvh .equ 17 ; holds valid if offsetvh > offset
motor_on .equ 21 ; holds valid if motors on
motor_off .equ 22 ; holds valid if motors off
motor_num .equ 23 ; selects the motor to use
motor_alg .equ 24 ; selects the speed and direction
idle .equ 25 ; selects whether idle/or stop required
```

```
;End of statement.h include file
```

## **B.5 : Include file for Z84013/015 internal peripheral mapping**

Filename:	rambo.h
Application:	Include file for use in Z80 microcontrollers

```
;          Z84015 SET UP LABELS
;this include file holds all the pointers to the Z84015 I/O map
;Set up counter chip (CTC)
ctc_ctrl_0   .equ  010h
ctc_ctrl_1   .equ  011h
ctc_ctrl_2   .equ  012h
ctc_ctrl_3   .equ  013h

;set up serial port (SIO)
sio_data_a   .equ  018h
sio_ctrl_a   .equ  019h
sio_data_b   .equ  01Ah
sio_ctrl_b   .equ  01Bh

;set up watch dog timer
watch_data   .equ  0F0h
watch_ctrl   .equ  0F1h

;set up interrupt priority register
irq_priority .equ  0F4h

;End of rambo.h include file.
```

## **B.6 : Include file to set up internal peripherals**

Filename:	setup_io.rel
Application:	Include file for use in Z80 microcontrollers

```
; Default set up procedure for Z84015 IO
; Set up counter / timer, (bit 0 denotes ctrl byte)
; set the interrupt control word to low byte of int_table

    io_ctrl(ctc_ctrl_0,(int_table_ctc & 0f8h)
    io_ctrl(ctc_ctrl_0,00110111b)

; set up ctc, Ch. 0 internal 200Hz (nominally - for any multitasking)

    io_ctrl(ctc_ctrl_0,195)

; No int., timer mode, /256, rising edge, autotrig, add const, free run

    io_ctrl(ctc_ctrl_1,00010111b)
    io_ctrl(ctc_ctrl_1,32)                ;Ch. 1 Tx/rx baud rate

;ctc_1 = 16*1200 baud (Tx/rx clock)
; No int., timer mode, /16, rising edge, autotrig, add const, free run
; use this timer to get 30Khz interrupts

    io_ctrl(ctc_ctrl_2,10000101b)
    io_ctrl(ctc_ctrl_2,220)

; int., timer mode, /256, rising edge, autotrig, add const, free run,1Khz
    io_ctrl(ctc_ctrl_3,00110111b)

;Fine guide modulation
    io_ctrl(ctc_ctrl_3,2)

;ctc_3 = 1/2 speed if used
;No int., counter mode, nc, rising edge, nc, add const, free run
;counters set up for general use, now enter time delay constants
;clocks now set up, now onto SIO
;
;    SIO UNIT
;
;set up the SIO to give 9600 baud async. mode on channel A set up channel
;a only. write register 4 MUST be set up first, and then everything else
;Note that sio_ctrl(x,y) macro loads y to write register x
;reset channel A#: do this just to clear any duff data in the regs.
```

```
sio_ctrl(0,00011000b)
```

```
;As Before, set up write register 4 before anything else  
;rx Clk *16 baud rate, 8bit sync., 1 stop, no parity
```

```
sio_ctrl(4,01000100b)
```

```
;Write register 1 determines the interrupt status of the SIO  
;this is set up to give the following:  
;d0: disable interrupts from DCD/CTS/SYNC [0]  
;d1: disable transmitter interrupts [0], there is a polled routine for Tx  
;and async int's would foul up the protocol.  
;d2:Status does NOT affect vector [0] this returns a fixed vector,  
;independent on any SIO setting. vector in WR2.  
;d3/d4: Interrupts on first rx character, parity error is not a special  
;condition. [01]  
;d5/d6/d7: Wait ready config. set to [000], not used.
```

```
    sio_ctrl(1,00011000b)  
;    sio_ctrl(1,00001000b)
```

```
;Write register 2 sets the interrupt vector load into  
;sio channel B
```

```
ld    a,2  
out   (sio_ctrl_b),a  
ld    a,int_table_sio & 0ffh  
out   (sio_ctrl_b),a
```

```
;Write register 3 sets the logic control  
;d0:rx disable (for now), as it will be enabled as the end of setup [0]  
;d1:Inhibit sync. char load. Not used so set to [0]  
;d2:Address search mode off. [0]  
;d3:rx CRC checking off. IBM does not have CRC checker [0]  
;d4:auto enter hunt phase off. Not used [0]  
;d5:Autoenables off. automatically switches CTS/DCD [0]  
;d6/d7:Rx bits per character 8bit format, so [11]
```

```
sio_ctrl(3,11000000b) ;set up clock/parity/stop
```

```
;Write register 5 holds data that control the transmitter  
;d0:tx CRC enable : off, see above [0]  
;d1:RTS, Switch off for now, used later [1]  
;d2:CRC-16/SDLC ;Yep, well, This is to do with the polynomial for the  
;CRC error checker. Switch off[0]  
;d3:transmit enable. Definitely on. [1]  
;d4:Send Break. Definitely off.[0] (sends a High break to line)
```

;d5/d6:tx bits per character 8bit format, so [11]  
;d7:DTR set, switch off this output for now.[0]

```
sio_ctrl(5,01101000b) ;set up transmitter
```

; Write register 6,7 relate to the Mono/by sync. modes. switch to 0

```
sio_ctrl(6,00000000b)  
sio_ctrl(7,00000000b)
```

; This sets up the interrupt priority daisy chain, with SIO having  
; a high priority and SIO unit low.[xxxxxx01]

```
io_ctrl(irq_priority,00000001b)
```

;set up the watch dog control

;The watchdog timer - if it runs out of time, i.e. the program has  
;crashed will pull the Tx request line low. Held low for more that 0.5s  
;without returning high will cut the 415V and 230V lines - shutting down  
;the telescope. The telescope will be UNSTABLE and must be settled manual  
;disable it if it is a prototype

```
io_ctrl(watch_data,00000011b)  
io_ctrl(watch_ctrl,0b1h)
```

;End of setup\_io.rel include file.

## **B.7 : Include file to configure SCMA assembler**

Filename:	scma.pre
Application:	Include file for use in Z80 microcontrollers

;This file sets up the scma assembler to optimise all variables

.title "Microcontroller unit ver. 3.1"

.nopcode

.list

.sym

; define yes and no for directives

yes .equ 1

no .equ 0

;End of scma.pre include file.

## Appendix C : Z80 Source Code

The device dependant source files contain the code for each microcontroller. It was compiled using the SCMA cross assembler / compiler which encoded the object code to INTEL memory format. During compilation the include files were inserted (and compiled) and useful diagnostic MAP, LST, SYM files were produced.

The batch file used to invoke the SCMA compiler was:

```
C:\design\scma\scma -80 -g0 -c -f -l -h -s -p -a15 c:\design\scma\z80\%1.z80
```

providing the correct start-up conditions (directories, file existence) were met.

Code generation and correction time was dramatically decreased by the use of a EPROM emulator and logic analyser.

The SMART Communications PROMULATOR loaded up to 64K of Z80 code through the parallel link taking an average loading time of 5 seconds. The features such as automatic reset generation, KB loading, run, halt indicators and battery backup made this tool indispensable. The code was downloaded through the LD3.EXE package after problems with older LD.EXE program, invoked by the batch file:

```
c:\design\prom\ld3 c:\design\scma\z80\%1.obj/i/16k
```

Two logic analysers were used to analyse program execution: a Tektronix 64\*20Mhz and ST512 48\*50Mhz. The Tektronix became faulty due to age and a suspected lightning strike and was replaced by the ST512. This analyser was ideal for hardware / software debugging, only requiring the add on op code disassembly package to complement the test gear.

The compiled code was initially downloaded to EPROM using the Lloyd programmer, downloading the data through a serial link. Due to a fused memory IC onboard the programmer it was replaced by (two) Xender XP6005 programmers, the first one having fused due to the incorrect insertion of a GAL (upside down). Briefly, this took out the output FET's to the inserted IC pins, and a DCS80386SX motherboard. The DCS machine was claimed to be improperly designed (by Xender).

## C.1 : Source code for R.A. axis microcontroller

Filename:	RA.Z80
Application:	Source code for R.A. axis Z80 microcontroller

```
;
;
; ST. ANDREWS TWIN PHTOMETRIC TELESCOPE
; R.A. MAIN AXIS DRIVE PROGRAM
; 150993V5.0/R.T.GEARS/TPT-TECH.DOC
;
; Load the include files
host      .equ      ra_main      ; define host as ra_main
#include   "c:\\design\\scma\\z80\\set_scma.pre"
#include   "c:\\design\\scma\\z80\\rambo.h"
#include   "c:\\design\\scma\\z80\\char.h"
#include   "c:\\design\\scma\\z80\\map_io.h"
#include   "c:\\design\\scma\\z80\\memory.h"
#include   "c:\\design\\scma\\z80\\macro_io.def"
;*****
start     .org      0000h
          di                ;Processor startup code
          im      2         ;interrupts from table
          ld      hl,stack   ;load stack from memory.h
          ld      sp,hl      ;and load
          ld      a,(int_table&0ff00h) >> 8 ;load I reg with table addr.
          ld      i,a        ;load
          call   io_setup    ;setup on chip peripherals
          jp     start_main  ;and go
;*****
          .org      038h
rx_test:
          push   af
          push   bc
          push   de
          push   hl
          push   ix
          push   iy
rx_test1:
          sio_in(0)         ;wait for character to be received
          bit    0,a         ;mask off
          jp     z,rx_test1 ;and loop if not yet available
rx_test6:
          ld     ix,rx_buffer ; get start of buffer
          ld     iy,rx_cursor ; get cursor pointer
          ld     a,(rx_cursor) ; get cursor
          ld     b,a
```

```

rx_test4:
    inc    ix                ; move ix to rx cursor
    djnz  rx_test4         ; and repeat
    dec   ix                ; correct for initial loop
rx_test3:
    sio_in(0)              ; is there a character to be received??
    bit   0,a               ; mask off
    jp    z,rx_test2       ; no, so exit
    in    a,(sio_data_a)   ; yes, so load data
    ld    (ix+0),a         ; and store in rx_buffer
    cp    'Z'              ; end of message (chr_cr)
    jp    nz,rx_test5      ; carry on if not end of message
    ld    a,1              ; reset cursor
    ld    (iy+0),a         ; and load iy into rx_cursor
    ld    a,(rx_buffer)    ; is it for the host??
    cp    host             ; use host
    jp    nz,rx_test7     ; no, so just delete
    ld    a,(rx_buffer+1)  ; is the remote host the IBM??
    cp    'I'
    jp    nz,rx_test7     ; no,so delete
    ld    de,rx_holding    ; load holding reg with rx buffer
    ld    hl,rx_buffer     ; (de) <= (hl) for bc times
    ld    bc,30            ; repeat 30 times
    ldir
    ld    a,'I'            ; signal new command
    ld    (holding_flag),a
rx_test7:
    ld    ix,rx_buffer     ; clear up
    call  clear_buffer     ; clear rx_buffer
    jp    rx_test2        ; and exit
rx_test5:
    inc   (iy+0)           ; next location
    jp    rx_test6        ; and loop
rx_test2:
    sio_ctrl(0,038h)      ; send a return from interrupt
    sio_ctrl(0,020h)     ; and enable on next rx character
    pop   iy
    pop   ix
    pop   hl
    pop   de
    pop   bc
    pop   af
    ei
    reti
;*****
ctc_int:
    push  af                ; save registers used

```

```

push bc ; push all registers on stack
push de
push hl
push ix
push iy
ld ix,intcount ; point to scaler dividers in memory
dec (ix+2) ; decrement scaler 1
jp nz,ctcint1 ; not ready for processing,so exit
ld a,(ix+0) ; reload divider
ld (ix+2),a ; and store back in register
;time to update present encoder reading (approx 10 updates/sec)
ld a,(0FFF0h) ; low byte of fine encoder from I/O
ld l,a ; and store (as part of twin reg. hl)
ld a,(0FFF1h) ; now get high byte of fine encoder
and 007h ; mask out all but low 3 bits
ld h,a ; and store
ld (newencoder),hl ; new fine encoder now stored in ram
; Now, has it cycled itself?? i.e. is
; update < 0 (CCW) or update > 2^10 (CW)
ld de,(oldencoder) ; get old encoder value
ld a,0 ; clear direction flag
and a ; clear carry flag
sbc hl,de ; get update value
jp nc,ctcint2 ; carry not set, so direction +ve
ld a,1 ; set direction flag
ld hl,(oldencoder) ; swap registers
ld de,(newencoder) ; to clear carry on subtraction
and a ; clear carry
sbc hl,de ; and subtract
ctcint2:
ld de,0100h ; if delta |encoder| > 2^9 sync passed
sbc hl,de ; compare, and if no sync carry set
jp c,ctcint3 ; no sync found, so compile encoder
cp 1 ; sync +ve or -ve, check sync flag
jp z,ctcint4 ; direction +ve, so load coarse from I/O
ld a,(coarse) ; else load coarse from memory
dec a ; and coarse -1
ld (coarse),a ; and store back again
jp ctcint3 ; and compile encoder
ctcint4:
ld a,(0FFF2h) ; load coarse encoder
and 01Fh ; mask out all unused bits
ld (coarse),a ; and store
ld a,1 ; and show that position os OK
ld (okflag),a ; store
ctcint3:
ld hl,(newencoder) ; get new position

```

```

    ld    (oldencoder),hl    ; store into old position
    ld    a,(coarse)        ; get coarse position
    sla  a                  ; shift encoder 3 times to top location
    sla  a
    sla  a                  ; OK
    or   h                  ; overlay fine and coarse encoder
    ld   h,a                ; and store back again
    ld   (present),hl      ; and store in present
;get offset (works - so don't touch it)
    ld   hl,(present)      ; compare present to future
    ld   de,(future)      ; to get direction of offset
    and  a                  ; clear carry flag
    sbc  hl,de              ;
    jp   c,calcoff1        ; jump if future < present
    ld   (offset),hl       ; save offset in memory
    ld   a,0                ; and set direction to ??
    ld   (offset_dir),a
    jp   ctcint5
calcoff1:
    ld   hl,(future)      ; here swap future and present registers
    ld   de,(present)     ; and here too
    and  a                  ; clear carry flag
    sbc  hl,de
    ld   (offset),hl      ; save offset in memory
    ld   a,1                ; and invert direction
    ld   (offset_dir),a
;calcoff ends here
ctcint5:
    dec  (ix+3)            ; decrement scaler 2
    jp   nz,ctcint1        ; not ready for processing, so exit
    ld   a,(ix+1)          ; reload divider chain again
    ld   (ix+3),a          ; and store
    ld   hl,(future)      ; increment future position
    inc  hl                 ; 16 bit increment
    ld   (future),hl      ; and store back again
ctcint1:
    ; exit routine here
    pop  iy
    pop  ix
    pop  hl                ; clear up and exit
    pop  de
    pop  bc
    pop  af
    ei                      ; and enable interrupts
    reti                   ; and exit
;*****
unused:
    ei

```

```

    reti
;*****
;
    .org    (0ff00h & ($ + 0ffh))
io_setup:
#include    "c:\\design\\scma\\z80\\setup_io.rel"
    ret
;*****
;
    .org    (0ff00h & ($ + 0ffh))
;Interrupt table in here
int_table:
int_table_etc:
    .byte   unused & 0ffh
    .byte   (unused & 0ff00h) > > 8
    .byte   unused & 0ffh
    .byte   (unused & 0ff00h) > > 8
    .byte   ctc_int & 0ffh
    .byte   (ctc_int & 0ff00h) > > 8
    .byte   unused & 0ffh
    .byte   (unused & 0ff00h) > > 8
    .byte   unused & 0ffh
    .byte   (unused & 0ff00h) > > 8
int_table_sio:
    .byte   rx_test & 0ffh
    .byte   (rx_test & 0ff00h) > > 8
;*****
;
    .org    (0ff00h & ($ + 00ffh))
start_main:
    call    setup                ;setup memory, ports default values
    ei                        ;enable interrupts
    sio_ctrl(5,01101000b)      ;testd
    sio_ctrl(0,020h)          ;enable rx on next character
    ld      ix,text_on         ;send node Ok message
    call    tx_request         ;and send
main_loop:
    call    movemon
    call    rx_mon
    call    print_text
    jp     main_loop
;*****
;
starttest:
    call    ret_clear          ; clear rx buffer
    ld      ix,diag1          ; all setup, so transmit
                                ; Diag 1 messages
    call    tx_request        ; tx
    call    print_text        ; tx message to network
    ld      hl,14904d         ; Delay startup 5 sec
    call    delay

```

```

ld    a,1                ; set direction CW
ld    (slew_dir),a      ; and store
ld    a,0
ld    (okflag),a        ; invalidate ok flag
call  slewstart
starttest1:
ld    a,(okflag)        ; wait for flag to be updated
cp    1
jp    nz,starttest1
call  slewstop
ld    hl,2981d          ; delay for 1 sec
call  delay
ld    ix,diag2          ; finished, so send next messages
call  tx_request        ; tx
ld    hl,(present)      ; lock present & future together
ld    (future),hl
jp    print_text        ; tx message to network
;*****
setup:
ld    a,0FFh            ; shut off all motors, whatever state
ld    (base_io+motor),a ; motors off
ld    (base_io+direction),a ; and direction relays
ld    ix,present
call  clear_reg
ld    ix,offset
call  clear_reg
ld    ix,future
call  clear_reg
ld    hl,0FA14h         ; load in int counters
ld    (intcount),hl    ; and store
ld    hl,00533h         ; Load Fine encoder offset default
ld    (h1),hl
ld    hl,40d            ; Load default set hysteresis
ld    (h2),hl
ld    hl,5d             ; Load default guide hysteresis
ld    (h3),hl
ld    hl,299            ; 100mS delay
ld    (delay_mech),hl  ; for mechanical delay
ld    hl,1491           ; 500mS delay
ld    (delay_over),hl  ; for telescope inertia
ld    hl,1491           ; 500mS delay
ld    (delay_inert),hl ; for overload inertia
ld    hl,1491 ;500mS delay
ld    (delay_turn),hl  ; for rewinding clamp 1/4 turn
ld    hl,000FFh        ; 85 mS arbitration delay
ld    (delay_arb),hl   ; for arbitration delay
ld    hl,000FFh        ; 85mS (seems like a good figure)

```

```

ld      (delay_out),hl      ; for tx inactive settling time
ld      hl,01 ;338uS
ld      (delay_cap),hl     ; for capacitive decay in comms line
ld      a,0                 ; Set slew flag to 0
ld      (slewflag),a
ld      (moveflag),a       ; and switch move to manual
ld      (okflag),a        ; invalidiate encoder ok flag
ld      (limitflag),a     ; clear limits
ld      a,1                ; Set cursor to start of rx buffer
ld      (rx_cursor),a
ld      ix,rx_buffer
call   clear_buffer
ld      ix,rx_holding
call   clear_buffer
ld      a,'O'
ld      (holding_flag),a
ret

;*****
;sort out the commands here -scrap below
rx_mon:
ld      a,(rx_holding+2)
cp      'Q'
jp      z,testcode
cp      'M'
jp      z,moveint
cp      'D'
jp      z,dispres
cp      'P'
jp      z,inppres
cp      'R'                ;encoder reset, sync encoders
jp      z,starttest
cp      'H'                ;soft reset, start again
jp      z,start
ld      a,(holding_flag)  ; anything there??
cp      'I'                ; (indicated with 'I')
ret     nz                 ; exit if nothing
ld      a,'O'              ; or load O to invalidate it
ld      (holding_flag),a
ld      ix,text_rx
jp      tx_request

;*****
clear_reg:
ld      a,0
ld      (ix+0),a
ld      (ix+1),a
ret

;*****

```

vectorin:

```
ld    a,(rx_holding+3) ; vectorin routine load iy with the
select('P',present) ; address of the 16 bit (2 byte)
select('F',future) ; variable to be acted on or read out
select('O',offset) ; this holds all the data to be changed
select('S',h1) ; save for the move register, which
select('T',h2) ; is acted on manually
select('G',h3)
select('N',0FFF0h)
select('W',0FFF2h)
select('H',0FFF4h)
select('M',delay_mech)
select('I',delay_inert)
select('A',delay_arb)
select('U',delay_out)
select('K',okflag)
select('D',intcount)
ld    iy,0FFFEh ; sent to dud space
ret ; and return. this MAY NOT be a good idea.
```

\*\*\*\*\*

testcode:

```
ld    a,'P'
ld    (rx_holding+3),a
call  dispres
call  print_text
ld    ix,text_rtn
call  tx_request
call  print_text
jp    testcode
```

\*\*\*\*\*

movemon:

```
ld    a,(moveflag) ; is automove flag set??
cp    0 ; automatic if !=0
ret    z ; it isn't, so exit (slew off in moveint)
ld    a,(okflag) ; is unit OK??
cp    0 ; buggered if 0
jp    z,movemon1 ; yes, so stop all motors
ld    a,(offset_dir) ; else load offset direction to slew
ld    (slew_dir),a ; and store again
ld    hl,(offset) ; is slew hysteresis > offset
ld    de,(h1) ; h1 holds the slew hysteresis
and    a ; clear carry flag
sbc    hl,de ; subtract it and look for carry
jp    nc,slewon ; OK,so slew
call  slewoff ; no,so check slewing logic off
ld    hl,(offset) ; is fine set hysteresis > offset
ld    de,(h2) ; h1 holds the fine set hysteresis
```

```

and    a                ; clear carry flag
sbc    hl,de            ; subtract it and look for carry
jp     nc,seton        ; Ok, so set on
ld     hl,(offset)     ; is fine guide hysteresis > offset
ld     de,(h3)         ; h1 holds the fine guide hysteresis
and    a                ; clear carry flag
sbc    hl,de            ; subtract it and look for carry
jp     nc,guideon     ; Ok,so guide on
ld     a,0              ; reset automove flag
ld     (moveflag),a    ; save in register
ld     ix,text_ok      ; send message saying drive stopped
call   tx_request      ; and transmit

movemon1:
call   slewoff         ; switch slew off
jp     fineoff         ; else switch off everything
;*****
moveint:
ld     a,01h           ; preload automove as active
ld     (moveflag),a    ; and store
ld     a,(rx_holding+3) ; read in control byte
cp     'G'             ; automove - let computer go
jp     z,moveint1     ; yep, so return
ld     a,00h           ; automove has been canceled, so stop
ld     (moveflag),a    ; and store
call   slewoff         ; check that the slewing is off
ld     a,(rx_holding+3) ; read in control byte
ld     b,0FBh          ; preload with set -
cp     '1'             ; is it set- ('1')
jp     z,manmove      ; yes, so exit
ld     b,0FEh          ; preload with guide -
cp     '2'             ; is it guide- ('2')
jp     z,manmove      ; yes, so exit
ld     b,0F7h          ; preload with guide +
cp     '3'             ; is it guide + ('3')
jp     z,manmove      ; yes, so exit
ld     b,0FDh          ; preload with set +
cp     '4'             ; is it set + ('4')
jp     z,manmove      ; yes, so exit
ld     b,0              ; preload b with slew direction CW
cp     '9'             ; is it CW slew??
jp     z,moveint2     ; yes, so start
ld     b,1              ; preload b with slew direction CCW
cp     '8'             ; is it CCW slew??
jp     z,moveint2     ; yes, so start
ld     b,0FFh          ; stop if not

manmove:
ld     a,b              ; swap for load

```

```

        ld    (base_io+direction),a; and load
        jp    ret_com             ; and exit
moveint2:
        ld    a,b                 ; swap b=>a for load
        ld    (slew_dir),a       ; store in slew direction
        call  slewstart          ; start slewing
moveint1:
        ; exit routine.
        jp    ret_com             ; and exit
;*****
slewon:
; check that the slewing is on (by checking flag)
        ld    a,(slewflag)       ; get slew flag
        cp    1
        ret   z                  ; everything's going ok - so exit
        jp    slewstart          ;slew needs to be switched on
;*****
slewoff:
; check that the slewing is off (as before)
        ld    a,(slewflag)       ; get slew flag
        cp    0
        ret   z                  ; everything's stopped - so exit
        jp    slewstop
;*****
seton:
        ld    b,0F7h             ; preload b with fine motor guide+
        ld    a,(offset_dir)     ; get direction
        cp    0                  ; which direction??
        jp    nz,seton1          ; cw, so start cw motor
        ld    b,0FEh             ; ccw, so load b with guide-
seton1:
        ld    a,b                 ; swap for load
        ld    (base_io+direction),a
        ret
;*****
guideon:
        ld    b,0FDh             ; preload b with fine motor set+
        ld    a,(offset_dir)     ; get direction
        cp    0                  ; which direction??
        jp    nz,guideon1       ; cw, so start cw motor
        ld    b,0FBh             ; ccw, so load b with set-
guideon1:
        ld    a,b                 ; swap for load
        ld    (base_io+direction),a
        ret
;*****
fineoff:
        ld    a,0FFH             ; switch all fine inputs off

```

```

    ld    (base_io+direction),a
    ret
;*****
inppres:
    call  vectorin          ; set vector to destination
    call  clear_reg        ; clear register
    ld    ix,rx_holding+4  ; set pointer to start of 1's &0's
    ld    b,010h           ; set repeat loop for 16 (2 byte)
inppres1:
    ld    a,(ix+0)         ; load in bit to be tested
    cp    '1'              ; is it a '1'??
    scf                    ; preload with a carry set
    jp    z,inppres2       ; and if it is, start rolling
    and   a                 ; else clear flag
inppres2:
    rl    (iy+0)           ; rotate carry into memory location
    rl    (iy+1)           ; and through into high byte
    inc   ix                ; next rx holding buffer
    djnz  inppres1
    jp    ret_com
;*****
dispres:
    ld    ix,text_16       ; create 16 binary mask
    call  tx_request       ; and validate it for tx'ing
    call  vectorin         ; get input vector to load data from
dispres5:
    ld    ix,tx_buffer     ; set pointer to start of tx buffer
    ld    a,(iy+1)         ; load high byte of display buffer => a
    ld    b,8              ; set loop to shift 8 bits
dispres2:
    sla   a                 ; shift top bit into carry
    jp    nc,dispres1      ; if not '1' next shift
    inc   (ix+3)           ; it's a '1', so inc '0' => '1'
dispres1:
    inc   ix                ; next bit in tx buffer
    djnz  dispres2         ; repeat 8 times
    ld    a,(iy+0)         ; load low byte of display buffer => a
    ld    b,8              ; set loop to shift 8 bits
dispres4:
    sla   a                 ; shift top bit into carry
    jp    nc,dispres3      ; if not '1' next shift
    inc   (ix+3)           ; it's a '1', so inc '0' => '1'
dispres3:
    inc   ix                ; next bit in tx buffer
    djnz  dispres4         ; repeat 8 times
    jp    ret_clear

```

\*\*\*\*\*

```
slewestart:
    ld    a,0ffh          ; cancel sidereal drive
    ld    (base_io+direction),a
    ld    a,0ffh          ; switch clutch on
    ld    (base_io+motor),a
slewestart1:
    ld    a,(0fff1h)      ; is clutch on?
    bit   5,a
    jp    nz,slewestart1
    call  clampoff        ; clutch is on, unlock clamp
    ld    a,(slew_dir)    ; get offset direction
    srl   a                ; move into carry
    ld    a,0ffh          ; and set a to directiontemp
    rr    a                ; and rotate it back in
    ld    (base_io+direction),a
    ld    hl,(delay_mech) ; wait for relays
    call  delay
    ld    a,07fh          ; switch on motor
    ld    (base_io+motor),a
    ld    a,001h          ; flag to indicate slewing is on
    ld    (slewflag),a    ; and store
    ret
```

\*\*\*\*\*

```
clampoff3:
    ld    a,0ffh          ; overload, so stop motor
    ld    (base_io+motor),a
    ld    hl,(delay_mech) ; wait for relays
    call  delay
    ld    a,0ffh          ; reverse clamp in to build inertia
    ld    (base_io+direction),a
    ld    hl,(delay_mech) ; wait for relays
    call  delay
    ld    a,0bfh          ; switch motor back on
    ld    (base_io+motor),a
    ld    hl,(delay_turn)
    call  delay            ; about 0.5 turn
    ld    a,0ffh          ; switch motor off
    ld    (base_io+motor),a
    ld    hl,(delay_mech) ; wait for relays
    call  delay
clampoff:
    ld    a,0bfh          ; set direction
    ld    (base_io+direction),a
    ld    hl,(delay_mech) ; wait for relays
    call  delay
clampoff2:
```

```

ld    a,(0fff4h)      ; is clamp out?
bit   5,a
jp    nz,clampoff1   ; yes, so exit
ld    hl,(delay_over)
call  delay           ; delay to overcome overload
ld    a,(0fff1h)     ; is overload sensor active?
bit   6,a
jp    z,clampoff3    ; yes? knock out clamp and repeat
ld    a,0bfh         ; no, so switch clamp motor on

ld    (base_io+motor),a
jp    clampoff2

clampoff1:
ld    a,0ffh         ; clamp out, so motor off
ld    (base_io+motor),a
ld    hl,(delay_mech) ; wait for relays
call  delay
ld    a,0ffh         ; reset direction
ld    (base_io+direction),a
ret

;*****
slewstop:
ld    a,0ffh         ; stop slew motor
ld    (base_io+motor),a
ld    hl,(delay_mech) ; wait for relays
call  delay
ld    a,0ffh         ; reset contacts
ld    (base_io+direction),a
call  clampon        ; lock clamp on
ld    a,0efh         ; switch off clutch
ld    (base_io+motor),a
ld    a,00h          ; indicate that slewing has finished
ld    (slewflag),a
ret                  ; and exit

;*****
clampon:
ld    a,0ffh         ; set direction clamp in
ld    (base_io+direction),a
ld    hl,(delay_mech) ; wait for relays
call  delay

clampon2:
ld    a,(0fff1h)     ; is overload sensor active?
bit   6,a
jp    z,clampon1     ; yes? next routine
ld    a,0bfh         ; no, so switch on clutch
ld    (base_io+motor),a
jp    clampon2

```

```

clampon1:
    ld    hl,(delay_over)    ; wait for inertia to build up
    call  delay
clampon3:
    ld    a,(0fff1h)        ; is overload sensor active?
    bit   6,a
    jp    z,clampon4        ; yes? next routine
    ld    a,0bfh            ; no, so switch on clutch
    ld    (base_io+motor),a
    jp    clampon3
clampon4:
    ld    a,0ffh            ; it has hit the end, so stop motor
    ld    (base_io+motor),a
    ld    hl,(delay_mech)    ; wait for relays
    call  delay
    ld    a,0ffh            ; reset relays
    ld    (base_io+direction),a
    ret

;*****
; This routine does all the delays for the program, and all the delays can be
; loaded by the user at runtime. All default delays are loaded in setup at the
; start and then modified by user. NOTE: changing delay times can serious
; affect the performance of the system!!!. Think about it. delay time in uS
; delay time = 2.5+335.5*HL uS
delay:
    ld    de,01
delay1:
    ld    b,0ffh
delay2:
    djnz  delay2
    and  a
    sbc  hl,de
    jp   nz,delay1
    ret

;*****
ret_com:
    ld    ix,text_ack
    call  tx_request
ret_clear:
    ld    a,'O'
    ld    (holding_flag),a
    ld    ix,rx_holding
    jp   clear_buffer

;*****
clear_buffer:
    ld    b,250
    ld    a,0

```

```

clear1:
    ld    (ix+0),a
    inc  ix
    djnz clear1
    ret
;*****
tx_request:
    ld    iy,tx_buffer
tx_req2:
    ld    a,(ix+0)
    cp    chr_null
    jp    nz,tx_req1
    ld    a,chr_null
    ld    (iy+0),a
    ret
tx_req1:
    ld    (iy+0),a
    inc  ix
    inc  iy
    jp    tx_req2
;*****
; this routine holds all the protocol for transmitting a data
; packet, including txrx arbitration, etc
print_text:
    ld    ix,tx_buffer           ; get start of tx buffer
    ld    a,(ix+0)             ; get valid text line character
    cp    'S'                  ; is this character = valid??
    ret  nz                    ; no, so exit
    sio_in(0)                  ; get cts bit
    bit   5,a                  ; is cts set??
    ret  z                    ; txrx line active - so quit
    sio_ctrl(5,11101000b)     ; set txrx control active
    ld    hl,(delay_arb)
    call delay                 ; delay for a preset time
    sio_ctrl(5,01101000b)     ; set txrx control inactive
    ld    hl,(delay_cap)
    call delay                 ; delay for capacitor
    sio_in(0)                  ; get cts bit
    bit   5,a                  ; is cts set??
    ret  z                    ; yes, so get off line
    sio_ctrl(5,11101000b)     ; switch on txrx line for good
    sio_ctrl(3,11000000b)     ; shut off receiver
    sio_ctrl(1,00000000b)     ; switch off interrupts
print_t2:
    ld    a,(ix+0)             ; a < first character of line
    cp    chr_null             ; is it the end of the message
    jp    nz,print_t1         ; no, so print message

```

```

        ld    a,chr_null          ; end of message, so send null
        call print_a            ; and print it.
print_t3:
        sio_in(0)                ; get tx buffer empty
        bit   2,a                ; is tx buffer empty?
        jp    nz,print_t3       ; no, so wait
print_t4:
        sio_in(1)                ; get all_sent bit
        bit   0,a                ; is it all sent
        jp    nz,print_t4       ; no, so wait
        ld    a,'*'              ; signal with any char
        ld    ix,tx_buffer       ; like *
        ld    (ix+0),a           ; that message is sent
        ld    hl,(delay_out)
        call delay
        sio_ctrl(5,01101010b)    ; set txrx control inactive
        in    a,(sio_data_a)     ; clear any characters held
        in    a,(sio_data_a)     ; in holding buffer
        in    a,(sio_data_a)     ;
        in    a,(sio_data_a)     ;
        sio_ctrl(3,11000001b)    ; and switch on reciever
        sio_ctrl(1,00011000b)    ; and enable interrupts
        ret                       ; return
print_t1:
        call print_a            ; print character
        inc  ix                  ; increment pointer
        jp   print_t2           ; and repeat
;*****
; This routine must not be called to send a character only.
; it will violate the network - no TXRX control
print_a:
        ld    b,a                ; b <= a (store it temp)
print_1:
        sio_in(0)                ; get tx empty (bit status)
        bit   2,a                ; empty ??
        jp    z,print_1          ; no, so loop until it is
        ld    a,b                ; transfer character back to a
        out   (sio_data_a),a     ; and send it to the comms
        ret                       ; and return
;*****
text_on      .text  "SI"
             .byte  host
             .text  "NODE_RESET_ACK"
             txt_end

diag1       .text  "SI"
            .byte  host

```

```

        .text "SYSTEM_NODE_ACK"
        txt_end

diag2    .text "SI"
        .byte host
        .text "SYSTEM_TEST_OK"
        txt_end

text_ack .text "SI"
        .byte host
        .text ACK"
        txt_end

text_rtn .text "SI"
        .byte host
        .byte 10
        .byte 13
        txt_end

text_ok  .text "SI"
        .byte host
        .text "OK"
        txt_end

text_rx  .text "SI"
        .byte host
        .text "RX"
        txt_end

text_d .text "SI"
        .byte host
        .text "D000ACK"
        txt_end

text_input: .text "SI"
        .byte host
        .text "00000000ACK"
        txt_end

text_16: .text "SI"
        .byte host
        .text "0000000000000000ACK"
        txt_end

text_err .text "SI"
        .byte host
        .text "ERR"
        txt_end

;*****

        .end

```

## C.2 : Source code for DEC. axis microcontroller

Filename:	dec.z80
Application:	Source code for DEC. axis Z80 microcontroller

```
;
;
;   ST. ANDREWS TWIN PHTOMETRIC TELESCOPE
;   DEC MAIN AXIS DRIVE PROGRAM
;   150993V5.0/R.T.GEARS/TPT-TECH.DOC
;
; define axis address
host      .equ  dec_main      ; define dec iD
; Load the include files
#include   "c:\\design\\scma\\z80\\set_scma.pre"
#include   "c:\\design\\scma\\z80\\rambo.h"
#include   "c:\\design\\scma\\z80\\char.h"
#include   "c:\\design\\scma\\z80\\map_io.h"
#include   "c:\\design\\scma\\z80\\memory.h"
#include   "c:\\design\\scma\\z80\\macro_io.def"
;*****
start     .org  0000h
          di                      ; Processor startup code
          im  2                    ; interrupts from table
          ld  hl,stack              ; load stack from memory.h
          ld  sp,hl                 ; and load
          ld  a,(int_table&0ff00h)>>8; load I reg with table addr.
          ld  i,a                   ; load
          call io_setup             ; setup on chip peripherals
          jp  start_main           ; and go
;*****
          .org  038h
rx_test:
          push af
          push bc
          push de
          push hl
          push ix
          push iy
rx_test1:
          sio_in(0)                 ; wait for character to be received
          bit  0,a                   ; mask off
          jp  z,rx_test1           ; and loop if not yet available
rx_test6:
          ld  ix,rx_buffer          ; get start of buffer
          ld  iy,rx_cursor         ; get cursor pointer
          ld  a,(rx_cursor)        ; get cursor
```

```

        ld    b,a
rx_test4:
        inc  ix          ; move ix to rx cursor
        djnz rx_test4   ; and repeat
        dec  ix          ; correct for initial loop
rx_test3:
        sio_in(0)       ; is there a character to be received??
        bit  0,a         ; mask off
        jp   z,rx_test2 ; no, so exit
        in   a,(sio_data_a) ; yes, so load data
        ld   (ix+0),a    ; and store in rx_buffer
        cp   'Z'        ; end of message (chr_cr)
        jp   nz,rx_test5 ; carry on if not end of message
        ld   a,1         ; reset cursor
        ld   (iy+0),a    ; and load iy into rx_cursor
        ld   a,(rx_buffer) ; is it for the host??
        cp   host       ; use host
        jp   nz,rx_test7 ; no, so just delete
        ld   a,(rx_buffer+1) ; is the remote host the IBM??
        cp   'I'
        jp   nz,rx_test7 ; no,so delete
        ld   de,rx_holding ; load holding reg with rx buffer
        ld   hl,rx_buffer ; (de) <= (hl) for bc times
        ld   bc,30       ; repeat 30 times
        ldir
        ld   a,'I'      ; signal new command
        ld   (holding_flag),a
rx_test7:
        ld   ix,rx_buffer ; clear up
        call clear_buffer ; clear rx_buffer
        jp   rx_test2     ; and exit
rx_test5:
        inc  (iy+0)      ; next location
        jp   rx_test6     ; and loop
rx_test2:
        sio_ctrl(0,038h) ; send a return from interrupt
        sio_ctrl(0,020h) ; and enable on next rx character
        pop  iy
        pop  ix
        pop  hl
        pop  de
        pop  bc
        pop  af
        ei
        reti
;*****
ctc_int:

```

```

push af          ; save registers used
push bc          ; push all registers on stack
push de
push hl
push ix
push iy
ld ix,intcount  ; point to scaler dividers in memory
dec (ix+2)      ; decrement scaler 1
jp nz,ctcint1  ; not ready for processing,so exit
ld a,(ix+0)     ; reload divider
ld (ix+2),a    ; and store back in register
; time to update present encoder reading (approx 10 updates/sec)
ld a,(0FFF2h)  ; get low byte of fine encoder from input
ld l,a         ; and store (as part of twin reg. hl)
ld a,(0FFF3h)  ; now get high byte of fine encoder
and 007h      ; mask out all but low 3 bits
ld h,a        ; and store
ld (newencoder),hl ; new fine encoder now stored in ram
; now, has it cycled itself?? i.e. is
; update < 0 (CCW) or update > 2^10 (CW)
ld de,(oldencoder) ; get old encoder value
ld a,0         ; clear direction flag
and a         ; clear carry flag
sbc hl,de     ; get update value
jp nc,ctcint2 ; carry not set, so direction +ve
ld a,1        ; set direction flag
ld hl,(oldencoder) ; swap registers
ld de,(newencoder) ; and here to to clear carry on subtraction
and a        ; clear carry
sbc hl,de    ; and subtract
ctcint2:
ld de,0100h  ; if delta |encoder| > 2^9 sync passed
sbc hl,de    ; compare, and if no sync carry set
jp c,ctcint3 ; no sync found, so compile encoder
cp 1        ; sync +ve or -ve, so interrogate sync flag
jp z,ctcint4 ; direction +ve, so load coarse from I/O
ld a,(coarse) ; else load coarse from memory
dec a       ; and coarse -1
ld (coarse),a ; and store back again
jp ctcint3  ; and compile encoder
ctcint4:
ld a,(0FFF0h) ; load coarse encoder
and 01Fh      ; mask out all unused bits
ld (coarse),a ; and store
; Yep, This encoder is 6bit wide, and I'm only saving 5 bits!! Why?? simple
; fine + coarse encoders would give 17 bits, and 17 bit addition is bloody
; difficult! That with 30deg. dead space from the scope pier means that the

```

; top bit will never be set. Thus bodge it.

```
    ld    a,1                ; and show that position os OK
    ld    (okflag),a        ; store
ctcint3:
    ld    hl,(newencoder)   ; get new position
    ld    (oldencoder),hl   ; store into old position
    ld    a,(coarse)        ; get coarse position
    sla  a                  ; shift encoder 3 times to top location
    sla  a
    sla  a                  ; OK
    or   h                  ; overlay fine and coarse encoder
    ld    h,a               ; and store back again
    ld    (present),hl      ; and store in present
; get offset (works - so don't touch it)
    ld    hl,(present)      ; compare present to future
    ld    de,(future)       ; to get direction of offset
    and  a                  ; clear carry flag
    sbc  hl,de              ;
    jp   c,calcoff1         ; jump if future < present
    ld    (offset),hl       ; save offset in memory
    ld    a,0               ; and set direction to ??
    ld    (offset_dir),a
    jp   ctcint5
calcoff1:
    ld    hl,(future)       ; here swap future and present registers
    ld    de,(present)     ; and here too
    and  a                  ; clear carry flag
    sbc  hl,de              ;
    ld    (offset),hl       ; save offset in memory
    ld    a,1               ; and invert direction
    ld    (offset_dir),a   ; calcoff ends here
ctcint5:
    dec  (ix+3)             ; decrement scaler 2
    jp   nz,ctcint1        ; not ready for processing, so exit
    ld   a,(ix+1)          ; reload divider chain again
    ld   (ix+3),a          ; and store
ctcint1:
; exit routine here
    pop  iy
    pop  ix
    pop  hl                ; clear up and exit
    pop  de
    pop  bc
    pop  af
    ei                      ; and enable interrupts
    reti                   ; and exit
```

\*\*\*\*\*

unused:

```

    ei
    reti
;*****
    .org (0ff00h & ($) + 0ffh)
io_setup:
#include "c:\\design\\scma\\z80\\setup_io.rel"
    ret
;*****
    .org (0ff00h & ($) + 0ffh)
; Interrupt table in here
int_table:
int_table_ctc:
    .byte unused & 0ffh
    .byte (unused & 0ff00h) >> 8
    .byte unused & 0ffh
    .byte (unused & 0ff00h) >> 8
    .byte ctc_int & 0ffh
    .byte (ctc_int & 0ff00h) >> 8
    .byte unused & 0ffh
    .byte (unused & 0ff00h) >> 8
    .byte unused & 0ffh
    .byte (unused & 0ff00h) >> 8
int_table_sio:
    .byte rx_test & 0ffh
    .byte (rx_test & 0ff00h) >> 8
;*****
    .org (0ff00h&($+00ffh))
start_main:
    call setup ; setup memory, ports default values
    ei ; enable interrupts
    sio_ctrl(5,01101000b) ; testd
    sio_ctrl(0,020h) ; enable rx on next character
    ld ix,text_on ; and load on message
    call tx_request ; and transmit
main_loop:
    call movemon
    call rx_mon
    call print_text
    jp main_loop
;*****
starttest:
    call ret_clear ; clear buffer
    ld ix,diag1 ; all setup, so transmit Diag 1 messages
    call tx_request ; tx
    call print_text ; tx message to network
    ld hl,14904d ; Delay startup 5 sec
    call delay

```

```

ld    a,1                ; set direction CW
ld    (slew_dir),a      ; and store
ld    a,0
ld    (okflag),a        ; invalidate ok flag
call  slewstart
starttest1:
ld    a,(okflag)        ; wait for flag to be updated
cp    1
jp    nz,starttest1
call  slewstop
ld    hl,2981d           ; delay for 1 sec
call  delay
ld    ix,diag2           ; finished, so send next messages
call  tx_request        ; tx
ld    hl,(present)      ; lock present & future together
ld    (future),hl
call  centre            ; centre tangent arm
jp    print_text        ; tx message to network
,*****
setup:
ld    a,0FFh            ; shut off all motors, whatever state
ld    (base_io+motor),a ; motors off
ld    (base_io+direction),a ; and others
ld    ix,present
call  clear_reg
ld    ix,offset
call  clear_reg
ld    ix,future
call  clear_reg
ld    hl,0FA14h         ; load in int counters
ld    (intcount),hl    ; and store
ld    hl,00533h         ; Load default slew hysteresis
ld    (h1),hl
ld    hl,40d            ; Load default set hysteresis
ld    (h2),hl
ld    hl,5d             ; Load default guide hysteresis
ld    (h3),hl
ld    hl,299            ; 100mS delay
ld    (delay_mech),hl  ; for mechanical delay
ld    hl,1491           ; 500mS delay
ld    (delay_over),hl  ; for telescope inertia
ld    hl,1491           ; 500mS delay
ld    (delay_inert),hl ; for overload inertia
ld    hl,1491           ; 500mS delay
ld    (delay_turn),hl  ; for rewinding clamp 1/4 turn
ld    hl,000FFh        ; 85 mS arbitration delay
ld    (delay_arb),hl   ; for arbitration delay

```

```

ld    hl,000FFh      ; 85mS (seems like a good figure)
ld    (delay_out),hl ; for tx inactive settling time
ld    hl,01          ; 338uS
ld    (delay_cap),hl ; for capacitive decay in comms line
ld    a,0            ; Set slew flag to 0
ld    (slewflag),a
ld    (moveflag),a  ; and switch move to manual
ld    (okflag),a    ; invalidiadte encoder ok flag
ld    (limitflag),a ; clear limits
ld    a,1           ; Set cursor to start of rx buffer
ld    (rx_cursor),a
ld    ix,rx_buffer
call  clear_buffer
ld    ix,rx_holding
call  clear_buffer
ld    a,'O'
ld    (holding_flag),a
ret

;*****
; sort out the commands here -scrap below
rx_mon:
ld    a,(rx_holding+2)
cp    'Q'
jp    z,testcode
cp    'M'
jp    z,moveint
cp    'D'
jp    z,dispres
cp    'P'
jp    z,inppres
cp    'R'          ; soft reset, reset encoders
jp    z,starttest
cp    'H'          ; Hard reset, start again
jp    z,start
cp    'C'          ; centre dec tangent arm
jp    z,centrego
ld    a,(holding_flag) ; anything there??
cp    'I'          ; (indicated with 'I')
ret    nz          ; exit if nothing
ld    a,'O'        ; or load O to invalidate it
ld    (holding_flag),a
ld    ix,text_rx
jp    tx_request

;*****
clear_reg:
ld    a,0
ld    (ix+0),a

```

```

    ld    (ix+1),a
    ret
;*****
vectorin:
    ld    a,(rx_holding+3)    ; vectorin routine load iy with the rx
    select('P',present)      ; address of the 16 bit (2 byte)
    select('F',future)        ; variable to be acted on or read out
    select('O',offset)        ; this holds all the data to be changed
    select('S',h1)            ; save for the move register
    select('T',h2)            ; is acted on manually
    select('G',h3)
    select('N',0FFF0h)
    select('W',0FFF2h)
    select('H',0FFF4h)
    select('M',delay_mech)
    select('I',delay_inert)
    select('A',delay_arb)
    select('U',delay_out)
    select('K',okflag)
    select('D',intcount)
    ld    iy,0FFFEh          ; sent to dud space
    ret                      ; and return. this MAY NOT be a good idea..
;*****
testcode:
    ld    a,'P'
    ld    (rx_holding+3),a
    call  dispres
    call  print_text
    ld    ix,text_rtn
    call  tx_request
    call  print_text
    jp    testcode
;*****
movemon:
    ld    a,(moveflag)        ; is automove flag set??
    cp    0                   ; automatic if!=0
    ret   z                   ; it isn't, so exit (slew off in moveint)
    ld    a,(okflag)          ; is unit OK??
    cp    0                   ; buggered if 0
    jp    z,movemon1         ; yes, so stop all motors
    ld    a,(offset_dir)      ; else load offset direction to slew
    ld    (slew_dir),a        ; and store again
    ld    hl,(offset)         ; is slew hysteresis > offset
    ld    de,(h1)             ; h1 holds the slew hysteresis
    and   a                   ; clear carry flag
    sbc   hl,de               ; subtract it and look for carry
    jp    nc,slewon           ; OK, so slew

```

```

call    slewoff          ; no,so check slewing logic off
ld      hl,(offset)      ; is fine set hysteresis > offset
ld      de,(h2)         ; h1 holds the fine set hysteresis
and     a
sbc     hl,de
jp      nc,seton        ; Ok, so set on
ld      a,0             ; reset automove flag
ld      (moveflag),a    ; save in register
ld      ix,text_ok      ; send message saying drive stopped
call    tx_request      ; and transmit

movemon1:
call    slewoff          ; switch slew off
jp      fineoff         ; else switch off everything
;*****
moveint:
ld      a,01h           ; preload automove as active
ld      (moveflag),a    ; and store
ld      a,(rx_holding+3) ; read in control byte
cp      'G'             ; automove - let computer go
jp      z,moveint1     ; yep, so return
ld      a,00h           ; automove has been canceled, so stop
ld      (moveflag),a    ; and store
call    slewoff         ; check that the slewing is off
ld      a,(rx_holding+3) ; read in control byte
ld      b,0FDh          ; preload with set -
cp      '1'             ; is it set- ('1')
jp      z,manmove      ; yes, so exit
ld      b,0FDh          ; preload with guide -
cp      '2'             ; is it guide- ('2')
jp      z,manmove      ; yes, so exit
ld      b,0FCh          ; preload with guide +
cp      '3'             ; is it guide + ('3')
jp      z,manmove      ; yes, so exit
ld      b,0FCh          ; preload with set +
cp      '4'             ; is it set + ('4')
jp      z,manmove      ; yes, so exit
ld      b,0             ; preload b with slew direction CW
cp      '9'             ; is it CW slew??
jp      z,moveint2     ; yes, so start
ld      b,1             ; preload b with slew direction CCW
cp      '8'             ; is it CCW slew??
jp      z,moveint2     ; yes, so start
ld      b,0FFh         ; stop if not

manmove:
ld      a,b             ; swap for load
ld      (base_io+direction),a; and load
jp      ret_com         ; and exit

```

```

moveint2:
    ld    a,b                ; swap b => a for load
    ld    (slew_dir),a      ; store in slew direction
    call  slewstart         ; start slewing
moveint1:
    ; exit routine.
    jp    ret_com           ; and exit
;*****
slewon:
; check that the slewing is on (by checking flag)
    ld    a,(slewflag)      ; get slew flag
    cp    1
    ret   z                 ; everything's going ok - so exit
    jp    slewstart        ;slew needs to be switched on
;*****
slewoff:
; check that the slewing is off (as before)
    ld    a,(slewflag)      ; get slew flag
    cp    0
    ret   z                 ; everything's stopped - so exit
    jp    slewstop
;*****
guideon:
seton:
    ld    b,0FCh            ; preload b with fine motor guide+
    ld    a,(offset_dir)    ; get direction
    cp    0                 ; which direction??
    jp    nz,seton1        ; cw, so start cw motor
    ld    b,0FDh            ; ccw, so load b with guide-
seton1:
    ld    a,b                ; swap for load
    ld    (base_io+direction),a
    ret
;*****
fineoff:
    ld    a,0FFH            ; switch all fine inputs off
    ld    (base_io+direction),a
    ret
;*****
centregoto:
    call  centre            ; centre tangent arm
    jp    ret_com           ; and clear rx buffer
;*****
centre:
    ld    a,(0FFF1h)        ; get centre optoswitch
    ld    b,a                ; store in b ( save for later)
    ld    c,0FCh            ; preload c with direction of fine motor
    bit   7,a                ; which side of centre is tangent arm on

```

```

        jp    nz,centre1      ; this way, so run motor
        ld    c,0FDh         ; no, other way, so reload c with
                                ; correct dir
centre1:
        ld    a,c            ; swap a,c for load
        ld    (base_io+direction),a; start tangent arm motor up
        ld    a,(0FFF1h)     ; get centre optoswitch (again)
        xor   b              ; has it changed???
        jp    z,centre1      ; no, so continue
        ld    a,0FFh         ; yes, centre has been reached
        ld    (base_io+direction),a; so stop motor and exit
        ret
;*****
inppres:
        call  vectorin       ; set vector to destination
        call  clear_reg      ; clear register
        ld    ix,rx_holding+4 ; set pointer to start of 1's &0's
        ld    b,010h        ; set repeat loop for 16 (2 byte)
inppres1:
        ld    a,(ix+0)       ; load in bit to be tested
        cp    '1'           ; is it a '1'??
        scf                    ; preload with a carry set
        jp    z,inppres2     ; and if it is, start rolling
        and   a              ; else clear flag
inppres2:
        rl    (iy+0)         ; rotate carry into memory location
        rl    (iy+1)         ; and through into high byte
        inc   ix             ; next rx holding buffer
        djnz inppres1
        jp    ret_com
;*****
dispres:
        ld    ix,text_16     ; create 16 binary mask
        call  tx_request     ; and validate it for tx'ing
        call  vectorin       ; get input vector to load data from
dispres5:
        ld    ix,tx_buffer   ; set pointer to start of tx buffer
        ld    a,(iy+1)       ; load high byte of display buffer => a
        ld    b,8            ; set loop to shift 8 bits
dispres2:
        sla   a              ; shift top bit into carry
        jp    nc,dispres1    ; if not '1' next shift
        inc   (ix+3)         ; it's a '1', so inc '0' => '1'
dispres1:
        inc   ix             ; next bit in tx buffer
        djnz dispres2        ; repeat 8 times
        ld    a,(iy+0)       ; load low byte of display buffer => a

```

```

    ld    b,8                ; set loop to shift 8 bits
dispres4:
    sla   a                  ; shift top bit into carry
    jp    nc,dispres3       ; if not '1' next shift
    inc   (ix+3)            ; it's a '1', so inc '0' => '1'
dispres3:
    inc   ix                ; next bit in tx buffer
    djnz  dispres4         ; repeat 8 times
    jp    ret_clear
;*****
slewstart:
    ld    a,0ffh            ; cancel fine drive
    ld    (base_io+direction),a
    call  clampoff         ; unlock clamp
    ld    a,(slew_dir)      ; get offset direction
    srl   a                 ; move into carry
    ld    a,0ffh            ; and set a to directiontemp
    rr    a                 ; and rotate it back in
    ld    (base_io+direction),a
    ld    hl,(delay_mech)   ; wait for relays
    call  delay
    ld    a,07fh            ; switch on motor
    ld    (base_io+motor),a
    ld    a,001h            ; flag to indicate slewing is on
    ld    (slewflag),a     ; and store
    ret
;*****
clampoff:
    ld    a,0FFh            ; load clamp direction OUT
    ld    (base_io+direction),a; and store
    ld    hl,(delay_mech)   ; wait for contacts
    call  delay             ; cold start,
                                ; the best for your intimate contacts
clampoff1:
    ld    a,0B0h            ; now start up motor and buzzer
    ld    (base_io+motor),a ; and store
    ld    a,(0FFF4h)        ; get clamp out microswitch
    bit   4,a               ; is it set??
    jp    z,clampoff1       ; no, so repeat
    ld    hl,(delay_over)   ; wait for a msec
    call  delay
    ld    a,(0FFF4h)        ; get clamp out microswitch
    bit   4,a               ; is it set??
    jp    z,clampoff1       ; no, so repeat
    ld    a,0FFh            ; now stop motor
                                ; (will automatically stop anyway)
    ld    (base_io+motor),a ; and store

```

```

    ld    hl,(delay_mech)    ; wait for contacts
    jp    delay              ; and return
;*****
slewstop:
    ld    a,0ffh            ; stop slew motor
    ld    (base_io+ motor),a
    ld    hl,(delay_mech)   ; wait for relays
    call  delay
    ld    a,0ffh            ; reset contacts
    ld    (base_io+ direction),a
    call  clampon           ; lock clamp on
    ld    a,00h             ; indicate that slewing has finished
    ld    (slewflag),a
    ret                      ; and exit
;*****
clampon:
    ld    a,0BFh            ; load clamp direction IN
    ld    (base_io+ direction),a ; and store
    ld    hl,(delay_mech)   ; wait for contacts
    call  delay             ; wait..
clampon1:
    ld    a,0BFh            ; now start up motor
    ld    (base_io+ motor),a ; and store
    ld    a,(0FFF4h)        ; get clamp in microswitch
    bit   5,a               ; is it set??
    jp    z,clampon1        ; no, so repeat
    ld    a,0FFh            ; now stop motor
                                ; (will automatically stop anyway)
    ld    (base_io+ motor),a ; and store
    ld    hl,(delay_mech)   ; wait for contacts
    call  delay
    ret                      ; and return
;*****
limitmon:
    ld    a,(0FFF4h)        ; get limits in
    and   00Ch              ; mask off all but fine limits
    call  nz,centre         ; if hit, recentre 'scope
    ld    a,(0FFF4h)        ; get limits in again
    and   0C0h              ; mask off all but axis limits
    ret   z                 ; none hit, so exit
    ld    a,0               ; clear moveflag
    ld    (moveflag),a      ; and store
    call  slewstop          ; stop slewing
    call  fineoff           ; stop fine motors
    ld    a,(0FFF4h)        ; get limits in
    and   0CCh              ; mask off all other limits
    ld    b,a               ; swap for xor

```

```

    ld    a,(limitflag)      ; load in old limits
    xor   b                  ; have they changed??
    ret   z                  ; no, so exit
    ld   a,b                 ; yes, so save new limits
    ld   (limitflag),a      ; and save
    ld   ix,text_hit        ; send hit limit
    jp   tx_request         ; and send
;*****
; This routine does all the delays for the program, and all the delays can be
; loaded by the user at runtime. All default delays are loaded in setup at the
; start and then modified by user. NOTE: changing delay times can serious
; affect the performance of the system!!!. Think about it. delay time in uS
; delay time = 2.5+335.5*HL uS
delay:
    ld   de,01
delay1:
    ld   b,0ffh
delay2:
    djnz delay2
    and  a
    sbc  hl,de
    jp   nz,delay1
    ret
;*****
ret_com:
    ld   ix,text_ack
    call tx_request
ret_clear:
    ld   a,'O'
    ld   (holding_flag),a
    ld   ix,rx_holding
    jp   clear_buffer
;*****
clear_buffer:
    ld   b,250
    ld   a,0
clear1:
    ld   (ix+0),a
    inc  ix
    djnz clear1
    ret
;*****
tx_request:
    ld   iy,tx_buffer
tx_req2:
    ld   a,(ix+0)
    cp   chr_null

```

```

    jp    nz,tx_req1
    ld    a,chr_null
    ld    (iy+0),a
    ret
tx_req1:
    ld    (iy+0),a
    inc   ix
    inc   iy
    jp    tx_req2
;*****
; this routine holds all the protocol for transmitting a data
; packet, including txrx arbitration, etc
print_text:
    ld    ix,tx_buffer           ; get start of tx buffer
    ld    a,(ix+0)              ; get valid text line character
    cp    'S'                   ; is this character = valid??
    ret   nz                    ; no, so exit
    sio_in(0)                   ; get cts bit
    bit   5,a                   ; is cts set??
    ret   z                     ; txrx line active - so quit
    sio_ctrl(5,11101000b)       ; set txrx control active
    ld    hl,(delay_arb)
    call  delay                 ; delay for a preset time
    sio_ctrl(5,01101000b)       ; set txrx control inactive
    ld    hl,(delay_cap)
    call  delay                 ; delay for capacitor
    sio_in(0)                   ; get cts bit
    bit   5,a                   ; is cts set??
    ret   z                     ; yes, so get off line
    sio_ctrl(5,11101000b)       ; switch on txrx line for good
    sio_ctrl(3,11000000b)       ; shut off receiver
    sio_ctrl(1,00000000b)       ; switch off interrupts
print_t2:
    ld    a,(ix+0)              ; a < first character of line
    cp    chr_null              ; is it the end of the message
    jp    nz,print_t1          ; no, so print message
    ld    a,chr_null            ; end of message, so send null
    call  print_a              ; and print it.
print_t3:
    sio_in(0)                   ; get tx buffer empty
    bit   2,a                   ; is tx buffer empty?
    jp    nz,print_t3          ; no, so wait
print_t4:
    sio_in(1)                   ; get all_sent bit
    bit   0,a                   ; is it all sent
    jp    nz,print_t4          ; no, so wait
    ld    a,'*'                ; signal with any char

```

```

ld    ix,tx_buffer      ; like *
ld    (ix+0),a          ; that message is sent
ld    hl,(delay_out)
call  delay
sio_ctrl(5,01101010b)  ; set txrx control inactive
in    a,(sio_data_a)   ; clear any characters held
in    a,(sio_data_a)   ; in holding buffer
in    a,(sio_data_a)   ;
in    a,(sio_data_a)   ;
sio_ctrl(3,11000001b)  ; and switch on reciever
sio_ctrl(1,00011000b)  ; and enable interrupts
ret                                ; return

print_t1:
call  print_a           ; print character
inc   ix                ; increment pointer
jp    print_t2          ; and repeat
;*****
; This routine must not ba called to send a character only.
; it will violate the network - no TXRX control
print_a:
ld    b,a               ; b <= a (store it temp)
print_1:
sio_in(0)               ; get tx empty (bit status)
bit   2,a                ; empty ??
jp    z,print_1         ; no, so loop untill it is
ld    a,b                ; transfer character back to a
out   (sio_data_a),a    ; and send it to the comms
ret                       ; and return
;*****
text_on    .text  "SI"
           .byte  host
           .text  "NODE_RESET_OK"
           txt_end

diag1     .text  "SI"
           .byte  host
           .text  "SYSTEM_NODE_ACK"
           txt_end

diag2     .text  "SI"
           .byte  host
           .text  "SYSTEM_TEST_OK"
           txt_end

text_ack  .text  "SI"
           .byte  host
           .text  ACK"

```

```

txt_end

text_hit .text "SI"
        .byte host
        .text LIMIT_HIT_ERR"
txt_end

text_rtn .text "SI"
        .byte host
        .byte 10
        .byte 13
txt_end

text_ok .text "SI"
        .byte host
        .text "OK"
txt_end

text_rx .text "SI"
        .byte host
        .text "RX"
txt_end

text_d .text "SI"
        .byte host
        .text "D000ACK"
txt_end

text_input: .text "SI"
           .byte host
           .text "00000000ACK"
txt_end

text_16: .text "SI"
        .byte host
        .text "0000000000000000ACK"
txt_end

text_err .text "SI"
        .byte host
        .text "ERR"
txt_end

```

### C.3 : Source code for OFFSET axis microcontrollers

Filename:	OFFSET.Z80
Application:	Source code for offset axis Z80 microcontrollers

```
; ST. ANDREWS TWIN PHOTOMETRIC TELESCOPE
; OFFSET AXIS DRIVE PROGRAM
; 150993V5.0/R.T.GEARS/TPT-TECH.DOC
;
; Set host to axis
host      .equ   ra_offset      ; define Offset Ra/Dec id
host      .equ   dec_offset     ; delete / comment as needed
; Load the include files
#include   "c:\\design\\scma\\z80\\set_scma.pre"
#include   "c:\\design\\scma\\z80\\rambo.h"
#include   "c:\\design\\scma\\z80\\char.h"
#include   "c:\\design\\scma\\z80\\map_io.h"
#include   "c:\\design\\scma\\z80\\memory.h"
#include   "c:\\design\\scma\\z80\\macro_io.def"
;*****
start     .org   0000h
          di                      ; Processor startup code
          im    2                  ; interrupts from table
          ld    hl,stack           ; load stack from memory.h
          ld    sp,hl              ; and load
          ld    a,(int_table&0ff00h) > > 8 ; load I reg with table addr.
          ld    i,a                ; load
          call  io_setup           ; setup on chip peripherals
          jp    start_main        ; and go
;*****
          .org   038h
rx_test:
          push  af
          push  bc
          push  de
          push  hl
          push  ix
          push  iy
rx_test1:
          sio_in(0)                ; wait for character to be received
          bit   0,a                 ; mask off
          jp    z,rx_test1         ; and loop if not yet available
rx_test6:
          ld    ix,rx_buffer       ; get start of buffer
          ld    iy,rx_cursor      ; get cursor pointer
          ld    a,(rx_cursor)     ; get cursor
```

```

        ld    b,a
rx_test4:
        inc   ix                ; move ix to rx cursor
        djnz  rx_test4         ; and repeat
        dec   ix                ; correct for initial loop
rx_test3:
        sio_in(0)              ; is there a character to be received??
        bit   0,a               ; mask off
        jp    z,rx_test2       ; no, so exit
        in    a,(sio_data_a)    ; yes, so load data
        ld    (ix+0),a          ; and store in rx_buffer
        cp    'Z'               ; end of message (chr_cr)
        jp    nz,rx_test5       ; carry on if not end of message
        ld    a,1               ; reset cursor
        ld    (iy+0),a          ; and load iy into rx_cursor
        ld    a,(rx_buffer)     ; is it for the host??
        cp    host              ; use host
        jp    nz,rx_test7       ; no, so just delete
        ld    a,(rx_buffer+1)   ; is the remote host the IBM??
        cp    'I'
        jp    nz,rx_test7       ; no,so delete
        ld    de,rx_holding     ; load holding reg with rx buffer
        ld    hl,rx_buffer      ; (de) <= (hl) for bc times
        ld    bc,30             ; repeat 30 times
        ldir
        ld    a,'I'             ; signal new command
        ld    (holding_flag),a
rx_test7:
        ld    ix,rx_buffer      ; clear up
        call  clear_buffer      ; clear rx_buffer
        jp    rx_test2          ; and exit
rx_test5:
        inc   (iy+0)            ; next location
        jp    rx_test6          ; and loop
rx_test2:
        sio_ctrl(0,038h)        ; send a return from interrupt
        sio_ctrl(0,020h)        ; and enable on next rx character
        pop   iy
        pop   ix
        pop   hl
        pop   de
        pop   bc
        pop   af
        ei
        reti
;*****
ctc_int:

```

```

push af          ; save registers used
push bc          ; push all registers on stack
push de
push hl
push ix
push iy
ld a,(oldencoder) ; get previous encoder reading
ld c,a          ; c = old encoder reading
ld a,(0FFF2h)   ; get new encoder reading from I/O port
srl a
and 07h         ; mask out unused high bits
ld (newencoder),a ; and store in new_encoder memory location
cp c           ; is old_encoder = new_encoder (no move)
jp z,ctcint1   ; dome 'stationary' during update, so exit.
bit 2,a        ; is sync bit set (dome through 0/360 Deg)
jp z,ctcint2   ; no, so get inc/dec status
ld a,(dome_direction) ; sync pulse,
                ; so determine whether to set or clear
ld hl,(present) ; subtract 360*4 from encoder data
ld de,1440      ; de holds the encoder max res. data
cp 01h         ; if dome_direction=1 (+ve) clear with hl
jp nz,ctcint3a ; clear, using preloaded hl
and a          ; clear carry flag and ..
sbc hl,de      ; subtract 360Deg. from Present value
jp ctcint3     ;and store

ctcint3a:
and a          ; clear carry flag and ...
adc hl,de      ; add 360 Deg. to present value

ctcint3:
ld (present),hl ; load present location with set/clear hl reading
ld a,1          ; and Indicate that dome is OK (valid)
ld (okflag),a  ; load valid flag
jp ctcint7     ; and jump to calculate offset routine

ctcint2:
; Encoder has incremented/decremented, so look up old encoder
; in data table, and see which side of it is the new reading
ld ix,data_table-1 ; ix points to start of incremental cycle table
ld a,(oldencoder) ; get old encoder reading
and 03h         ; mask out all but phase quad information
                ; from the encoder

ctcint4:
inc ix         ; next phase quad data table location
cp (ix+1)     ; Data entry == old_encoder ??
jp nz,ctcint4 ; no so repeat (max 4 times) :
                ; can only be 00 01 11 10
                ; data table pointer (ix) now pointing
                ; to old Phase

```

```

; now compare either side to the new Phase
ld    a,(newencoder)    ; load a with new encoder, to enable
; compare with ix+?
and   03h                ; mask out all but phase quad information
; in memory
cp    (ix+2)             ; is the dome going cw (+ve)
jp    z,ctcint5          ; yes, so jump to cw routine
cp    (ix+0)             ; is dome going ccw (-ve)
jp    z,ctcint6          ; yes, so jump to ccw routine.
ld    a,0                ; if it's not going cw/ccw (as above) -
; it's buggered.
ld    (okflag),a        ; so clear ok flag -
; show that encoder is knackered
jp    ctcint1           ; and exit (will calculate offset,
; but ok flag overrides)

ctcint5:
ld    a,01h              ; set the direction = cw
ld    (dome_direction),a ; and save in encoder direction,
; for use by interrupt routine
ld    hl,(present)      ; get current position
inc   hl                 ; and increment
ld    (present),hl      ; and store back again
jp    ctcint7           ; jump to exit saving old encoder

ctcint6:
ld    a,0                ; set the direction = ccw
ld    (dome_direction),a ; and save in encoder direction,
; for use by interrupt routine
ld    hl,(present)      ; get current position
dec   hl                 ; and decrement
ld    (present),hl      ; and store back again

ctcint7:
ld    hl,(present)      ; compare present to future
ld    de,(future)       ; to get direction of offset
and   a                  ; clear carry flag
sbc   hl,de              ;
jp    c,ctcint9          ; jump if future < present
ld    (offset),hl        ; save offset in memory
ld    a,0                ; and set direction to ??
ld    (offset_dir),a
jp    ctcint10

ctcint9:
ld    hl,(future)        ; here swap future and present registers
ld    de,(present)      ; and here too
and   a                  ; clear carry flag
sbc   hl,de
ld    (offset),hl        ; save offset in memory
ld    a,1                ; and invert direction

```

```

        ld      (offset_dir),a
ctcint10:
ctcint1:
        ld      a,(newencoder)      ; shift new encoder => old encoder
        ld      (oldencoder),a      ; and move new encoder -> old
        pop     iy
        pop     ix
        pop     hl                    ; clear up and exit
        pop     de
        pop     bc
        pop     af
        ei                          ; and enable interrupts
        reti                          ; and exit
;*****
unused:
        ei
        reti
;*****
        org    (0ff00h & ($ + 0ffh))
io_setup:
#include    "c:\\design\\scma\\z80\\setdo_io.rel"
        ret
;*****
        .org    (0ff00h & ($ + 0ffh))
; Interrupt table in here
int_table:
int_table_ctc:
        .byte   unused & 0ffh
        .byte   (unused & 0ff00h) >> 8
        .byte   unused & 0ffh
        .byte   (unused & 0ff00h) >> 8
        .byte   ctc_int & 0ffh
        .byte   (ctc_int & 0ff00h) >> 8
        .byte   unused & 0ffh
        .byte   (unused & 0ff00h) >> 8
        .byte   unused & 0ffh
        .byte   (unused & 0ff00h) >> 8
int_table_sio:
        .byte   rx_test & 0ffh
        .byte   (rx_test & 0ff00h) >> 8
;*****
        .org    (0ff00h & ($ + 00ffh))
start_main:
        call    setup                ; setup memory, ports default values
        ei                          ; enable interrupts
        sio_ctrl(5,01101000b)        ; testd
        sio_ctrl(0,020h)             ; enable rx on next character

```

```

        ld    ix,text_on
        call  tx_request
main_loop:
        call  movemon
        call  limitmon
        call  rx_mon
        call  print_text
        jp    main_loop
;*****
starttest:
        ld    hl,14904d          ; Delay startup 5 sec
        call  delay
        call  centre            ; centre unit
        ld    hl,2981d          ; Delay centre 1 sec
        call  delay
        ld    a,0
        ld    (okflag),a        ; invalidate ok flag
        ld    a,0FCh            ; switch on motor, +ve direction.
        ld    (base_io+direction),a
starttest1:
        ld    a,(okflag)        ; wait for flag to be updated
        cp    1
        jp    nz,starttest1
        ld    a,0FFh            ; switch off all valves
        ld    (base_io+direction),a
        ld    hl,2981d          ; delay for 1 sec
        call  delay
        di                        ; disable interrupts for a sec. [metaphorically]
        ld    hl,08000h         ; store present,future as mid point ..
        ld    (present),hl      ; lock present & future together
        ld    (future),hl
        ei
        call  ret_clear
        ld    ix,diag2          ; finished, so send next messages
        call  tx_request        ; tx
        ret
;*****
setup:
        ld    a,0FFh            ; shut off all motors, whatever state
        ld    (base_io+motor),a ; motors off
        ld    (base_io+direction),a; direction relays off
        ld    ix,present
        call  clear_reg
        ld    ix,offset
        call  clear_reg
        ld    ix,future
        call  clear_reg

```

```

ld    hl,0FA14h        ; load in int counters
ld    (intcount),hl   ; and store
ld    hl,4             ; Load default slew hysteresis (5 deg)
ld    (h1),hl
ld    hl,4             ; Load default set hysteresis (2.5 deg)
ld    (h2),hl
ld    hl,4             ; Load default guide hysteresis (1deg)
ld    (h3),hl
ld    hl,299          ; 100mS delay
ld    (delay_mech),hl ; for mechanical delay
ld    hl,1491         ; 500mS delay
ld    (delay_over),hl ; for telescope inertia
ld    hl,1491         ; 500mS delay
ld    (delay_inert),hl ; for overload inertia
ld    hl,1491         ; 500mS delay
ld    (delay_turn),hl ; for rewinding clamp 1/4 turn
ld    hl,000FFh       ; 85 mS arbitration delay
ld    (delay_arb),hl  ; for arbitration delay
ld    hl,000FFh       ; 85mS (seems like a good figure)
ld    (delay_out),hl  ; for tx inactive settling time
ld    hl,01           ; 338uS
ld    (delay_cap),hl  ; for capacitive decay in comms line
ld    a,0             ; Set slew flag to 0
ld    (slewflag),a
ld    (moveflag),a    ; and switch move to manual
ld    (okflag),a     ; invalidate encoder ok flag
ld    (limitflag),a  ; clear limits
ld    a,1             ; Set cursor to start of rx buffer
ld    (rx_cursor),a
ld    ix,rx_buffer
call  clear_buffer
ld    ix,rx_holding
call  clear_buffer
ld    a,'O'
ld    (holding_flag),a
ret

```

\*\*\*\*\*

; sort out the commands here -scrap below

rx\_mon:

```

ld    a,(rx_holding+2)
cp    'Q'
jp    z,testcode
cp    'M'
jp    z,moveint
cp    'D'
jp    z,dispres
cp    'P'

```

```

jp      z,inppres
cp      'R'                ; soft reset, start again
jp      z,starttest
cp      'C'
jp      z,centrego
cp      'H'
jp      z,start           ; hard reset
ld      a,(holding_flag)  ; anything there??
cp      'I'                ; (indicated with 'I')
ret     nz                 ; exit if nothing
ld      a,'O'              ; or load O to invalidate it
ld      (holding_flag),a
ld      ix,text_rx
jp      tx_request
;*****
clear_reg:
ld      a,0
ld      (ix+0),a
ld      (ix+1),a
ret
;*****
vectorin:
ld      a,(rx_holding+3)  ; vectorin routine load iy with the
select('P',present)     ; address of the 16 bit (2 byte)
select('F',future)       ; variable to be acted on or read out
select('O',offset)       ; this holds all the data to be changed
select('S',h1)           ; save for the move register, which
select('T',h2)           ; is acted on manually
select('G',h3)
select('N',0FFF0h)
select('W',0FFF2h)
select('H',0FFF4h)
select('M',delay_mech)
select('I',delay_inert)
select('A',delay_arb)
select('U',delay_out)
select('K',okflag)
select('D',intcount)
ld      iy,0FFFEh        ; sent to dud space
ret                       ; and return. this MAY NOT
                          ; be a good idea..
;*****
testcode:
ld      a,'P'
ld      (rx_holding+3),a
call    dispres
call    print_text

```

```

    ld    ix,text_rtn
    call  tx_request
    call  print_text
    jp    testcode
;*****
movemon:
    ld    a,(moveflag)      ; is automove flag set??
    cp    0                  ; automatic if! =0
    ret   z                  ; it isn't, so exit - axis can move manually
    ld    a,(okflag)        ; is unit OK??
    cp    0                  ; bugged if 0
    jp    z,movemon1        ; yes, so stop all motors
    ld    hl,(offset)        ; is slew hysteresis > offset
    ld    de,(hl)           ; hl holds the slew hysteresis
    and   a                  ; clear carry flag
    sbc   hl,de              ; subtract it and look for carry
    jp    nc,slewon          ; OK,so slew
    ld    a,0                ; reset automove flag
    ld    (moveflag),a      ; save in register
    ld    ix,text_ok         ; send message saying drive stopped
    call  tx_request         ; and transmit
movemon1:
    ld    a,0FFh            ; else switch off everything
    ld    (base_io+direction),a
    ret
;*****
limitmon:
    ld    a,(0FFF4h)        ; get limits from input
    and   3                  ; and mask out all others..
    cp    3                  ; are all bits set? - no limits hit
    ret   z
    ld    ix,text_hit
    call  tx_request
    call  print_text
    jp    starttest         ; limit hit, recentre
;*****
slewon:
    ld    a,(offset_dir)    ; get offset direction
    cp    00h                ; which direction is it going in??
    jp    z,slewon1         ; CCW, so next routine
    ld    a,0FCh            ; get slew output
    ld    (base_io+direction),a ; and load CW direction
    ret                      ; and exit
slewon1:
    ld    a,0FDh            ; Motor CCW
    ld    (base_io+direction),a ; and load
    ret

```

```

;*****
moveint:
    ld    a,01h                ; preload automove as active
    ld    (moveflag),a        ; and store
    ld    a,(rx_holding+3)    ; read in control byte
    cp    'G'                  ; automove - let computer go
    jp    z,moveint1         ; yep, so return
    ld    a,00h                ; automove has been canceled, so stop
    ld    (moveflag),a        ; and store
    ld    a,(rx_holding+3)    ; read in control byte
    ld    b,0FCh              ; preload with set -
    cp    '1'                  ; is it CW
    jp    z,manmove1         ; yes, so exit
    ld    b,0FDh              ; preload with guide -
    cp    '2'                  ; is it CCW
    jp    z,manmove1
    ld    b,0FFh              ; stop if not
manmove1:
    ld    a,b                  ; swap for load
    ld    (base_io+direction),a; and load
moveint1:
    jp    ret_com             ; and exit
;*****
centrego:
    call  centre              ; centre tangent arm
    jp    ret_com             ; and clear rx buffer
;*****
centre:
    ld    a,(0FFF2h)          ; get centre optoswitch
    and   1
    ld    b,a                  ; store in b ( save for later)
    ld    c,0FCh              ; preload c with direction of fine motor
    bit   0,a                  ; which side of centre is tangent arm on
    jp    nz,centre1         ; this way, so run motor
    ld    c,0FDh              ; no, other way,
                                ;so reload c with correct dir
centre1:
    ld    a,c                  ; swap a,c for load
    ld    (base_io+direction),a; start tangent arm motor up
    ld    a,(0FFF2h)          ; get centre optoswitch (again)
    and   1                    ; clear everything else
    xor   b                    ; has it changed???
    jp    z,centre1          ; no, so continue
    ld    a,0FFh              ; yes, centre has been reached
    ld    (base_io+direction),a; so stop motor and exit
    ret
;*****

```

```

inppres:
    call    vectorin        ; set vector to destination
    call    clear_reg       ; clear register
    ld     ix,rx_holding+4  ; set pointer to start of 1's &0's
    ld     b,010h          ; set repeat loop for 16 (2 byte)
inppres1:
    ld     a,(ix+0)        ; load in bit to be tested
    cp     '1'             ; is it a '1'??
    scf                                ; preload with a carry set
    jp     z,inppres2      ; and if it is, start rolling
    and    a                ; else clear flag
inppres2:
    rl     (iy+0)          ; rotate carry into memory location
    rl     (iy+1)          ; and through into high byte
    inc    ix              ; next rx holding buffer
    djnz  inppres1
    jp     ret_com
;*****
dispres:
    ld     ix,text_16      ; create 16 binary mask
    call   tx_request      ; and validate it for tx'ing
    call   vectorin        ; get input vector to load data from
dispres5:
    ld     ix,tx_buffer    ; set pointer to start of tx buffer
    ld     a,(iy+1)        ; load high byte of display buffer => a
    ld     b,8             ; set loop to shift 8 bits
dispres2:
    sla   a                ; shift top bit into carry
    jp    nc,dispres1      ; if not '1' next shift
    inc   (ix+3)           ; it's a '1', so inc '0' => '1'
dispres1:
    inc   ix               ; next bit in tx buffer
    djnz dispres2          ; repeat 8 times
    ld   a,(iy+0)          ; load low byte of display buffer => a
    ld   b,8               ; set loop to shift 8 bits
dispres4:
    sla   a                ; shift top bit into carry
    jp    nc,dispres3      ; if not '1' next shift
    inc   (ix+3)           ; it's a '1', so inc '0' => '1'
dispres3:
    inc   ix               ; next bit in tx buffer
    djnz dispres4          ; repeat 8 times
    jp    ret_clear
;*****
; This routine does all the delays for the program, and all the delays can be
; loaded by the user at runtime. All default delays are loaded in setup and are
; start and then modified by user. NOTE: changing delay times can seriously

```

; affect the performance of the system!!!. Think about it. delay time in uS

; delay time = 2.5+335.5\*HL uS

delay:

ld de,01

delay1:

ld b,0ffh

delay2:

djnz delay2

and a

sbc hl,de

jp nz,delay1

ret

;\*\*\*\*\*

ret\_com:

ld ix,text\_ack

call tx\_request

ret\_clear:

ld a,'O'

ld (holding\_flag),a

ld ix,rx\_holding

jp clear\_buffer

;\*\*\*\*\*

clear\_buffer:

ld b,250

ld a,0

clear1:

ld (ix+0),a

inc ix

djnz clear1

ret

;\*\*\*\*\*

tx\_request:

ld iy,tx\_buffer

tx\_req2:

ld a,(ix+0)

cp chr\_null

jp nz,tx\_req1

ld a,chr\_null

ld (iy+0),a

ret

tx\_req1:

ld (iy+0),a

inc ix

inc iy

jp tx\_req2

;\*\*\*\*\*

; this routine holds all the protocol for transmitting a data

; packet, including txrx arbitration, etc

print\_text:

```
ld    ix,tx_buffer      ; get start of tx buffer
ld    a,(ix+0)          ; get valid text line character
cp    'S'                ; is this character = valid??
ret   nz                 ; no, so exit
sio_in(0)                ; get cts bit
bit   5,a                ; is cts set??
; ret   z                 ; txrx line active - so quit
sio_ctrl(5,11101000b)    ; set txrx control active
ld    hl,(delay_arb)
call  delay              ; delay for a preset time
sio_ctrl(5,01101000b)    ; set txrx control inactive
ld    hl,(delay_cap)
call  delay              ; delay for capacitor
sio_in(0)                ; get cts bit
bit   5,a                ; is cts set??
ret   z                 ; yes, so get off line
sio_ctrl(5,11101000b)    ; switch on txrx line for good
sio_ctrl(3,11000000b)    ; shut off receiver
sio_ctrl(1,00000000b)    ; switch off interrupts
```

print\_t2:

```
ld    a,(ix+0)          ; a < first character of line
cp    chr_null           ; is it the end of the message
jp    nz,print_t1        ; no, so print message
ld    a,chr_null         ; end of message, so send null
call  print_a            ; and print it.
```

print\_t3:

```
sio_in(0)                ; get tx buffer empty
bit   2,a                ; is tx buffer empty?
jp    nz,print_t3        ; no, so wait
```

print\_t4:

```
sio_in(1)                ; get all_sent bit
bit   0,a                ; is it all sent
jp    nz,print_t4        ; no, so wait
ld    a,'*'              ; signal with any char
ld    ix,tx_buffer       ; like *
ld    (ix+0),a           ; that message is sent
ld    hl,(delay_out)
call  delay
sio_ctrl(5,01101010b)    ; set txrx control inactive
in    a,(sio_data_a)     ; clear any characters held
in    a,(sio_data_a)     ; in holding buffer
in    a,(sio_data_a)     ;
in    a,(sio_data_a)     ;
sio_ctrl(3,11000001b)    ; and switch on receiver
sio_ctrl(1,00011000b)    ; and enable interrupts
```

```

        ret                ; return
print_t1:
        call   print_a      ; print character
        inc   ix            ; increment pointer
        jp    print_t2      ; and repeat
;*****
; This routine must not be called to send a character only.
; it will violate the network - no TXRX control
print_a:
        ld    b,a          ; b <= a (store it temp)
print_1:
        sio_in(0)         ; get tx empty (bit status)
        bit   2,a          ; empty ??
        jp   z,print_1     ; no, so loop until it is
        ld   a,b           ; transfer character back to a
        out  (sio_data_a),a ; and send it to the comms
        ret                ; and return
;*****
data_table:
        .byte 00000000b
        .byte 00000001b
        .byte 00000011b
        .byte 00000010b
        .byte 00000000b
        .byte 00000001b
        .byte 00000011b
        .byte 00000010b
        .byte 00000000b
;*****
diag1 .text "SI"
        .byte host
        .text "SYSTEM_NODE_ACK"
        txt_end

diag2 .text "SI"
        .byte host
        .text "SYSTEM_TEST_OK"
        txt_end

text_ack .text "SI"
        .byte host
        .text "ACK"
        txt_end

text_on .text "SI"
        .byte host
        .text "NODE_ON"

```

```

txt_end

text_hit .text "SI"
        .byte host
        .text "LIMIT_HIT_ERR"
txt_end

text_rtn .text "SI"
        .byte host
        .byte 10
        .byte 13
txt_end

text_ok .text "SI"
        .byte host
        .text "OK"
txt_end

text_rx .text "SI"
        .byte host
        .text "RX"
txt_end

text_d .text "SI"
        .byte host
        .text "D000ACK"
txt_end

text_input: .text "SI"
        .byte host
        .text "00000000ACK"
txt_end

text_16: .text "SI"
        .byte host
        .text "0000000000000000ACK"
txt_end

text_err .text "SI"
        .byte host
        .text "ERR"
txt_end
;*****
.end

```

## C.4 : Source code for DOME axis microcontroller

Filename:	dome.z80
Application:	Source code for dome axis Z80 microcontroller

```
;  
;  
; ST. ANDREWS TWIN PHTOMETRIC TELESCOPE  
;  
; DOME AXIS DRIVE PROGRAM  
;  
; 150993V5.0/R.T.GEARS/TPT-TECH.DOC  
;  
;  
; Set host to axis  
host      .equ   dome      ; define dome id  
;  
; Load the include files  
#include  "c:\\design\\scma\\z80\\set_scma.pre"  
#include  "c:\\design\\scma\\z80\\rambo.h"  
#include  "c:\\design\\scma\\z80\\char.h"  
#include  "c:\\design\\scma\\z80\\map_io.h"  
#include  "c:\\design\\scma\\z80\\memory.h"  
#include  "c:\\design\\scma\\z80\\macro_io.def"  
;*****  
start    .org   0000h  
        di                      ; Processor startup code  
        im    2                  ; interrupts from table  
        ld    hl,stack           ; load stack from memory.h  
        ld    sp,hl              ; and load  
        ld    a,(int_table&0ff00h) >> 8 ; load I reg with table addr.  
        ld    i,a                ; load  
        call  io_setup           ; setup on chip peripherals  
        jp    start_main        ; and go  
;*****  
        .org   038h  
rx_test:  
        push  af  
        push  bc  
        push  de  
        push  hl  
        push  ix  
        push  iy  
rx_test1:  
        sio_in(0)                ; wait for character to be received  
        bit   0,a                 ; mask off  
        jp    z,rx_test1         ; and loop if not yet available  
rx_test6:  
        ld    ix,rx_buffer        ; get start of buffer  
        ld    iy,rx_cursor       ; get cursor pointer  
        ld    a,(rx_cursor)      ; get cursor
```

```

        ld    b,a
rx_test4:
        inc   ix                ; move ix to rx cursor
        djnz  rx_test4         ; and repeat
        dec   ix                ; correct for initial loop
rx_test3:
        sio_in(0)              ; is there a character to be received??
        bit   0,a               ; mask off
        jp    z,rx_test2       ; no, so exit
        in    a,(sio_data_a)   ; yes, so load data
        ld    (ix+0),a         ; and store in rx_buffer
        cp    'Z'              ; end of message (chr_cr)
        jp    nz,rx_test5      ; carry on if not end of message
        ld    a,1               ; reset cursor
        ld    (iy+0),a         ; and load iy into rx_cursor
        ld    a,(rx_buffer)    ; is it for the host??
        cp    host             ; use host
        jp    nz,rx_test7      ; no, so just delete
        ld    a,(rx_buffer+1)  ; is the remote host the IBM??
        cp    'I'
        jp    nz,rx_test7      ; no,so delete
        ld    de,rx_holding    ; load holding reg with rx buffer
        ld    hl,rx_buffer     ; (de) <= (hl) for bc times
        ld    bc,30            ; repeat 30 times
        ldir
        ld    a,'I'            ; signal new command
        ld    (holding_flag),a
rx_test7:
        ld    ix,rx_buffer     ; clear up
        call  clear_buffer     ; clear rx_buffer
        jp    rx_test2         ; and exit
rx_test5:
        inc   (iy+0)           ; next location
        jp    rx_test6         ; and loop
rx_test2:
        sio_ctrl(0,038h)       ; send a return from interrupt
        sio_ctrl(0,020h)       ; and enable on next rx character
        pop   iy
        pop   ix
        pop   hl
        pop   de
        pop   bc
        pop   af
        ei
        reti
;*****
etc_int:

```

```

push  af          ; save registers used
push  bc          ; push all registers on stack
push  de
push  hl
push  ix
push  iy
ld    a,(oldencoder) ; get previous encoder reading
ld    c,a         ; c = old encoder reading
ld    a,(encoder_io) ; get new encoder reading from I/O port
and   07h        ; mask out unused high bits
ld    (newencoder),a ; and store in new_encoder memory location
cp    c          ; is old_encoder = new_encoder (no move)
jp    z,ctcint7  ; dome 'stationary' during update, so exit.
bit   2,a        ; is sync bit set
                    ;( dome going through 0/360 Deg)
jp    z,ctcint2  ; no, so get inc/dec status
ld    a,(dome_direction) ; sync pulse, so determine whether to set or
                    ;clear
ld    hl,0000h   ; preload hl with clear, a with dome
                    ; direction, and
cp    01h        ; if dome_direction=1 (+ve) clear with hl
jp    z,ctcint3  ; clear, using preloaded hl
ld    hl,0059Fh  ; dome here going -ve so load hl with 4 *360-1
ctcint3:
ld    (present),hl ; load present location with set/clear hl reading
ld    a,1         ; and Indicate that dome is OK (valid)
ld    (okflag),a  ; load valid flag
jp    ctcint7     ; and jump to calculate offset routine
ctcint2:
; Encoder has incremented/decremented,so look up old encoder
; in data table, and see which side of it is the new reading
ld    ix,data_table-1 ; point ix to start of incremental cycle table
ld    a,(oldencoder) ; get old encoder reading
and   03h        ; mask out all but phase quad information
                    ; from the encoder
ctcint4:
inc   ix         ; next phase quad data table location
cp    (ix+1)     ; Data entry == old_encoder ??
jp    nz,ctcint4 ; no so repeat (max 4 times) :
                    ;can only be 00 01 11 10
; data table pointer (ix) now pointing to old Phase
                    ; now compare either side to the new Phase
ld    a,(newencoder) ; load a with new encoder,
                    ; to enable compare with ix+?
and   03h        ; mask out all but phase quad
                    ; information in memory
cp    (ix+2)     ; is the dome going cw (+ve)

```

```

    jp    z,ctcint5      ; yes, so jump to cw routine
    cp    (ix+0)        ; is dome going ccw (-ve)
    jp    z,ctcint6      ; yes, so jump to ccw routine.
;   ld    a,0          ; if it's not going cw/ccw (as above)
;                       ; it's buggered.
    ld    a,1          ; if it's not going cw/ccw (as above)
;                       ; it's buggered.
    ld    (okflag),a    ; so clear ok flag -
;                       ; show that encoder is knackered
    jp    ctcint7      ; and exit (will calculate offset,
;                       ; but ok flag overrides)

ctcint5:
    ld    a,01h        ; set the direction = cw
    ld    (dome_direction),a ; and save in encoder direction,
;                       ; for use by interrupt routine
    ld    hl,(present) ; get current position
    inc   hl           ; and increment
    ld    (present),hl ; and store back again
    jp    ctcint7      ; jump to exit saving old encoder

ctcint6:
    ld    a,0          ; set the direction = ccw
    ld    (dome_direction),a ; and save in encoder direction,
;                       ; for use by interrupt routine
    ld    hl,(present) ; get current position
    dec   hl           ; and decrement
    ld    (present),hl ; and store back again

ctcint7:
    ld    hl,(present) ; compare present to future
    ld    de,(future)  ; to get direction of offset
    and   a            ; clear carry flag
    sbc   hl,de        ;
    jp    c,ctcint9    ; jump if future < present
    ld    (offset),hl  ; save offset in memory
    ld    a,0          ; and set direction to ??
    ld    (offset_dir),a
    jp    ctcint10

ctcint9:
    ld    hl,(future)  ; here swap future and present registers
    ld    de,(present) ; and here too
    and   a            ; clear carry flag
    sbc   hl,de        ;
    ld    (offset),hl  ; save offset in memory
    ld    a,1          ; and invert direction
    ld    (offset_dir),a

ctcint10:
; this extra bit to the calcoffset routine is due to the cyclic nature of dome.
    ld    de,720      ; is offset > 1/2 turn

```

```

and    a                ; clear carry for subtraction
sbc    hl,de            ; subtract it
jp     c,ctcint1       ; all ok, offset < 1/2 turn
ld     hl,1440         ; else modify offset
ld     de,(offset)     ;
sbc    hl,de           ; offset = 360+4-offset
ld     (offset),hl     ; and save
ld     (offset_dir),a  ; get direction
xor    01h             ; and invert direction
ld     a,(offset_dir) ; and save
ctcint1:
ld     a,(newencoder)  ; shift new encoder => old encoder
ld     (oldencoder),a ; and move new encoder -> old
pop    iy
pop    ix
pop    hl              ; clear up and exit
pop    de
pop    bc
pop    af
ei                      ; and enable interrupts
reti                   ; and exit
;*****
unused:
ei
reti
;*****
.org (0ff00h & ($ + 0ffh))
io_setup:
#include "c:\\design\\scma\\z80\\setdo_io.rel"
ret
;*****
.org (0ff00h & ($ + 0ffh))
; Interrupt table in here
int_table:
int_table_ctc:
.byte unused & 0ffh
.byte (unused & 0ff00h) >> 8
.byte unused & 0ffh
.byte (unused & 0ff00h) >> 8
.byte ctc_int & 0ffh
.byte (ctc_int & 0ff00h) >> 8
.byte unused & 0ffh
.byte (unused & 0ff00h) >> 8
.byte unused & 0ffh
.byte (unused & 0ff00h) >> 8
int_table_sio:
.byte rx_test & 0ffh

```

```

        .byte (rx_test & 0ff00h) >> 8
;*****
        .org (0ff00h&($+00ffh))
start_main:
        call setup                ; setup memory, ports default values
        ei                        ; enable interrupts
        sio_ctrl(5,01101000b)    ; testd
sio_ctrl(0,020h)                  ; enable rx on next character
        ld ix,text_on
        call tx_request
main_loop:
        call movemon
        call rx_mon
        call print_text
        jp main_loop
;*****
starttest:
        ld hl,14904d              ; Delay startup 5 sec
        call delay
        ld a,0
        ld (okflag),a            ; invalidate ok flag
        ld a,0FFh
        ld (base_io+motor),a
        ld a,0FEh                ; switch on dome CW fast!!!
        ld (base_io+direction),a
starttest1:
        ld a,(okflag)            ; wait for flag to be updated
        cp 1
        jp nz,starttest1
        ld a,0FFh                ; switch off all valves
        ld (base_io+motor),a
        ld (base_io+direction),a
        ld hl,2981d              ; delay for 1 sec
        call delay
        ld ix,diag2              ; finished, so send next messages
        call tx_request          ; tx
        ld hl,(present)         ; lock present & future together
        ld (future),hl
        call ret_clear
        ld ix,diag2              ; finished, so send next messages
        call tx_request          ; tx
        ret
;*****
setup:
        ld a,0FFh                ; shut off all motors, whatever state
        ld (base_io+motor),a    ; motors off
        ld (base_io+direction),a; direction relays off

```

```

ld    ix,present
call  clear_reg
ld    ix,offset
call  clear_reg
ld    ix,future
call  clear_reg
ld    hl,0FA14h      ; load in int counters
ld    (intcount),hl ; and store
ld    hl,20          ; Load default slew hysteresis (5 deg)
ld    (h1),hl
ld    hl,10          ; Load default set hysteresis (2.5 deg)
ld    (h2),hl
ld    hl,4d          ; Load default guide hysteresis (1deg)
ld    (h3),hl
ld    hl,299         ; 100mS delay
ld    (delay_mech),hl ; for mechanical delay
ld    hl,1491        ; 500mS delay
ld    (delay_over),hl ; for telescope inertia
ld    hl,1491        ; 500mS delay
ld    (delay_inert),hl ; for overload inertia
ld    hl,1491        ; 500mS delay
ld    (delay_turn),hl ; for rewinding clamp 1/4 turn
ld    hl,000FFh      ; 85 mS arbitration delay
ld    (delay_arb),hl ; for arbitration delay
ld    hl,000FFh      ; 85mS (seems like a good figure)
ld    (delay_out),hl ; for tx inactive settling time
ld    hl,01          ; 338uS
ld    (delay_cap),hl ; for capactive decay in comms line
ld    a,0            ; Set slew flag to 0
ld    (slewflag),a
ld    (moveflag),a   ; and switch move to manual
ld    (okflag),a     ; invalidiate encoder ok flag
ld    (limitflag),a  ; clear limits
ld    a,1            ; Set cursor to start of rx buffer
ld    (rx_cursor),a
ld    ix,rx_buffer
call  clear_buffer
ld    ix,rx_holding
call  clear_buffer
ld    a,'O'
ld    (holding_flag),a
ret

; *****
; sort out the commands here -scrap below
rx_mon:
    ld    a,(rx_holding+2)
    cp    'Q'

```

```

jp    z,testcode
cp    'M'
jp    z,moveint
cp    'D'
jp    z,dispres
cp    'P'
jp    z,inppres
cp    'R'                ; soft reset, start again
jp    z,starttest
cp    'H'
jp    z,start           ; hard reset
ld    a,(holding_flag) ; anything there??
cp    'I'                ; (indicated with 'I')
ret   nz                ; exit if nothing
ld    a,'O'             ; or load O to invalidate it
ld    (holding_flag),a
ld    ix,text_rx
jp    tx_request
;*****
clear_reg:
ld    a,0
ld    (ix+0),a
ld    (ix+1),a
ret
;*****
vectorin:
ld    a,(rx_holding+3) ; vectorin routine load iy with the
select('P',present)   ; address of the 16 bit (2 byte)
select('F',future)    ; varible to be acted on or read out
select('O',offset)    ; this holds all the data to be changed
select('S',h1)        ; save for the move register, which
select('T',h2)        ; is acted on manually
select('G',h3)
select('N',0FFF0h)
select('W',0FFF2h)
select('H',0FFF4h)
select('M',delay_mech)
select('I',delay_inert)
select('A',delay_arb)
select('U',delay_out)
select('K',okflag)
select('D',intcount)
ld    iy,0FFFEh       ; sent to dud space
ret                    ; and return. this MAY NOT
                       ; be a good idea.
;*****
testcode:

```

```

ld    a,'P'
ld    (rx_holding+3),a
call  dispres
call  print_text
ld    ix,text_rtn
call  tx_request
call  print_text
jp    testcode
;*****
movemon:
ld    a,(moveflag)      ; is automove flag set??
cp    0                  ; automatic if! =0
ret   z                  ; it isn't, so exit - axis can move manually
ld    a,(okflag)        ; is unit OK??
cp    0                  ; bugged if 0
jp    z,movemon1        ; yes, so stop all motors
ld    hl,(offset)       ; is slew hysteresis > offset
ld    de,(h1)           ; h1 holds the slew hysteresis
and   a                  ; clear carry flag
sbc   hl,de             ; subtract it and look for carry
jp    nc,slewon         ; OK, so slew
ld    hl,(offset)       ; is fine set hysteresis > offset
ld    de,(h2)           ; h1 holds the fine set hysteresis
and   a                  ; clear carry
sbc   hl,de             ; and subtract
jp    nc,seton          ; Ok, so set on
ld    hl,(offset)       ; is fine guide hysteresis > offset
ld    de,(h3)           ; h1 holds the fine guide hysteresis
and   a                  ; clear carry
sbc   hl,de             ; and subtract
jp    nc,guideon        ; Ok, so guide on
ld    a,0                ; reset automove flag
ld    (moveflag),a      ; save in register
ld    ix,text_ok        ; send message saying drive stopped
call  tx_request        ; and transmit
movemon1:
ld    a,0FFh            ; else switch off everything
ld    (base_io+motor),a
ld    (base_io+direction),a
ret
;*****
slewon:
ld    a,(offset_dir)    ; get offset direction
cp    00h               ; which direction is it going in??
jp    z,slewon1        ; CCW, so next routine
ld    a,0FEh           ; get slew output
ld    (base_io+direction),a ; and load CW direction

```

```

    ld    a,0FFh          ; clear outputs to CCW direction
    ld    (base_io+motor),a ; and load
    ret                                ; and exit
slewon1:
    ld    a,0FFh          ; clear outputs to CW direction
    ld    (base_io+direction),a ; and load
    ld    a,0FEh          ; get slew output to CCW
    ld    (base_io+motor),a ; and load
    ret
;*****
seton:
    ld    a,(offset_dir)   ; get offset direction
    cp    00h              ; which direction is it going in??
    jp    z,seton1         ; CCW, so next routine
    ld    a,0FDh           ; get set output
    ld    (base_io+direction),a ; and load CW valves
    ld    a,0FFh           ; clear outputs to CCW direction
    ld    (base_io+motor),a ; and load
    ret                                ; and exit
seton1:
    ld    a,0FFh           ; clear CW outputs
    ld    (base_io+direction),a ; and load
    ld    a,0FDh           ; get set output to CCW
    ld    (base_io+motor),a ; and load
    ret
;*****
guideon:
    ld    a,(offset_dir)   ; get offset direction
    cp    00h              ; which direction is it going in??
    jp    z,guideon1      ; CCW, so next routine
    ld    a,0FBh           ; get guide output
    ld    (base_io+direction),a ; and load CW valves
    ld    a,0FFh           ; clear outputs to CCW
    ld    (base_io+motor),a ; and load
    ret                                ; and exit
guideon1:
    ld    a,0FFh           ; clear outputs to CW
    ld    (base_io+direction),a ; and load
    ld    a,0FDh           ; get guide output CCW
    ld    (base_io+motor),a ; and load
    ret
;*****
moveint:
    ld    a,01h            ; preload automove as active
    ld    (moveflag),a     ; and store
    ld    a,(rx_holding+3) ; read in control byte
    cp    'G'              ; automove - let computer go

```

```

jp      z,moveint1      ; yep, so return
ld      a,00h           ; automove has been canceled, so stop
ld      (moveflag),a    ; and store
ld      a,(rx_holding+3) ; read in control byte
ld      b,0FEh         ; preload with set -
cp      '1'            ; is it set- ('1')
jp      z,manmove1     ; yes, so exit
ld      b,0FDh         ; preload with guide -
cp      '2'            ; is it guide- ('2')
jp      z,manmove1     ; yes, so exit
ld      b,0FBh         ; preload with guide +
cp      '3'            ; is it guide + ('3')
jp      z,manmove1     ; yes, so exit
ld      b,0FEh         ; preload with set +
cp      '4'            ; is it set + ('4')
jp      z,manmove2     ; yes, so exit
ld      b,0FDh         ; preload b with slew direction CW
cp      '5'            ; is it CW slew??
jp      z,manmove2     ; yes, so start
ld      b,0FBh         ; preload b with slew direction CCW
cp      '6'            ; is it CCW slew??
jp      z,manmove2     ; yes, so start
ld      b,0FFh         ; stop if not

manmove1:
ld      a,b             ; swap for load
ld      (base_io+direction),a ; and load
ld      a,0FFh         ; load CCW all off
ld      (base_io+motor),a ; and load
jp      ret_com        ; and exit

manmove2:
ld      a,b             ; swap for load
ld      (base_io+motor),a ; and load
ld      a,0FFh         ; load CCW all off
ld      (base_io+direction),a ; and load

moveint1:
jp      ret_com        ; and exit
;*****
inppres:
call    vectorin       ; set vector to destination
call    clear_reg      ; clear register
ld      ix,rx_holding+4 ; set pointer to start of 1's &0's
ld      b,010h         ; set repeat loop for 16 (2 byte)

inppres1:
ld      a,(ix+0)       ; load in bit to be tested
cp      '1'            ; is it a '1'??
scf     ; preload with a carry set
jp      z,inppres2     ; and if it is, start rolling

```

```

        and    a            ; else clear flag
inppres2:
    rl      (iy+0)        ; rotate carry into memory location
    rl      (iy+1)        ; and through into high byte
    inc     ix            ; next rx holding buffer
    djnz   inppres1
    jp     ret_com
;*****
dispres:
    ld      ix,text_16    ; create 16 binary mask
    call   tx_request     ; and validate it for tx'ing
    call   vectorin      ; get input vector to load data from
dispres5:
    ld      ix,tx_buffer  ; set pointer to start of tx buffer
    ld      a,(iy+1)     ; load high byte of display buffer => a
    ld      b,8          ; set loop to shift 8 bits
dispres2:
    sla     a            ; shift top bit into carry
    jp     nc,dispres1   ; if not '1' next shift
    inc     (ix+3)       ; it's a '1', so inc '0' => '1'
dispres1:
    inc     ix           ; next bit in tx buffer
    djnz   dispres2     ; repeat 8 times
    ld     a,(iy+0)     ; load low byte of display buffer => a
    ld     b,8          ; set loop to shift 8 bits
dispres4:
    sla     a            ; shift top bit into carry
    jp     nc,dispres3   ; if not '1' next shift
    inc     (ix+3)       ; it's a '1', so inc '0' => '1'
dispres3:
    inc     ix           ; next bit in tx buffer
    djnz   dispres4     ; repeat 8 times
    jp     ret_clear
;*****
; This routine does all the delays for the program, and all the delays can be
; loaded by the user at runtime. All default delays are loaded in setup at the
; start and then modified by user. NOTE: changing delay times can seriously
; affect the performance of the system!!!. Think about it. delay time :
; delay time = 2.5+335.5*HL uS
delay:
    ld     de,01
delay1:
    ld     b,0ffh
delay2:
    djnz   delay2
    and    a
    sbc   hl,de

```

```

        jp    nz,delay1
        ret
;*****
ret_com:
        ld    ix,text_ack
        call  tx_request
ret_clear:
        ld    a,'O'
        ld    (holding_flag),a
        ld    ix,rx_holding
        jp    clear_buffer
;*****
clear_buffer:
        ld    b,250
        ld    a,0
clear1:
        ld    (ix+0),a
        inc  ix
        djnz clear1
        ret
;*****
tx_request:
        ld    iy,tx_buffer
tx_req2:
        ld    a,(ix+0)
        cp    chr_null
        jp    nz,tx_req1
        ld    a,chr_null
        ld    (iy+0),a
        ret
tx_req1:
        ld    (iy+0),a
        inc  ix
        inc  iy
        jp    tx_req2
;*****
; this routine holds all the protocol for transmitting a data
; packet, including txrx arbitration, etc
print_text:
        ld    ix,tx_buffer           ; get start of tx buffer
        ld    a,(ix+0)              ; get valid text line character
        cp    'S'                   ; is this character = valid??
        ret  nz                     ; no, so exit
        sio_in(0)                   ; get cts bit
        bit   5,a                   ; is cts set??
        sio_ctrl(5,11101000b)      ; set txrx control active
        ld    hl,(delay_arb)

```

```

call    delay                ; delay for a preset time
sio_ctrl(5,01101000b)      ; set txrx control inactive
ld      hl,(delay_cap)
call    delay                ; delay for capacitor
sio_in(0)                  ; get cts bit
bit     5,a                 ; is cts set??
ret     z                   ; yes, so get off line
sio_ctrl(5,11101000b)      ; switch on txrx line for good
sio_ctrl(3,11000000b)      ; shut off receiver
sio_ctrl(1,00000000b)      ; switch off interrupts

print_t2:
ld      a,(ix+0)            ; a < first character of line
cp      chr_null            ; is it the end of the message
jp      nz,print_t1         ; no, so print message
ld      a,chr_null          ; end of message, so send null
call    print_a             ; and print it.

print_t3:
sio_in(0)                  ; get tx buffer empty
bit     2,a                 ; is tx buffer empty?
jp      nz,print_t3         ; no, so wait

print_t4:
sio_in(1)                  ; get all_sent bit
bit     0,a                 ; is it all sent
jp      nz,print_t4         ; no, so wait
ld      a,'*'               ; signal with any char
ld      ix,tx_buffer        ; like *
ld      (ix+0),a            ; that message is sent
ld      hl,(delay_out)
call    delay
sio_ctrl(5,01101010b)      ; set txrx control inactive
in      a,(sio_data_a)      ; clear any characters held
in      a,(sio_data_a)      ; in holding buffer
in      a,(sio_data_a)      ;
in      a,(sio_data_a)      ;
sio_ctrl(3,11000001b)      ; and switch on reciever
sio_ctrl(1,00011000b)      ; and enable interrupts
ret                                ; return

print_t1:
call    print_a             ; print character
inc     ix                  ; increment pointer
jp      print_t2            ; and repeat
;*****
; This routine must not be called to send a character only.
; it will violate the network - no TXRX control
print_a:
ld      b,a                 ; b <= a (store it temp)
print_1:

```

```

sio_in(0)          ; get tx empty (bit status)
bit    2,a         ; empty ??
jp     z,print_1   ; no, so loop until it is
ld     a,b         ; transfer character back to a
out    (sio_data_a),a ; and send it to the comms
ret                                ; and return
;*****
data_table:
    .byte 00000000b
    .byte 00000001b
    .byte 00000011b
    .byte 00000010b
    .byte 00000000b
    .byte 00000001b
    .byte 00000011b
    .byte 00000010b
    .byte 00000000b
;*****
diag1 .text "SI"
      .byte host
      .text "SYSTEM_NODE_ACK"
      txt_end

diag2 .text "SI"
      .byte host
      .text "SYSTEM_TEST_OK"
      txt_end

text_ack .text "SI"
        .byte host
        .text "ACK"
        txt_end

text_on .text "SI"
        .byte host
        .text "NODE_ON"
        txt_end

text_hit .text "SI"
        .byte host
        .text "LIMIT_HIT_ERR"
        txt_end

text_rtn .text "SI"
        .byte host
        .byte 10
        .byte 13

```

```

txt_end

text_ok      .text  "SI"
             .byte  host
             .text  "OK"
             txt_end

text_rx      .text  "SI"
             .byte  host
             .text  "RX"
             txt_end

text_d .text  "SI"
             .byte  host
             .text  "D000ACK"
             txt_end

text_input:  .text  "SI"
             .byte  host
             .text  "00000000ACK"
             txt_end

text_16:     .text  "SI"
             .byte  host
             .text  "0000000000000000ACK"
             txt_end

text_err     .text  "SI"
             .byte  host
             .text  "ERR"
             txt_end
;*****
.end

```

## **Appendix D : C source files for the operator interface modules**

The operator controls the telescope through a console in the warm room. This computer provides the user with multiple modules, each graphically displaying different aspects of the telescope. These modules were developed in C for use on a PC clone running under the *Windows 3.1* operating system.

An Elonex 33MHz 80386DX & 80387DX with 4MB RAM, SVGA and at least 10MB free disk cache was used for development and compilation operating under both DOS 5 and *Windows 3.1* environments. The memory management was provided by the MAX386 memory manager. Other drivers resident in the system included Nortons' anti virus 2.00 and a GeniScan scanner driver.

The modules were compiled using the Microsoft C700 compiler under DOS 5. Compilation under *Windows 3.1* was tried using a Dosshell, but it always returned an 'Out Of Memory' error. This was unexpected as Microsoft claim it should be possible, but was put down to either the physical RAM size or the Norton's anti virus driver.

The above problem was highlighted by running PWB, Microsoft's programmers workbench under DOS. When the Windows library was included the same 'Out Of Memory Error' error code was returned during compilation. Thus all source code was written in EDIT.COM and compiled from the DOS prompt using the NMAKE command.

The debugging utilities for each module were limited to SPY.EXE, DDESPY.EXE and error file reports. While a comprehensive 'snap shot' of each module was unavailable, displaying the message queues to each module outlined the task path and proved extremely useful.

Due to the complexity of the source code all files have been listed here, as subtle switches (such as /nod) produce incomprehensible errors.

The host machine for all modules was a 16Mhz 80386SX + 80387SX 4MB RAM, 40MB free hard and VGA screen. No problems were present in the transfer except that the desktop area had decreased, allowing only 39% of information to be displayed.

## D.1 : TCLCTRL : Make file

Filename:	TCLCTRL.MAK
Application:	make file for C700 complier

TCLCTRL.exe : TCLCTRL.obj TCLCTRL.def TCLCTRL.res

link TCLCTRL, /align:16, NUL, /nod slibcew libw shell,  
TCLCTRL rc TCLCTRL.res

TCLCTRL.obj : TCLCTRL.c TCLCTRL.h TCLCTRL.ico  
cl -c -Gsw -Ow -W2 -Zi TCLCTRL.c

TCLCTRL.res : TCLCTRL.rc TCLCTRL.h TCLCTRL.ico  
rc -r TCLCTRL.rc

## D.2 : TCLCTRL : Definition file

Filename:	TCLCTRL.DEF
Application:	Definition file for C700 complier

```
=====
; tclctrl.def module definition file
=====
NAME          TCLCTRL
DESCRIPTION   'TPT TCLCTRL COMMAND MODULE'
EXETYPE       WINDOWS
STUB          'WINSTUB.EXE'
CODE          PRELOAD MOVEABLE DISCARDABLE
DATA         PRELOAD MOVEABLE MULTIPLE
HEAPSIZE      1024
STACKSIZE     8192
EXPORTS       WndProc
              EntryProc
```

### D.3 : TCLCTRL : Include file

Filename:	TCLCTRL.H
Application:	Include file for C700 complier

```
//TCLCTRL include file
#ifndef H_TCLCTRL
#define H_TCLCTRL
#define IDM_ABORT_SOURCE          1
#define IDM_ABORT_COMMAND        2
#define IDM_STOP                  3
#define IDM_ENTRY                 4
#define IDM_EXIT                  5
#define IDM_UPDATE                6
#define IDM_ABOUT                 7
#define ID_TIMER1                 8

long FAR PASCAL WndProc (HWND, WORD, WORD, LONG) ;
long FAR PASCAL EntryProc(HWND,WORD, WORD, LONG) ;
long FAR PASCAL DisplayProc(HWND,WORD, WORD, LONG) ;
#endif
```

## D.4 : TCLCTRL : Source file

Filename:	TCLCTRL.C
Application:	Source file for C700 complier

```
//Telescope Control Module
#include < windows.h >
#include < stdio.h >
#include < stdlib.h >
#include < string.h >
#include < shellapi.h >
#include "tclctrl.h"
#include "global.h"
#include "system.h"
#include "system.c"
;
#define MAX_TEXT      26
#define ID_RAREF      3
#define ID_DECREF     4
#define ID_RAOFF      6
#define ID_DECOFF     7
#define ID_DOME       9
#define ID_FILREF     11
#define ID_FILOFF     12
#define ID_APEREF     14
#define ID_APEOFF     15
#define ID_PATH       16
#define ID_STEP       18
#define ID_HEAD       20
#define ID_INT        22
#define ID_SIDEREAL   24
#define ID_LED        25

UINT SetWindowDisplay(UINT);
UINT GetInt(char* ,UINT);

typedef struct tagVar {
    char      lpszString[20],lpszFile[20];
    UINT      iData,iStatus;
    POINT     TextPos;
    HWND      hwndText;
} VAR;

char*      PointRef(int,int,int,int,int,int);
char*      GetMacro(char* ,char* );
char*      PointOffset(int,int,int,int,int,int);
```

```

char*      GetBinary(unsigned int);
char*      PointDome(int Azimuth);
static FILE *ErrorFile;
FARPROC    lpfnOldEntryProc;
HWND       hwndAbortSource,hwndAbortCommand,hwndStop,hwndEntry;
UINT       GetSidereal(void);
UINT       GetCommand(UINT);
unsigned int TranslateCommand(HWND,char*);
unsigned int TransmitMessage(char*);
static unsigned int
    RaRefEnc=0,RaOffEnc=0,DecRefEnc=0,DecOffEnc=0,Module,
    t=1,TimerFlag=0;
static ATOM Aatom;
static HWND hwndTclLink;
static VAR  varText[MAX_TEXT]=
    {
    {"REFERENCE", "",0,1,{20,1}},
    {"OFFSET", "",0,1,{40,1}},
    {"R.A.", "",0,1,{0,2}},
    {"UNDEFINED", "",43200,1,{20,2}},
    {"UNDEFINED", "",2052,1,{40,2}},
    {"DEC", "",0,1,{0,3}},
    {"UNDEFINED", "",0,1,{20,3}},
    {"UNDEFINED", "",0,1,{40,3}},
    {"DOME", "",0,1,{0,4}},
    {"UNDEFINED", "UMF_DOME_AZIMUTH",0,2,{20,4}},
    {"FILTER", "",0,1,{0,5}},
    {"UNDEFINED", "UMF_REFERENCE_FILTER",0,1,{20,5}},
    {"UNDEFINED", "UMF_OFFSET_FILTER",0,1,{40,5}},
    {"APERTURE", "",0,1,{0,6}},
    {"UNDEFINED", "UMF_REFERENCE_FILTER",0,1,{20,6}},
    {"UNDEFINED", "UMF_OFFSET_FILTER",0,1,{40,6}},
    {"STAR/SKY", "UMF_PATH_STATE",0,1,{0,7}},
    {"STEPS", "",0,1,{20,7}},
    {"UNDEFINED", "UMF_PATH_STEP",0,2,{40,7}},
    {"CAMERA", "",0,1,{0,8}},
    {"UNDEFINED", "UMF_CAMERA_HEAD",0,1,{20,8}},
    {"INTEGRATION TIME", "",0,1,{0,9}},
    {"UNDEFINED", "",0,2,{20,9}},
    {"SIDEREAL", "",0,1,{0,10}},
    {"UNDEFINED", "",0,2,{20,10}},
    {"LED:", "",0,1,{30,10}}
    };

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpszCmdParam, int nCmdShow)
{

```

```

static char    szAppName[] = "tclctrl" ;
FARPROC      lpfnEntryProc;
HWND         hwnd;
MSG          msg;
WNDCLASS     wndclass;
UINT         i;

if(hPrevInstance)return FALSE;

wndclass.style          = CS_HREDRAW | CS_VREDRAW;
wndclass.lpfnWndProc    = WndProc ;
wndclass.cbClsExtra     = 0;
wndclass.cbWndExtra     = 0;
wndclass.hInstance     = hInstance ;
wndclass.hIcon          = LoadIcon(hInstance,szAppName) ;
wndclass.hCursor        = LoadCursor(NULL,IDC_ARROW) ;
wndclass.hbrBackground  = GetStockObject(WHITE_BRUSH);
wndclass.lpszMenuName   = NULL;
wndclass.lpszClassName  = szAppName ;

RegisterClass(&wndclass) ;

hwnd = CreateWindow(szAppName,
    "TCLCTRL:INTERACTIVE",
    WS_OVERLAPPED|WS_CLIPCHILDREN,
    0,0,0,0,
    NULL,
    NULL,
    hInstance,
    NULL) ;

for(i=0; i<MAX_TEXT; i++)
    varText[i].hwndText = CreateWindow("edit",varText[i].lpszString,
    WS_CHILD | WS_VISIBLE | ES_LEFT|ES_READONLY,
    0,0,0,0,hwnd,i+100, hInstance,NULL);
hwndAbortSource = CreateWindow("button","Abort Batch",
    WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
    0,0,0,0,hwnd,IDM_ABORT_SOURCE, hInstance,NULL);
hwndAbortCommand = CreateWindow("button","Abort Command",
    WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
    0,0,0,0,hwnd,IDM_ABORT_COMMAND, hInstance,NULL);
hwndStop = CreateWindow("button","Stop",
    WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
    0,0,0,0,hwnd,IDM_STOP, hInstance,NULL);
hwndEntry = CreateWindow("edit","STARTUP",
    WS_CHILD | WS_VISIBLE | ES_LEFT,
    0,0,0,0,hwnd,IDM_ENTRY, hInstance,NULL);

```

```

lpfnEntryProc      = MakeProcInstance( (FARPROC)EntryProc,hInstance);
lpfnOldEntryProc  = (FARPROC)
                  GetWindowLong(hwndEntry,GWL_WNDPROC);
SetWindowLong(hwndEntry,GWL_WNDPROC,(LONG) lpfnEntryProc);

while(!SetTimer(hwnd,ID_TIMER1,2000,NULL))
{
    MessageBox(hwnd,"TCLCTRL:SetTimer Failed",
               "TCLCTRL",MB_OK);
    return FALSE;
}

ShowWindow(hwnd,nCmdShow);
UpdateWindow(hwnd);

while(GetMessage(&msg,NULL,0,0))
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
return msg.wParam ;
}

long FAR PASCAL WndProc (HWND hwnd, WORD message,
                        WORD wParam, LONG lParam)
{
    static int          cxChar,cyChar,cxClient,cyClient,FileFlag=0;
    static char         lpszFile[80],lpszTitle[80],lpszFileEntry[80],lpszInput[80];
    HDC                 hdc;
    TEXTMETRIC          tm;
    PAINTSTRUCT         ps;
    static FILE         *Ffile;
    HANDLE              hDrop;
    UINT                i;

    switch(message)
    {
        case WM_CREATE:
            ErrorFile=fopen(GetFileName("SCL_TCLCTRL_LOG"),"wt");
            DragAcceptFiles(hwnd,TRUE);
            if(WinExec(GetFileName("SCL_TCLLINK_EXE"),
                     SW_MINIMIZE) < 32)
            {
                fprintf(ErrorFile,
                       "TCLCTRL:System Fragmented. Terminating Module\n");
                MessageBox(hwnd,

```

```

        "TCLCTRL:System Fragmented. \r\n Terminating
        Module", "TCLCTRL", MB_ICONSTOP|MB_OK);
    PostQuitMessage(0);
}

if((hwndTclLink = FindWindow(NULL, "TCLLINK:INTERLINK"))
    == NULL)
{
    MessageBox(hwnd, "TCLCTRL:Module Interlock unavailable"
        , "TCLCTRL", MB_INFORMATION | MB_OK);
    fprintf(ErrorFile, "TCLCTRL:Module Interlock unavailable");
    PostQuitMessage(0);
}
GlobalAddAtom("INVALID");
fprintf(ErrorFile, "TCLCTRL:Executing Module\n");
return 0;

case WM_SETFOCUS:
    SetFocus(hwndEntry);
    return 0;

case WM_TIMER:
    GetSidereal();
    if(TimerFlag > 0)
    {
        TimerFlag--;
        return 0;
    }
    if(FileFlag == 0 || ModuleWait == 0) return 0;
    if(fgets(lpszFileEntry, 79, Ffile) == NULL )
    {
        FileFlag = 0;
        fclose(Ffile);
        fprintf(ErrorFile, "TCLCTRL:INTERACTIVE\n");
        SetWindowText(hwnd, "TCLCTRL:INTERACTIVE");
        return 0;
    }
    TranslateCommand(hwnd, lpszFileEntry);
    return 0;

case WM_SIZE:
    hdc = GetDC(hwnd);
    SelectObject(hdc, GetStockObject(SYSTEM_FIXED_FONT));
    GetTextMetrics(hdc, &tm);
    cxChar = tm.tmAveCharWidth;
    cyChar = tm.tmHeight + tm.tmExternalLeading;
    ReleaseDC(hwnd, hdc);

```

```

MoveWindow(hwnd,0,0,cxChar*60,cyChar*20,TRUE);
MoveWindow(hwndAbortSource,0,0,cxChar*20-1,
    cyChar*2,TRUE);
MoveWindow(hwndAbortCommand,cxChar*20,0,cxChar*20-1,
    cyChar*2,TRUE);
MoveWindow(hwndStop,cxChar*40,0,cxChar*20-1,
    cyChar*2,TRUE);
MoveWindow(hwndEntry,1,cyChar*2,cxChar*42,cyChar*2,TRUE);

for(i=0; i < MAX_TEXT; i++)
    {
    MoveWindow(varText[i].hwndText,
        cxChar*varText[i].TextPos.x,
        (cyChar+6)*(varText[i].TextPos.y+2),cxChar*20-
        1,cyChar+6,TRUE);
    SetWindowDisplay(i);
    }
return 0;

```

case WM\_DROPFILES:

```

if(FileFlag == 1)fclose(Ffile);
FileFlag = 0;
SetWindowText(hwnd,"TCLCTRL:UPDATING");
hDrop = (HANDLE) wParam;
DragQueryFile((HANDLE) wParam, 0, lpszFile, sizeof(lpszFile));
DragFinish((HANDLE) wParam);
if((Ffile=fopen(lpszFile,"rt"))==NULL)return 0;
FileFlag = 1;
strcpy(lpszTitle,"TCLCTRL:EXCECUTING ");
strcat(lpszTitle,lpszFile);
SetWindowText(hwnd,lpszTitle);
fprintf(ErrorFile,
    "TCLCTRL:EXECUTING DROPFILE < %s > \n",lpszFile);
return 0;

```

case WM\_DESTROY:

```

DragAcceptFiles(hwnd,FALSE);
GlobalDeleteAtom(Aatom);
fprintf(ErrorFile,"TCLCTRL:Terminating Module\n");
fclose(ErrorFile);
PostQuitMessage(0);
return 0;

```

case WM\_COMMAND:

```

switch(wParam)
    {
    case IDM_INTERLINK:

```

```

        GlobalGetAtomName((ATOM) lParam, lpszInput, 80);
        fprintf(ErrorFile,
                "TCLCTRL:INTERLINK Message < %s> \n",
                lpszInput);
        MessageBox(hwnd, lpszInput, "TCLCTRL", MB_OK);
    return 0;

    case IDM_ABORT_SOURCE:
        if(FileFlag == 1) fclose(Ffile);
        FileFlag = 0;
        SetWindowText(hwnd, "TCLCTRL:INTERACTIVE");
        SetFocus(hwndEntry);
        fprintf(ErrorFile,
                "TCLCTRL:ABORT SOURCE\n", lpszInput);
    return 0;

    case IDM_ABORT_COMMAND:
        SetWindowText(hwndEntry, "");
        SetFocus(hwndEntry);
        fprintf(ErrorFile,
                "TCLCTRL:ABORT COMMAND\n",
                lpszInput);
    return 0;

    case IDM_STOP:
        TransmitMessage(
            "AIZAIM0ZBIZBIM0ZCIZCIM0ZDIZDIM0ZFIZFIM0ZZ");
        SetFocus(hwndEntry);
        fprintf(ErrorFile, "TCLCTRL:STOP\n", lpszInput);
    return 0;
    }
    break;
}
return DefWindowProc(hwnd, message, wParam, lParam);
}

long FAR PASCAL EntryProc(HWND hwnd, WORD message,
                          WORD wParam, LONG lParam)
{
    int    iLength;
    char   lpszEntry[80];

    switch (message)
    {
        case WM_KEYDOWN:
            if(wParam == VK_RETURN)
            {

```

```

        iLength=GetWindowTextLength(hwndEntry)+1;
        GetWindowText(hwndEntry,lpszEntry,iLength);
        SetWindowText(hwndEntry,"Processing...");
        TranslateCommand(hwndEntry,lpszEntry);
        fprintf(ErrorFile,"TCLCTRL:INTERACTIVE
                COMMAND < %s> \n",lpszEntry);
        SetWindowText(hwndEntry,"");
        break;
    }
}

return CallWindowProc(lpfnOldEntryProc,hwnd,message,wParam,lParam);
}

unsigned int TranslateCommand(HWND hwnd, char *lpszInput)
{
char    lpszCom1[20],lpszCom2[20],s1[40],lpszOutput[80];
int     d1,d2,d3,d4,d5,d6,iData;

sscanf(lpszInput,"%s %s %d %d %d %d %d",
        &lpszCom1, &lpszCom2,&d1, &d2,&d4, &d5, &d6);
sscanf(lpszInput,"%s %s %s",&lpszCom1, &lpszCom2,&s1);
if(_strncmpi(lpszCom1,"DELAY")==0)
    {
    TimerFlag=atol(lpszCom2)/2;
    return 0;
    }
if(_strncmpi(lpszCom1,"POINT")==0)
    {
    if(_strncmpi(lpszCom2,"REF")==0)
        TransmitMessage(PointRef(d1,d2,d3,d4,d5,d6));
    if(_strncmpi(lpszCom2,"OFF")==0)
        TransmitMessage(PointOffset(d1,d2,d3,d4,d5,d6));
    if(_strncmpi(lpszCom2,"DOME")==0)
        TransmitMessage(PointDome(d1));
    return 0;
    }

if(_strncmpi(lpszCom1,"WAIT")==0)
    {
    ModuleWait=0;
    if(MessageBox(hwnd,
        "TCLCTRL: \r\n SYSTEM SUSPENDED. CONTINUE ?",
        "TCLCTRL MB_INFORMATION|MB_OK")==IDOK)
        ModuleWait=1;

    return 0;
    }
}

```

```

if(_strcmpi(lpszCom1, "FILTER") == 0)
{
    if(strcmpi(lpszCom2, "REF") == 0)
    {
        strcpy(lpszOutput, GetMacro(s1,
            GetFileName("UMF_REFERENCE_FILTER")));
        if(_strcmpi(lpszOutput, "ZZ") == 0) return 0;
        TransmitMessage(lpszOutput);
        strcpy(varText[ID_FILREF].lpszString, s1);
        SetWindowDisplay(ID_FILREF);
        return 0;
    }
    if(_strcmpi(lpszCom2, "OFF") == 0)
    {
        strcpy(lpszOutput,
            GetMacro(s1, GetFileName("UMF_OFFSET_FILTER")));
        if(_strcmpi(lpszOutput, "ZZ") == 0) return 0;
        TransmitMessage(lpszOutput);
        strcpy(varText[ID_FILOFF].lpszString, s1);
        SetWindowDisplay(ID_FILOFF);
        return 0;
    }
    if(_strcmpi(lpszCom2, "BOTH") == 0)
    {
        strcpy(lpszOutput, GetMacro(s1,
            GetFileName("UMF_BOTH_FILTER")));
        if(_strcmpi(lpszOutput, "ZZ") == 0) return 0;
        TransmitMessage(lpszOutput);
        strcpy(varText[ID_FILREF].lpszString, s1);
        strcpy(varText[ID_FILOFF].lpszString, s1);
        SetWindowDisplay(ID_FILREF);
        SetWindowDisplay(ID_FILOFF);
    }
    return 0;
}

if(_strcmpi(lpszCom1, "APERTURE") == 0)
{
    if(_strcmpi(lpszCom2, "REF") == 0)
    {
        strcpy(lpszOutput, GetMacro(s1,
            GetFileName("UMF_REFERENCE_APERTURE")));
        if(_strcmpi(lpszOutput, "ZZ") == 0) return 0;
        TransmitMessage(lpszOutput);
        strcpy(varText[ID_APEREF].lpszString, s1);
        SetWindowDisplay(ID_APEREF);
    }
}

```

```

        return 0;
    }
    if(_strcmpi(lpszCom2, "OFF") == 0)
    {
        strcpy(lpszOutput, GetMacro(s1,
            GetFileName("UMF_OFFSET_APERTURE")));
        if(_strcmpi(lpszOutput, "ZZ") == 0) return 0;
        TransmitMessage(lpszOutput);
        strcpy(varText[ID_APEOFF].lpszString, s1);
        SetWindowDisplay(ID_APEOFF);
        return 0;
    }
    if(_strcmpi(lpszCom2, "BOTH") == 0)
    {
        strcpy(lpszOutput, GetMacro(s1,
            GetFileName("UMF_BOTH_APERTURE")));
        if(_strcmpi(lpszOutput, "ZZ") == 0) return 0;
        TransmitMessage(lpszOutput);
        strcpy(varText[ID_APEREF].lpszString, s1);
        strcpy(varText[ID_APEOFF].lpszString, s1);
        SetWindowDisplay(ID_APEREF);
        SetWindowDisplay(ID_APEOFF);
    }
    return 0;
}

if(_strcmpi(lpszCom1, "CENTRE") == 0)
{
    if(_strcmpi(lpszCom2, "DEC_REF") == 0)
        TransmitMessage("BIZBICZZ");
    if(_strcmpi(lpszCom2, "R.A._OFF") == 0)
        TransmitMessage("CIZCICZZ");
    if(_strcmpi(lpszCom2, "DEC_OFF") == 0)
        TransmitMessage("DIZDICZZ");
    return 0;
}

if(_strcmpi(lpszCom1, "RESET") == 0)
{
    if(_strcmpi(lpszCom2, "R.A._REF") == 0)
        TransmitMessage("AIZAIRZZ");
    if(_strcmpi(lpszCom2, "DEC_REF") == 0)
        TransmitMessage("BIZBIRZZ");
    if(_strcmpi(lpszCom2, "R.A._OFF") == 0)
        TransmitMessage("CIZCIRZZ");
    if(_strcmpi(lpszCom2, "DEC_OFF") == 0)
        TransmitMessage("DIZDIRZZ");
}

```

```

if(_strcmpi(lpszCom2,"DOME")==0)
    TransmitMessage("FIZFIRZZ");
if(_strcmpi(lpszCom2,"ALL")==0)
    TransmitMessage("AIZAIRZBIZBIRZCIZCIRZDI ZFIRZZ");
return 0;
}

```

```

if(_strcmpi(lpszCom1,"CAMERA")==0)
{
    if((iData=atoi(lpszCom2))>0)
    {
        if(iData>0 && iData<5000)
        {
            strcpy(lpszOutput,"HII");
            strcat(lpszOutput,lpszCom2);
            strcat(lpszOutput,"ZZZ");
            TransmitMessage(lpszOutput);
            varText[ID_INT].iData=iData;
            SetWindowDisplay(ID_INT);
        }
        return 0;
    }
    if(_strcmpi(lpszCom2,"REF")==0)
    {
        TransmitMessage("HIC1ZZ");
        strcpy(varText[ID_HEAD].lpszString,lpszCom2);
    }
    if(_strcmpi(lpszCom2,"OFF")==0)
    {
        TransmitMessage("HIC2ZZ");
        strcpy(varText[ID_HEAD].lpszString,lpszCom2);
    }
    if(_strcmpi(lpszCom2,"FINDER_REF")==0)
    {
        TransmitMessage("HIC3ZZ");
        strcpy(varText[ID_HEAD].lpszString,lpszCom2);
    }
    if(_strcmpi(lpszCom2,"FINDER_OFF")==0)
    {
        TransmitMessage("HIC4ZZ");
        strcpy(varText[ID_HEAD].lpszString,lpszCom2);
    }
    SetWindowDisplay(ID_HEAD);
    return 0;
}

```

```

if(_strcmpi(lpszCom1,"SYSTEM")==0)

```

```

{
if(_strcmpi(lpszCom2,"R.A._REF")==0) RaRefEnc = (unsigned int ) d1;
if(_strcmpi(lpszCom2,"R.A._OFF")==0) RaOffEnc = (unsigned int ) d1;
if(_strcmpi(lpszCom2,"DEC_REF")==0) DecRefEnc = (unsigned int )d1;
if(_strcmpi(lpszCom2,"DEC_OFF")==0) DecOffEnc = (unsigned int )d1;
if(_strcmpi(lpszCom2,"COMMAND")==0)
    {
        strcat(s1,"ZZ");
        TransmitMessage(s1);
    }
return 0;
}

if(_strcmpi(lpszCom1,"LED")==0 && (iData=atoi(lpszCom2))<16)
    {
        strcpy(lpszOutput,"HIB");
        strcat(lpszOutput,lpszCom2);
        strcat(lpszOutput,"ZZZ");
        TransmitMessage(lpszOutput);
        varText[ID_LED].iData=iData;
        strcpy(varText[ID_LED].lpszString,"LED:");
        strcat(varText[ID_LED].lpszString,lpszCom2);
        SetWindowDisplay(ID_LED);
    }
return 0;

if(_strcmpi(lpszCom1,"SKY")==0)
    {
        if((iData=atoi(lpszCom2))>0)
            {
                if(iData>0 && iData<200)
                    {
                        strcpy(lpszOutput,"HIN");
                        strcat(lpszOutput,lpszCom2);
                        strcat(lpszOutput,"ZZZ");
                        TransmitMessage(lpszOutput);
                        varText[ID_STEP].iData=iData;
                        SetWindowDisplay(ID_STEP);
                    }
                return 0;
            }
        TransmitMessage("HIOZZ");
        strcpy(varText[ID_PATH].lpszString,"SKY");
        SetWindowDisplay(ID_PATH);
        return 0;
    }
}

```

```

if(_strcmpi(lpszCom1,"STAR")==0)
    {
        TransmitMessage("HIRZZ");
        strcpy(varText[ID_PATH].lpszString,"STAR");
        SetWindowDisplay(ID_PATH);
        return 0;
    }
return 0;
}

unsigned int TransmitMessage(char *lpszMessage)
{
    GlobalDeleteAtom(Aatom);
    Aatom = GlobalAddAtom(lpszMessage);
    SendMessage(hwndTclLink,WM_COMMAND,IDM_INTERLINK,Aatom);
    return 0;
}

char *PointRef(int RaHrs,int RaMin, int RaSec,int DecDeg, int DecMin, i Sec)
{
    static char    lpszOutput[80]="",lpszlabel[20];
    unsigned long Ra,Dec;
    unsigned int  RaTemp,DecTemp;
    unsigned int  RaFlag=0;

    Ra = (unsigned long) RaHrs * 3600L +
        (unsigned long) RaMin * 60L + (unsigned long) RaSec;
    if( Ra > 86400L )return "ZZ";
    Dec = (unsigned long) (DecDeg + 90) * 3600L +
        (unsigned long) DecMin * 60L + (unsigned long) DecSec;
    if( Dec > 648000L )return "ZZ";
    sprintf(varText[ID_RAREF].lpszString," %02d %02d %02d",
        RaHrs,RaMin,RaSec );
    sprintf(varText[ID_DECREf].lpszString,"%+03d %02d %02d",
        DecDeg,DecMin,Dec);
    SetWindowDisplay(ID_RAREF);
    SetWindowDisplay(ID_DECREf);
    if( Ra < 21600L )
        {
            RaFlag = 1;
            Ra += 43200L;
        }
    if( Ra > 64800L )
        {
            RaFlag=1;
            Ra -= 43200L;
        }
}

```

```

RaHrs = (int) (Ra/3600L);
RaTemp = (unsigned int) ( ((float) (Ra%3600L))/ 1.7578125 );
RaTemp = (RaTemp + RaRefEnc) % 2048 ;
RaTemp += (unsigned int) RaHrs * 2048;
if(RaFlag==1)Dec = 1296000L - Dec;
DecDeg= (int) (Dec/3600L);
DecTemp = (unsigned int) ( ((float) (Dec%3600L))/1.7578125 );
DecTemp = (DecTemp + DecRefEnc) % 2048 ;
DecTemp += DecDeg * 2048;
varText[ID_RAREF].iData=Ra;
varText[ID_DECREF].iData=Dec;
strcpy(lpszOutput,"BIZBIPF");
strcat(lpszOutput,GetBinary(DecTemp));
strcat(lpszOutput,"ZBIMGZAIZAIMPF");
strcat(lpszOutput,GetBinary(RaTemp));
strcat(lpszOutput,"ZAIMGZZ");
return lpszOutput;
}

char *PointOffset(int RaDeg,int RaMin, int RaSec,
                  int DecDeg, int DecMin DecSec)
{
static char    lpszOutput[80]="";
unsigned int   Ra,Dec;

Ra = (unsigned int) (21600 + RaOffEnc + RaDeg * 3600 +
                    RaMin * 60 + RaSec );
Dec = (unsigned int) (21600L + DecOffEnc + DecDeg * 3600 +
                    DecMin * 60 + DecSec );
if( Ra < 21600 || Ra > 0x9AB0 )return "ZZ";
if( Dec < 21600 || Dec > 0x9AB0 )return "ZZ";
varText[ID_RAOFF].iData=Ra;
varText[ID_DECOFF].iData=Dec;
sprintf(varText[ID_RAOFF].lpszString," %02d %02d %02d",
        RaDeg,RaMin,RaSec);
sprintf(varText[ID_DECOFF].lpszString,"%+03d %02d %02d",
        DecDeg,DecMin,DecSec)
SetWindowDisplay(ID_RAOFF);
SetWindowDisplay(ID_DECOFF);
strcpy(lpszOutput,"CIZCIPF");
strcat(lpszOutput,GetBinary(Ra));
strcat(lpszOutput,"ZDIZDIPF");
strcat(lpszOutput,GetBinary(Dec));
strcat(lpszOutput,"ZZ");
return lpszOutput;
}

```

```
char *PointDome(int Azimuth)
```

```
{  
static char    lpszOutput[80];  
  
Azimuth %= 360;  
varText[ID_DOME].iData = Azimuth;  
SetWindowDisplay(ID_DOME);  
strcpy(lpszOutput, "FIZFIPF");  
strcat(lpszOutput, GetBinary(Azimuth*4));  
strcat(lpszOutput, "ZFIMGZZ");  
return lpszOutput;  
}
```

```
char*  GetBinary(unsigned int iPos)
```

```
{  
int    Bit;  
static char lpszBinary[20];  
  
strcpy(lpszBinary, "0000000000000000");  
for(Bit=0; Bit <= 15; Bit++) if( (iPos & (1 << Bit) ) != 0)  
    lpszBinary[15-Bit] = '1';  
return lpszBinary;  
}
```

```
char*  GetMacro(char* lpszEntry, char * lpszFile)
```

```
{  
FILE    *file;  
static char    lpszInput[80], lpszFileLabel[80], lpszFileData[80];  
  
if((file = fopen(lpszFile, "rt")) == NULL) return "ZZ";  
while(1)  
{  
    if(fgets(lpszInput, 80, file) == NULL) break;  
    if(sscanf(lpszInput, "%s %s", &lpszFileLabel, &lpszFileData) == 2)  
    {  
        if(_strcmpi(lpszEntry, lpszFileLabel) == 0)  
        {  
            fclose(file);  
            return lpszFileData;  
        }  
    }  
}  
fclose(file);  
return "ZZ";  
}
```

```
UINT  SetWindowDisplay(UINT i)
```

```

{
int    iA,iB,iC;
char   lpszText[20],lpszNumber[10];

_itoa(varText[i].iData,lpszNumber,10);
switch(varText[i].iStatus)
    {
    case 0:
        strcpy(lpszText,"");
        break;
    case 1:
        strcpy(lpszText,varText[i].lpszString);
        break;
    case 2:
        strcpy(lpszText,lpszNumber);
        break;
    case 3:
        strcpy(lpszText,varText[i].lpszString);
        strcat(lpszText," [");
        strcat(lpszText,lpszNumber);
        strcat(lpszText,"]");
        break;
    case 4:
        iA=varText[i].iData/3600;
        iB=(varText[i].iData - iA*3600)/60;
        iC=varText[i].iData - iA*3600 - iB*60;
        sscanf(lpszText,"%2d %2d %2d",iA,iB,iC);
        break;
    case 5:
        iA=varText[i].iData/3600;
        iB=(abs(varText[i].iData) - abs(iA)*3600)/60;
        iC=abs(varText[i].iData) - abs(iA)*3600 - iB*60;
        sscanf(lpszText,"% +2d %2d %2d",iA,iB,iC);
        break;
    }
SetWindowText(varText[i].hwndText,lpszText);
return 0;
}

UINT GetSidereal(void)
{
varText[ID_SIDEREAL].iData=time(NULL);
SetWindowDisplay(ID_SIDEREAL);
return 0;
}

```

## D.5 : TCLCTRL : Resource script

Filename:	TCLCTRL.C
Application:	Resource script for C700 complier

```
#include <windows.h>
#include "TCLCTRL.h"
TCLCTRL ICON TCLCTRL.ico

TCLCTRL MENU
    BEGIN
        POPUP "&File"
            BEGIN
                MENUITEM "About", IDM_ABOUT
                MENUITEM SEPARATOR
                MENUITEM "Exit", IDM_EXIT
            END
        END
    END
END
```

## **D.6 : TCLLINK : Make file**

Filename:	TCLLINK.MAK
Application:	Make file for C700 complier

TCLLINK.exe : TCLLINK.obj TCLLINK.def TCLLINK.res

link TCLLINK, /align:16, NUL, /nod slibcew libw shell,  
TCLLINK rc TCLLINK.res

TCLLINK.obj : TCLLINK.c TCLLINK.h TCLLINK.ico  
cl -c -Gsw -Ow -W2 -Zp TCLLINK.c

TCLLINK.res : TCLLINK.rc TCLLINK.h TCLLINK.ico  
rc -r TCLLINK.rc

## D.7 : TCLLINK : Definition file

Filename:	TCLLINK.DEF
Application:	Defininition file for C700 complier

```
;=====
; TCLLINK.DEF MODULE DEFINITION FILE
;=====
NAME          TCLLINK
DESCRIPTION   'TPT TCLLINK COMMUNICATIONS CONTROLLER'
EXETYPE      WINDOWS
STUB         'WINSTUB.EXE'
CODE         PRELOAD MOVEABLE DISCARDABLE
DATA        PRELOAD MOVEABLE MULTIPLE
HEAPSIZE     1024
STACKSIZE   8192
EXPORTS      WndProc
```

## **D.8 : TCLLINK : Include file**

Filename:	TCLLINK.H
Application:	Include file for C700 compier

```
//TCLLINK include file
#ifndef H_TCLLINK
#define H_TCLLINK

#define IDM_EXIT          1
#define IDM_ABOUT        2
#define ID_TIMER1         3
#define ROUTE_NONE        0
#define ROUTE_NETWORK     1
#define ROUTE_CUBE        2
#define ROUTE_ERROR       3
#define TX_BUFFER         128
#define RX_BUFFER         128
#define NET_MCR           0x2EC
#define NET_MSR           0x2EE

#endif
```

## D.9 : TCLLINK : Source file

Filename:	TCLLINK.C
Application:	Source file for C700 complier

```
//TCLLINK COMMUNICATIONS MODULE
#include < windows.h >
#include < shellapi.h >
#include < stdio.h >
#include < string.h >
#include "system.h"
#include "TCLLINK.H"
#include "system.h"
#include "system.c"

long FAR PASCAL WndProc (HWND, WORD, WORD, LONG);

static int idComNet,idComCube;
static FILE *ErrorFile;

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpszCmd int nCmdShow)
{
static char szAppName[] = "TCLLINK";
HWND hwnd;
MSG msg;
WNDCLASS wndclass;

if(hPrevInstance)return FALSE;
wndclass.style = CS_HREDRAW | CS_VREDRAW;
wndclass.lpfWndProc = WndProc ;
wndclass.cbClsExtra = 0;
wndclass.cbWndExtra = 0;
wndclass.hInstance = hInstance ;
wndclass.hIcon = LoadIcon(hInstance,szAppName) ;
wndclass.hCursor = LoadCursor(NULL,IDC_ARROW) ;
wndclass.hbrBackground = GetStockObject(WHITE_BRUSH) ;
wndclass.lpszMenuName = NULL;
wndclass.lpszClassName = szAppName ;

RegisterClass(&wndclass) ;

hwnd = CreateWindow(szAppName,
                    "TCLLINK:INTERLINK",
                    WS_OVERLAPPEDWINDOW + WS_MINIMIZE, CW_USEDEFAULT,
                    CW_USEDEFAULT,
```

```

    CW_USEDEFAULT,
    CW_USEDEFAULT,
    NULL,
    NULL,
    hInstance,
    NULL) ;

while(!SetTimer(hwnd,ID_TIMER1,700,NULL))
{
    MessageBox(hwnd,"TCLLINK:SetTimer Failed","TCLLINK",MB_OK);
    return FALSE;
}

ShowWindow(hwnd, nCmdShow) ;
UpdateWindow(hwnd) ;

while(GetMessage(&msg,NULL,0,0))
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
return msg.wParam ;
}

long FAR PASCAL WndProc (HWND hwnd, WORD message,
                        WORD wParam, LONG lParam)
{
    unsigned int      TransmitMessage(HWND,char *,unsigned int);
    unsigned int      Breakout(HWND, char* ,int);
    static DCB        dcbnet,dcbcube;
    WORD              cFiles;
    static char       lpszFile[80],lpszTitle[80],c,lpszOutput[80],lpszBuffer[10],cChar;
    static char       lpszMessage[130],lpszInput[130];
    HANDLE            hDrop;
    static unsigned int FileFlag=FALSE,InterlinkFlag=FALSE;
    static FILE        *file;
    static unsigned int FileCursor,CursorA,CursorB;
    static ATOM        Aatom;
    static HWND        hwndTclCtrl;
    COMSTAT FAR*      ComStat;

    switch(message)
    {
        case WM_SETFOCUS:
            SetFocus(NULL);
            return 0;
    }
}

```

```

case WM_CREATE :
    DragAcceptFiles(hwnd,TRUE);
    if((ErrorFile=fopen(GetFileName("SCL_TCLLINK_LOG"),"wt"))
        ==NULL)
        BreakOut(hwnd,"TCLLINK:Log file failed on open",1);
    GlobalAddAtom("INVALID");
    if((idComCube=
        OpenComm("COM2",TX_BUFFER, RX_BUFFER)) < 0)
        BreakOut(hwnd,"OpenComm:COM2 failed",1);
    if(BuildCommDCB("COM2:9600,n,8,1", &dcbnet)<0)
        BreakOut(hwnd,"BuildCommDCB:COM2 failed",1);
    if(SetCommState(&dcbnet) < 0)
        BreakOut(hwnd,"SetCommState:COM2 failed",1);
    if((idComNet=
        OpenComm("COM4",TX_BUFFER, RX_BUFFER)) < 0)
        BreakOut(hwnd,"OpenComm:COM4 failed",1);
    if(BuildCommDCB("COM4:1200,n,8,1", &dcbcube)<0)
        BreakOut(hwnd,"BuildCommDCB:COM4 failed",1);
    if(SetCommState(&dcbcube) < 0)
        BreakOut(hwnd,"SetCommState:COM4 failed",1);
    MessageBeep(MB_OK);
    FlushComm(idComNet,0);
    FlushComm(idComCube,0);
    SetCommEventMask(idComNet,0);
    SetCommEventMask(idComCube,0);
    fprintf(ErrorFile,"TCLLINK:Module Started\r\n");
    return BreakOut(hwnd,"OK",0);

```

```

case WM_DESTROY:
    DragAcceptFiles(hwnd,FALSE);
    FlushComm(idComNet,0);
    FlushComm(idComCube,0);
    CloseComm(idComNet);
    CloseComm(idComCube);
    MessageBeep(MB_OK);
    fprintf(ErrorFile,"TCLLINK:Terminating Module\n");
    fclose(ErrorFile);
    PostQuitMessage(0);
    return 0;

```

```

case WM_TIMER:
    while(InterlinkFlag!=FALSE)
    {
        lpszMessage[CursorA++] = lpszInput[CursorB++];
        if(lpszMessage[CursorA-1] == 'Z')
        {
            if(CursorA < 3 || CursorB > 120)

```

```

        {
            InterlinkFlag=FALSE;
            SetWindowText
                (hwnd,"TCLLINK:INTERLINK");
            return 0;
        }
        lpszMessage[CursorA]=0;
        strcpy(lpszTitle,"TCLLINK:INTERLINK [");
        strcat(lpszTitle,lpszMessage);
        strcat(lpszTitle,"]");
        SetWindowText(hwnd,lpszTitle);
        TransmitMessage(hwnd,lpszMessage, CursorA-1);
        CursorA =0;
        return 0;
    }
}
if(FileFlag==FALSE)return 0;
for(FileCursor=0; FileCursor < TX_BUFFER;FileCursor++)
{
    if((lpszMessage[FileCursor]=fgetc(file))==EOF)
    {
        FileFlag=FALSE;
        fclose(file);
        SetWindowText(hwnd,"TCLLINK:INTERLINK");
        fprintf(ErrorFile,"
            TCLLINK:INTERLINK RESUMED");
        return 0;
    }
    if(lpszMessage[FileCursor]=='Z')break;
}
if(FileCursor > TX_BUFFER-2 || FileCursor < 2)return 0;
TransmitMessage(hwnd,lpszMessage,FileCursor);
return 0;

case WM_COMMAND:
    MessageBeep(-1);
    switch (wParam)
    {
        case IDM_INTERLINK:
            GlobalGetAtomName((ATOM) lParam,lpszInput,80);
            CursorA=0;
            CursorB=0;
            InterlinkFlag=TRUE;
            FileFlag=FALSE;
            fprintf(ErrorFile,
                "TCLLINK:INTERLINK:MESSAGE < %s > \n",
                    lpszInput);

```

```

        return 0;
    }
    return 0;

case WM_DROPFILES:
    if(FileFlag==TRUE)fclose(file);
    FileFlag==FALSE;
    FlushComm(idComNet,0);
    FlushComm(idComCube,0);
    SetWindowText(hwnd,"TCLLINK:UPDATE");
    hDrop = (HANDLE) wParam;
    DragQueryFile((HANDLE) wParam, 0, lpszFile, sizeof(lpszFile));
    DragFinish((HANDLE) wParam);
    if((file=fopen(lpszFile,"rt"))==NULL)return 0;
    FileFlag = TRUE;
    InterlinkFlag=FALSE;
    strcpy(lpszTitle,"TCLLINK:");
    strcat(lpszTitle,lpszFile);
    SetWindowText(hwnd,lpszTitle);
    fprintf(ErrorFile,"TCLLINK:DROPFILE:FILE < %s > \n",lpszFile);
    return 0;
}
return DefWindowProc(hwnd, message, wParam, lParam);
}

int BreakOut(HWND hwnd, char* lpszMessage,int iData)
{
if(iData==0)return 0;
MessageBox(hwnd,lpszMessage,"TCLLINK",MB_OK);
fprintf(ErrorFile,"TCLLINK:ERROR: < %s > \n",lpszMessage);
return FALSE;
}

unsigned int TransmitMessage(HWND hwnd, char *lpszMessage,
    unsigned int MessageLength)
{
int TxLength,idCom;

switch(lpszMessage[0])
{
case 'H':
    idCom=idComCube;
    lpszMessage +=2;
    lpszMessage[MessageLength-2]=0x0d;
    TxLength=MessageLength-1;
    break;
case 'A':

```

```

case 'B':
case 'C':
case 'D':
case 'F':
    idCom=idComNet;
    lpszMessage[MessageLength+1]=0;
    TxLength=MessageLength+1;
    break;
default:
    BreakOut(hwnd,"TCLINK:UNKNOWN TARGET",1);
    return 0;
}
ClearCommBreak(idCom);
FlushComm(idCom,0);
if(WriteComm(idCom,lpszMessage,TxLength) < 0)
    BreakOut(hwnd,"TCLINK:WriteCom Failed",1);
return 0;
}

```

## **D.10 : TCLLINK : Resource script**

Filename:	TCLLINK.RC
Application:	Resource script for C700 complier

```
#include      < windows.h >
#include      < shellapi.h >
#include      "TCLLINK.H"

TCLLINK ICON  TCLLINK.ICO
```

## **D.11 : TCLQUICK : Make file**

Filename:	TCLQUICK.MAK
Application:	Make file for C700 complier

telquick.exe : telquick.obj telquick.def telquick.res

link telquick, /align:16, NUL, /nod slibcew libw, telquick rc telquick.res

telquick.obj : telquick.c telquick.h telquick.ico  
cl -c -Gsw -Ow -W2 -Zi telquick.c

telquick.res : telquick.rc telquick.h telquick.ico  
rc -r telquick.rc

## D.12 : TCLQUICK : Definition file

Filename:	TCLQUICK.DEF
Application:	Definition file for C700 complier

```
=====
; tclquick.def module definition file
=====
NAME          TCLQUICK
DESCRIPTION   'TPT TCLQUICK COMMAND MODULE'
EXETYPE       WINDOWS
STUB          'WINSTUB.EXE'
CODE          PRELOAD MOVEABLE DISCARDABLE
DATA          PRELOAD MOVEABLE MULTIPLE
HEAPSIZE      1024
STACKSIZE     8192
EXPORTS       WndProc
```

### **D.13 : TCLQUICK : Include file**

Filename:	TCLQUICK.H
Application:	Include file for C700 complier

```
//TCLQUICK include file
#ifndef H_TCLQUICK
#define H_TCLQUICK

#define MAX_QUICK_ENTRY    15

#endif
```

## D.14 : TCLQUICK : Source file

Filename:	TCLQUICK.C
Application:	Source file for C700 complier

```
//Telescope Quick Control Module
#include    < windows.h >
#include    < stdio.h >
#include    < string.h >
#include    "tclquick.h"
#include    "global.h"
#include    "system.h"
#include    "system.c"

long FAR PASCAL WndProc (HWND, WORD, WORD, LONG);

HWND      hwndButton[MAX_QUICK_ENTRY];
static FILE *ErrorFile;

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpszCmdParam, int nCmdShow)
{
static char  szAppName[] = "tclquick" ;
HWND        hwnd;
MSG         msg;
WNDCLASS    wndclass;
int         i;

if(!hPrevInstance)
    {
    wndclass.style          = CS_HREDRAW | CS_VREDRAW;
    wndclass.lpfWndProc     = WndProc ;
    wndclass.cbClsExtra    = 0;
    wndclass.cbWndExtra    = 0;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon         = LoadIcon(hInstance,szAppName) ;
    wndclass.hCursor       = LoadCursor(NULL, IDC_ARROW) ;
    wndclass.hbrBackground = GetStockObject(WHITE_BRUSH) ;
    wndclass.lpszMenuName  = NULL;
    wndclass.lpszClassName = szAppName ;
    RegisterClass(&wndclass) ;
    }

hwnd = CreateWindow(szAppName,"TCLQUICK",
                    WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
                    0,0,0,0,NULL,NULL,hInstance,NULL) ;
```

```

for(i=0; i< MAX_QUICK_ENTRY; i++)
{
    hwndButton[i] = CreateWindow("button","UNDEFINED",
        WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
        0,0,0,0,hwnd,i,hInstance,NULL);
}

ShowWindow(hwnd,nCmdShow);
UpdateWindow(hwnd);

while(GetMessage(&msg,NULL,0,0))
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}

return msg.wParam ;
}

long FAR PASCAL WndProc (HWND hwnd, WORD message,
                        WORD wParam, LONG lParam)
{
    static int          cxChar,cyChar,cxClient,cyClient,i;
    static char         lpszString[MAX_QUICK_ENTRY][40],
                        lpszEntry[MAX_QUICK_ENTRY][40];
    char                lpszBuffer[120];
    HDC                 hdc;
    TEXTMETRIC          tm;
    PAINTSTRUCT         ps;
    FILE                *file;
    static ATOM         Aatom;
    static HWND         hwndTclLink;

    switch(message)
    {
        case WM_CREATE:
            ErrorFile=fopen(GetFileName("SCL_TCLQUICK_LOG"),"wt");
            if(WinExec(GetFileName("SCL_TCILINK_EXE"),
                SW_MINIMIZE) < 32)
            {
                strcpy(lpszBuffer,
                    "TCLQUICK:System Fragmented. Terminating Module");
                MessageBox(hwnd,lpszBuffer,"TCLQUICK",
                    MB_ICONSTOP|MB_OK);
                fprintf(ErrorFile,"%s\n",lpszBuffer);
                PostQuitMessage(0);
            }
    }
}

```

```

    }

if((hwndTclLink = FindWindow(NULL,"TCLLINK:INTERLINK"))
    ==NULL)
    {
    strcpy(lpszBuffer,
        "TCLQUICK:Module Interlock unavailable");
    MessageBox(hwnd,lpszBuffer,"TCLQUICK",
        MB_ICONSTOP|MB_OK);
    fprintf(ErrorFile,"%s\n",lpszBuffer);
    PostQuitMessage(0);
    }

GlobalAddAtom("INVALID");
if((file=fopen(GetFileName("SCL_TCLQUICK_CONFIG"),"rt"))
    ==NULL)
    {
    strcpy(lpszBuffer,"TCLQUICK:Invalid Initialisation file");
    MessageBox(hwnd,
        "TCLQUICK:INVALID \r\n INITIALISATION
        FILE","TCLQUICK"ICONSTOP | MB_OK);
    fprintf(ErrorFile,"%s\n",lpszBuffer);
    PostQuitMessage(0);
    return FALSE;
    }

for(i=0; i<MAX_QUICK_ENTRY; i++)
    {
    if(fgets(lpszBuffer,120,file)==NULL)break;
    sscanf(lpszBuffer,"%s %s",lpszEntry[i],lpszString[i]);
    }

fclose(file);
fprintf(ErrorFile,"TCLQUICK:Executing Module\n");
return 0;

case WM_SIZE :
if(wParam==SIZE_MINIMIZED)break;
if(wParam==SIZE_MAXIMIZED)return 0;
hdc = GetDC(hwnd);
SelectObject(hdc,GetStockObject(SYSTEM_FIXED_FONT));
GetTextMetrics(hdc,&tm);
cxChar = tm.tmAveCharWidth;
cyChar = tm.tmHeight + tm.tmExternalLeading;
ReleaseDC(hwnd,hdc);
SetWindowPos(hwnd,HWND_TOP,0,0,cxChar*20,cyChar*2*
    MAX_QUICK_ENTRY+47,SHOW_ACTIVE);

```

```

for(i=0; i<MAX_QUICK_ENTRY; i++)
    {
        SetWindowText(hwndButton[i],lpszEntry[i]);
        MoveWindow(hwndButton[i],0,i*cyChar*2,cxChar*20,
            cyChar*2-1,TRUE);
    }
return 0;

case WM_DESTROY:
    fprintf(ErrorFile,"TCLQUICK:Terminating Module\n");
    fclose(ErrorFile);
    GlobalDeleteAtom(Aatom);
    PostQuitMessage(0);
    return 0;

case WM_COMMAND:
    if( (wParam >= 0) && (wParam < MAX_QUICK_ENTRY) )
        {
            GlobalDeleteAtom(Aatom);
            Aatom = GlobalAddAtom(lpszString[wParam]);
            fprintf(ErrorFile,"TCLQUICK:Message Sent < %s> \n",
                lpszString[wParam]);
            SendMessage(hwndTclLink,WM_COMMAND,
                IDM_INTERLINK,Aatom);
            return 0;
        }
    }
return DefWindowProc(hwnd, message, wParam, lParam);
}

```

## **D.15 : TCLQUICK : Resource script**

Filename:	TCLQUICK.RC
Application:	Resource script for C700 complier

#include < windows.h >

#include "tclquick.h"

TCLQUICK ICON TCLQUICK.ico

## **D.16 : TCLMAN : Make file**

Filename:	TCLMAN.MAK
Application	Make file for C700 complier

tclman.exe : tclman.obj tclman.def tclman.res

link tclman, /align:16, NUL, /nod slibcew libw, tclman rc tclman.res

tclman.obj : tclman.c tclman.h tclman.ico  
cl -c -Gsw -Ow -W2 -Zi tclman.c

tclman.res : tclman.rc tclman.h tclman.ico  
rc -r tclman.rc

## D.17 : TCLMAN : Definition file

Filename:	TCLMAN.DEF
Application:	definition file for C700 complier

```
;=====
; tclman.def module definition file
;=====
```

```
NAME          TCLMAN
DESCRIPTION    'TPT TCLMAN COMMAND MODULE'
EXETYPE        WINDOWS
STUB           'WINSTUB.EXE'
CODE           PRELOAD MOVEABLE DISCARDABLE
DATA           PRELOAD MOVEABLE MULTIPLE
HEAPSIZE       1024
STACKSIZE      8192
EXPORTS        WndProc
```

## **D.18 : TCLMAN : Include file**

Filename:	TCLMAN.H
Application:	Include file for C700 compier

```
//TCLMAN include file
#ifndef H_TCLMAN
#define H_TCLMAN

#define TUBE_REFERENCE      0
#define TUBE_OFFSET        1
#define TUBE_DOME           2
#define IDM_REF             3
#define IDM_OFFSET         4
#define IDM_DOME           5
#define IDM_ABOUT          6
#define IDM_EXIT            7
#define IDM_SLEW           8
#define IDM_FINE           9
#define IDM_HELP_CONTENTS 10
#define IDM_HELP_TCLMAN    11
#define IDM_HELP_EXIT      12

#endif
```

## D.19 : TCLMAN : Source file

Filename:	TCLMAN.C
Application:	source file for C700 complier

```
//Telescope Point Control Module
#include < windows.h >
#include < stdio.h >
#include < string.h >
#include "tclman.h"
#include "global.h"
#include "system.h"
#include "system.c"

long FAR PASCAL WndProc (HWND, WORD, WORD, LONG);
HWND hwndButton[9];
static char lpszAttr[19][9] =
    {
        "", "-DEC", "",
        "-R.A.", "STOP", "+R.A.",
        "", "+DEC", "",
        "CCW 1", "STOP", "CW 1",
        "CCW 2", "STOP", "CW 2",
        "CCW 3", "STOP", "CW 3"
    };

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpszCmdParam, int nCmdShow)
{
    static char szAppName[] = "TCLMAN" ;
    HWND hwnd;
    MSG msg;
    WNDCLASS wndclass;
    UINT i;

    if(!hPrevInstance)
    {
        wndclass.style = CS_HREDRAW | CS_VREDRAW;
        wndclass.lpfWndProc = WndProc ;
        wndclass.cbClsExtra = 0;
        wndclass.cbWndExtra = 0;
        wndclass.hInstance = hInstance ;
        wndclass.hIcon = LoadIcon(hInstance, szAppName);
        wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
        wndclass.hbrBackground = GetStockObject(BLACK_BRUSH) ;
        wndclass.lpszMenuName = szAppName;
    }
}
```

```

    wndclass.lpszClassName    = szAppName ;

    RegisterClass(&wndclass) ;

}

hwnd = CreateWindow(szAppName,"TCLMAN:",
    WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
    0,0,0,0,NULL,NULL,hInstance,NULL) ;

for(i=0; i<9; i++)hwndButton[i] =
    CreateWindow("button",lpszAttr[i],
        WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
        0,0,0,0,hwnd,i+20,hInstance,NULL);

ShowWindow(hwnd,nCmdShow);
UpdateWindow(hwnd);

while(GetMessage(&msg,NULL,0,0))
    {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
    }

return msg.wParam ;
}

long FAR PASCAL WndProc (HWND hwnd, WORD message,
    WORD wParam, LONG lParam)
{
char *GetCommand(unsigned int,unsigned int, unsigned int);
static HWND      hwndTclLink;
HDC             hdc;
TEXTMETRIC      tm;
PAINTSTRUCT     ps;
static unsigned int  iApply=0,iX,iY,i,z=0,iSpeed=0;
HMENU           hMenu;
ATOM            Aatom;
static char      lpszMessage[80];

switch(message)
    {
    case  WM_CREATE:
        CheckMenuItem(hMenu,IDM_REF,MF_CHECKED);
        CheckMenuItem(hMenu,IDM_SLEW,MF_CHECKED);
        SetWindowText(hwnd,"TCLMAN : Reference");
        if(WinExec(GetFileName("SCL_TCLLINK_EXE"),

```

```

        SW_MINIMIZE) < 32)
    {
        MessageBox(hwnd,
        "TCLMAN: System Fragmented. Terminating Module",
        "TCLQUICK", MB_ICONSTOP | MB_OK);
        PostQuitMessage(0);
    }

if((hwndTclLink = FindWindow(NULL, "TCLLINK: INTERLINK"))
    == NULL)
    {
        MessageBox(hwnd,
        "TCLMAN: Module Interlock unavailable",
        "TCLMAN", MB_ICO B_OK);
        PostQuitMessage(0);
    }

GlobalAddAtom("INVALID");
return 0;

case WM_SIZE :
if(wParam == SIZE_MINIMIZED) break;
if(wParam == SIZE_MAXIMIZED) return 0;
hdc = GetDC(hwnd);
SelectObject(hdc, GetStockObject(SYSTEM_FIXED_FONT));
GetTextMetrics(hdc, &tm);
iX = 10*tm.tmAveCharWidth;
iY = 2*(tm.tmHeight + tm.tmExternalLeading);
SetWindowPos(hwnd, HWND_TOP, 0, 0, iX*3, iY*3+45,
    SWP_NOMOVE);
for(i=0; i<9; i++) MoveWindow(hwndButton[i], (2-i%3)*iX,
    (2-i/3)*iY, iX, iY, TRUE, ReleaseDC(hwnd, hdc);
return 0;

case WM_DESTROY:
WinHelp(hwnd, "tclman.hlp", HELP_QUIT, NULL);
GlobalDeleteAtom(Aatom);
PostQuitMessage(0);
return 0;

case WM_COMMAND:
hMenu = GetMenu(hwnd);
switch(wParam)
    {
        case IDM_HELP_CONTENTS:
        WinHelp(hwnd, "tcl.hlp", HELP_CONTENTS, 0L);
        return 0;
    }

```

```

case IDM_HELP_TCLMAN:
WinHelp(hwnd, "tcl.hlp", HELP_CONTEXT, 0x0001);
return 0;

case IDM_HELP_EXIT:
WinHelp(hwnd, "tcl.hlp", HELP_QUIT, NULL);
return 0;

case IDM_REF:
CheckMenuItem(hMenu, IDM_REF, MF_CHECKED);
CheckMenuItem(hMenu, IDM_OFFSET, MF_UNCHECKED);
CheckMenuItem(hMenu, IDM_DOME, MF_UNCHECKED);
iApply = 0;
z+ = 1;
for(i=0; i<9; i++)
    SetWindowText(hwndButton[i], lpszAttr[i]);
SetWindowText(hwnd, "TCLMAN:REFERENCE");
return 0;

case IDM_OFFSET:
CheckMenuItem(hMenu, IDM_REF, MF_UNCHECKED);
CheckMenuItem(hMenu, IDM_OFFSET, MF_CHECKED);
CheckMenuItem(hMenu, IDM_DOME, MF_UNCHECKED);
iApply = 1;
SetWindowText(hwnd, "TCLMAN:OFFSET");
for(i=0; i<9; i++)
SetWindowText(hwndButton[i], lpszAttr[i]);
return 0;

case IDM_DOME:
CheckMenuItem(hMenu, IDM_REF, MF_UNCHECKED);
CheckMenuItem(hMenu, IDM_OFFSET, MF_UNCHECKED);
CheckMenuItem(hMenu, IDM_DOME, MF_CHECKED);
iApply = 2;
SetWindowText(hwnd, "TCLMAN:DOME");
for(i=0; i<9; i++)
    SetWindowText(hwndButton[i], lpszAttr[i+9]);
return 0;

case IDM_SLEW:
CheckMenuItem(hMenu, IDM_SLEW, MF_CHECKED);
CheckMenuItem(hMenu, IDM_FINE, MF_UNCHECKED);
iSpeed = 0;
return 0;

case IDM_FINE:

```

```

    CheckMenuItem(hMenu, IDM_SLEW, MF_UNCHECKED);
    CheckMenuItem(hMenu, IDM_FINE, MF_CHECKED);
    iSpeed = 1;
    return 0;

    case  IDM_ABOUT:
    MessageBox(hwnd, "TCL Manual Interactive mode",
        "TPT", MB_ICONINFORMATION );
    return 0;

    case  IDM_EXIT:
    PostQuitMessage(0);
    return 0;
    }

if(wParam >=20 && wParam <=28)
    {
    GlobalDeleteAtom(Aatom);
    Aatom = GlobalAddAtom(
        GetCommand(wParam-20, iApply, iSpeed));
    SendMessage(hwndTclLink, WM_COMMAND,
        IDM_INTERLINK, Aatom);
    }
}

return DefWindowProc(hwnd, message, wParam, lParam);
}

char *GetCommand(unsigned int iDir, unsigned int iApply, unsigned int iSpeed)
{
switch(iDir + iApply*20 + iSpeed*10)
    {
    case 0: return "BIZBIM9ZAIZAIM9ZZZ";
    case 1: return "BIZBIM9ZZZ";
    case 2: return "BIZBIM9ZAIZAIM8ZZZ";
    case 3: return "AIZAIM9ZZZ";
    case 4: return "BIZBIM0ZAIZAIM0ZZZ";
    case 5: return "AIZAIM8ZZZ";
    case 6: return "BIZBIM8ZAIZAIM9ZZZ";
    case 7: return "BIZBIM8ZZZ";
    case 8: return "BIZBIM8ZAIZAIM8ZZZ";
    case 10: return "BIZBIM2ZAIZAIM2ZZZ";
    case 11: return "BIZBIM2ZZZ";
    case 12: return "BIZBIM2ZAIZAIM1ZZZ";
    case 13: return "AIZAIM2ZZZ";
    case 14: return "BIZBIM0ZAIZAIM0ZZZ";
    case 15: return "AIZAIM1ZZZ";
    case 16: return "BIZBIM1ZAIZAIM2ZZZ";
    }
}

```

```

case 17: return "BIZBIM1ZZZ";
case 18: return "BIZBIM1ZAIZAIM1ZZZ";
case 20: return "DIZDIM2ZCIZCIM2ZZZ";
case 21: return "DIZDIM2ZZZ";
case 22: return "DIZDIM2ZCIZCIM1ZZZ";
case 23: return "CIZCIM2ZZZ";
case 24: return "DIZDIM0ZCIZCIM0ZZZ";
case 25: return "CIZCIM1ZZZ";
case 26: return "DIZDIM1ZCIZCIM2ZZZ";
case 27: return "DIZDIM1ZZZ";
case 28: return "DIZDIM1ZCIZCIM1ZZZ";
case 30: return "DIZDIM2ZCIZCIM2ZZZ";
case 31: return "DIZDIM2ZZZ";
case 32: return "DIZDIM2ZCIZCIM1ZZZ";
case 33: return "CIZCIM2ZZZ";
case 34: return "DIZDIM0ZCIZCIM0ZZZ";
case 35: return "CIZCIM1ZZZ";
case 36: return "DIZDIM1ZCIZCIM2ZZZ";
case 37: return "DIZDIM1ZZZ";
case 38: return "DIZDIM1ZCIZCIM1ZZZ";
case 40: return "FIZFIM1ZZZ";
case 41: return "FIZFIM0ZZZ";
case 42: return "FIZFIM4ZZZ";
case 43: return "FIZFIM2ZZZ";
case 44: return "FIZFIM0ZZZ";
case 45: return "FIZFIM5ZZZ";
case 46: return "FIZFIM3ZZZ";
case 47: return "FIZFIM0ZZZ";
case 48: return "FIZFIM6ZZZ";
case 50: return "FIZFIM1ZZZ";
case 51: return "FIZFIM0ZZZ";
case 52: return "FIZFIM4ZZZ";
case 53: return "FIZFIM2ZZZ";
case 54: return "FIZFIM0ZZZ";
case 55: return "FIZFIM5ZZZ";
case 56: return "FIZFIM3ZZZ";
case 57: return "FIZFIM0ZZZ";
case 58: return "FIZFIM6ZZZ";
}
return "AIZZZ";
}

```

## D.20 : TCLMAN : Resource script

Filename:	TCLMAN.RC
Application:	Resource script for C700 compiler

```
#include <windows.h>
#include "TCLMAN.H"

TCLMANICON    TCLMAN.ICO

TCLMANACCELERATORS
    BEGIN
        VK_F1, IDM_HELP_CONTENTS, VIRTKEY
    END

TCLMANMENU
    BEGIN
        POPUP "&File"
            BEGIN
                MENUITEM "&Reference"           ",IDM_REF
                MENUITEM "&Offset"             ",IDM_OFFSET
                MENUITEM "&Dome"               ",IDM_DOME
                MENUITEM "&Slew"               ",IDM_SLEW
                MENUITEM "&Fine"              ",IDM_FINE
                MENUITEM "&About TCLMAN"      "IDM_ABOUT
                MENUITEM SEPARATOR
                MENUITEM "E&xit TCLMAN"      ",IDM_EXIT
            END
        POPUP "&Help"
            BEGIN
                MENUITEM "&Contents"         ",
                    IDM_HELP_CONTENTS
                MENUITEM "&TCLMAN"          ",
                    IDM_HELP_TCLMAN
                MENUITEM SEPARATOR
                MENUITEM "&Exit"            ",
                    IDM_HELP_EXIT
            END
    END
END
```

## Appendix E : Hardware data

All physical devices, link, wire etc. are specified by a code:

[TLDDD.PP]

Where T = device type, L = Location, DDD = device number, PP = device sub-unit (wire number, connector number). If multiple devices exist, an 'x' specifies all devices. Note that the connector number is the inverse of the manufacturers' listing.

Entry	T (Device type)	L (Location)
0	WIRE	ELSEWHERE
1	CONNECTOR	TELESCOPE WARM ROOM
2	FUSE	TELESCOPE COLD ROOM
3	SWITCH	PUMP ROOM
4	TRANSFORMER	ISOLATOR ROOM
5	ACTUATOR	R. A. REFERENCE UNIT
6	SENSOR / LIMIT UNIT	DEC REFERENCE UNIT
7	ENCODERS	R. A. OFFSET UNIT
8	UNDEFINED UNIT	DEC OFFSET UNIT
9	UNDEFINED UNIT	DOME UNIT

### E.1 : Device list

Code	Physical device	Code	Physical Device
11000	Spine (in warm room)	0x000	Earth strap
12000	Spine (in cold room)	00001	Input power cable
11001	Input mains connector	01002	Input power link
11011	Switch panel connector 1	34000	Input isolator switch
11012	Switch panel connector 2	34001	Isolator switch (console)
11013	Switch panel connector 3	34002	Isolator switch (hydraulics)
11014	Switch panel connector 4	31001	Single phase power switch
11015	Switchpanel connector 5	31002	Three phase power switch
11021	Fuse panel connector 1	31003	Reference focus motor switch
11022	Fuse panel connector 2	31004	Offset focus motor switch
11023	Fuse panel connector 3	41001	Three phase isolator transformer
11024	Fuse panel connector 4	41002	50VAC transformer
11025	Fuse panel connector 5	41003	PMT heater transformer
11030	Low voltage shelf connector	41004	+12VDC transformer
11041	Network box mains input	41005	-12VDC (A) transformer
11042	Network box power out	41006	-12VDC (B) transformer
11043	Network box sensor inputs	41007	50VDC transformer
11044	Network box RS232 Data I/O	41008	24VDC transformer
11045	Network box display	22001	Oil pump fuse
12001	Clamp sensor	22002	Oil pump fuse

1x011	Unit x: 3 phase connector	22002	Oil pump fuse
1x012	Unit x: Fine motor	22003	Dome shutter fuse
1x013	Unit x: Input limit switches	22004	Dome shutter fuse
1x014	Unit x: Auxiliary outputs	22005	Dome shutter fuse
1x015	Unit x: Unit power and comms	22006	Ra slew fuse
1x016	Unit x: Input encoder bank 1	22007	Ra slew fuse
1x017	Unit x: Input encoder bank 2	22008	Ra slew fuse
1x018	Unit x: Input encoder bank 3	22009	Ra clamp fuse
1x019	Unit x: Input encoder bank 4	22010	Ra clamp fuse
1x020	Unit x: Manual override 25DM	22011	Ra clamp fuse
10030	Manual override 25DF	22012	Dec slew fuse
1x101	[POWER] : Power / com in	22013	Dec slew fuse
1x102	[POWER] : IO / clock bus	22014	Dec slew fuse
1x103	[POWER] : IO / clock bus	22015	Dec clamp fuse
1x104	[POWER] : IO / clock bus	22016	Dec clamp fuse
1x201	[CPU] : CPU bus	22017	Dec clamp fuse
1x301	[I/O] : IO bus	22021	DEC. reference armature
1x302	[I/O] : MSB coarse encoder in	22022	DEC. reference field
1x303	[I/O] : LSB coarse encoder in	22023	R.A. offset armature
1x304	[I/O] : MSB fine encoder in	22024	R.A. offset field
1x305	[I/O] : LSB fine encoder in	22025	DEC. offset armature
1x306	[I/O] : Limit bank A in	22026	DEC. offset field
1x307	[I/O] : Limit bank B in	62001	R.A. reference limit CW
1x308	[I/O] : Manual override	62002	R.A. reference limit CW
1x309	[I/O] : Limit input	62003	R.A. reference clutch in
1x310	[I/O] : Auxiliary output	62004	R.A. reference clamp in
1x311	[I/O] : Motor out A/B	62005	R.A. reference clamp out
1x312	[I/O] : Motor out C/D	62011	DEC reference limit CW
1x313	[I/O] : Timing functions	62012	DEC reference limit CCW
1x401	[CLOCK] : Clock bus	62013	DEC reference fine limit CW
1x402	[CLOCK] : Clock TTL input	62014	DEC reference fine limit CCW
1x501	[SERIAL] : Incremental encoder	62015	DEC reference centre detent
1x502	[SERIAL] : LSB out	62016	DEC clamp in
1x503	[SERIAL] : MSB out	62017	DEC clamp out
1x601	[PROTECT] : Input	62021	R.A. offset limit CW
1x602	[PROTECT] : Output	62022	R.A. offset limit CCW
1x701	[NETWORK] : Power sesor in	62023	R.A. offset centre detent
1x702	[NETWORK] : LED bank A out	62031	DEC offset limit CW
1x703	[NETWORK] : LED bank B out	62032	DEC offset limit CCW
1x704	[NETWORK] : Relay drive	62033	DEC offset centre detent
1x705	[NETWORK] : Comms / power	70001	R.A. reference coarse encoder
1x801	Override handset connector	70002	R.A. reference fine encoder
19811	Dome opto-isolator in	70003	DEC reference coarse encoder
19812	Dome opto-isolator out	70004	DEC reference fine encoder

15001	R.A. opto-isolator in	70005	R.A. offset incremental encoder
15002	R.A. opto-isolator out	70006	DEC offset incremental encoder
11031	Dome rotation unit CPU in	70007	Dome incremental encoder
11032	Dome rotation unit power	82001	Reference PMT coolers.
11033	Dome CW/CCW valve output	82002	Offset PMT coolers
11034	Dump valve enable, output	82003	CCD camera control unit
11035	Pump motor on /off	82004	CUBE unit
13001	Low voltage input	81001	DCS PC (console)
12011	Sidereal drive door connector	81002	CUBE monitor
12012	Sidereal motor connector	81003	CCD camera video display
11051	Desk multibloc connectors	81004	Dome aximuth display
12011	Scope unswitched dual skt	81005	PMT desk
12012	Scope switch dual skt	81006	EHT power supply
		82011	Reference window heater
52001	R.A. reference slew motor	82013	Offset window heater
52002	R.A. reference clamp	81031	Dome rotation controller
52003	R.A. reference clutch	81041	Sidereal drive amplifier
52004	R.A. reference sidereal motor	85001	R.A. opto-isolator (sidereal i/p)
52011	Dec reference slew motor	89001	Dome o/p opto-isolator
52012	Dec reference clamp		
52013	Dec reference fine motor		
52021	Ra offset fine motor		
52031	Dec offset fine motor		
52041	Reference focus motor		
52051	Offset focus motor		
53061	Hydraulic pump motor		
53062	Hydraulic pump dump valve		
53063	Hydraulic pump CW valve		
53064	Hydraulic pump CCW valve		

**E.2 : Spine connector**

Pin	Description	Voltage	Device	Fuse	Wire Label	Wire Color
1	3 phase delta out: YELLOW	415VAC	Dec clamp	1A	C	GREEN
2	3 phase delta out: RED	415VAC	Dec clamp	1A	C	BLUE
3	3 phase delta out: BLUE	415VAC	Dec clamp	1A	C	RED
4	3 phase delta out: YELLOW	415VAC	Dec slew	1A	C	BROWN
5	3 phase delta out: RED	415VAC	Dec slew	1A	C	BLACK
6	3 phase delta out: BLUE	415VAC	Dec slew	1A	C	WHITE
7	3 phase delta out: YELLOW	415VAC	Ra clamp	1A	C	YELLOW
8	3 phase delta out: RED	415VAC	Ra clamp	1A	C	GREY
9	3 phase delta out: BLUE	415VAC	Ra clamp	1A	C	PURPLE
10	3 phase delta out: YELLOW	415VAC	Ra slew	1A	C	ORANGE
11	3 phase delta out: RED	415VAC	Ra slew	1A	C	GREEN
12	3 phase delta out: BLUE	415VAC	Ra slew	1A	C	PINK
13	3 phase delta out: YELLOW	415VAC	Oil pump	1A	D	GREEN
14	3 phase delta out: RED	415VAC	Oil pump	1A	D	BLACK
15	3 phase delta out: BLUE	415VAC	Oil pump	1A	D	WHITE
16	3 phase delta out: YELLOW	415VAC	Dome shutter	1A	D	RED
17	3 phase delta out: RED	415VAC	Dome shutter	1A	D	YELLOW
18	3 phase delta out: BLUE	415VAC	Dome shutter	1A	D	BLUE
19	(23 broken internally.) use 19					
20	Field	12VDC (1 phase)	Reference DEC	1A	A	YELLOW
21	Field	0VDC RETURN	Reference DEC		A	PINK
22	Armature	50VDC	Reference DEC	1A	A	BROWN
23	Armature	0VDC	Reference DEC		A	BLUE
24	Totem power	+12VDC	Reference DEC	1A	B	YELLOW

25	Totem power	0VDC	Reference DEC	[86501]		B	GREY
26	Totem power	-12VDC	Reference DEC	[86501]	1A	B	PINK
27	Field	12VDC (1 phase)	Offset R.A.	[87501]	1A	A	WHITE
28	Field	0VDC RETURN	Offset R.A.	[87501]		A	ORANGE
29	Armature	50VDC	Offset R.A.	[87501]	1A	A	DARK GREEN
30	Armature	0VDC	Offset R.A.	[87501]		A	LIGHT GREEN
31	Totem power	+12VDC	Offset R.A.	[87501]	1A	B	ORANGE
32	Totem power	0VDC	Offset R.A.	[87501]		B	PURPLE
33	Totem power	-12VDC	Offset R.A.	[87501]	1A	B	LIGHT GREEN
34	Field	12VDC (1 phase)	Offset DEC	[88501]	1A	A	RED
35	Field	0VDC RETURN	Offset DEC	[88501]		A	PURPLE
36	Armature	50VDC	Offset DEC	[88501]	1A	A	BLACK
37	Armature	0VDC	Offset DEC	[88501]		A	GREY
38	Totem power	+12VDC	Offset DEC	[88501]	1A	B	RED
39	Totem power	0VDC	Offset DEC	[88501]		B	BLACK
40	Totem power	-12VDC	Offset DEC	[88501]	1A	B	BLUE
41		nc					
42	Microcontroller unit POWER	+20VDC	Reference DEC unit	[86000]		Discrete	RED
43	Microcontroller unit RETURN	0VDC	Reference DEC unit	[86000]		Discrete	BLACK
44	Microcontroller unit POWER	+20VDC	Offset R.A. unit	[87000]		Discrete	RED
45	Microcontroller unit RETURN	0VDC	Offset R.A. unit	[87000]		Discrete	BLACK
46	Microcontroller unit POWER	+20VDC	Offset DEC unit	[88000]		Discrete	RED
47	Microcontroller unit RETURN	0VDC	Offset DEC unit	[88000]		Discrete	BLACK
49	Microcontroller unit POWER	+20VDC	Counter unit			Discrete	RED
50	Microcontroller unit RETURN	0VDC	Counter unit			Discrete	BLACK
51	Microcontroller unit POWER	+20VDC	Tertiary unit			Discrete	RED
52	Microcontroller unit RETURN	0VDC	Tertiary unit			Discrete	BLACK

	Max connected side: 52								
61	Reference window heater	6.2VAC			[82011]			Mains cable	BLUE
62	Reference window heater	6.2VAC			[82011]			Mains cable	BROWN
63	offset window heater	6.2VAC			[82012]				
64	offset window heater	6.2VAC			[82012]				
65	Earth				[81021]				
66	Line driver	0VDC			[81021]				
67	Line driver	+15VDC			[81021]				
68	Reference focus BLUE	50VAC			[52041]				
69	Reference focus RED	50VAC			[52041]				
70	Reference focus YELLOW	50VAC			[52041]				
71	offset focus BLUE	50VAC			[52051]				
72	offset focus RED	50VAC			[52051]				
73	offset focus YELLOW	50VAC			[52051]				
74	Oil heater	240VAC			[82072]				
75	Oil heater	0VAC			[82072]				
76	Oil heater	0VAC (in)			[82072]				
77	Clutch POWER	+24VDC			[52003]	Clutch		Discrete	RED
78	Clutch RETURN	0VDC			[52003]	Clutch		Discrete	BLACK
79	Microcontroller power	+20VDC			[85000]	R.A. Reference unit		Discrete	RED
80	Microcontroller power	0VDC			[85000]	R.A. Reference unit		Discrete	BLACK
81	Reserved sidereal drive	+32VDC			[81041]				
82	Reserved sidereal drive	0VDC			[81041]				
83	Reserved sidereal drive	-32VDC			[81041]				
84	Reserved sidereal drive	2.5V (Triag)			[81041]				
85	Reserved sidereal drive	2.5V (Triag)			[81041]				

86	Reserved sidereal drive								
117	Auxiliary RETURN	0VAC		scope switched	[12012]		Discrete		BLACK
118	Auxiliary POWER (Switched)	240VAC		scope switched	[12012]		Discrete		PINK
119	Auxiliary RETURN	0VAC		scope unswitched	[12011]		Discrete		BLACK
120	Auxiliary POWER (Unswitched)	240VAC		scope unswitched	[12011]		Discrete		PINK

### **E.3 : CPU memory and I/O logic : GAL pinout**

Pin	I/O	Description	Pin	I/O	Description
1	I	10MHz CLK	20	-	5V
2	I	A11	19	O	BEN : Data bus enable
3	I	A12	18	O	IRD : I/O Port read
4	I	A13	17	O	IWR : I/O Port write
5	I	A14	16	O	EOU : EPROM /OE
6	I	A15	15	O	EEN : EPROM /CE
7	I	/MREQ	14	O	RWR : RAM /WR
8	I	/WR	13	O	ROU : RAM /OE
9	I	/RD	12	O	REN : RAM /CE
10	-	0V	11	I	GAL chip enable

### **E.4 : Motor limit : GAL pinout**

Pin	I/O	Description	Pin	I/O	Description
1	I	NOT USED	20	-	5V
2	I	MEA : Motor A en.	19	O	NOT USED.
3	I	MEB : Motor B en.	18	O	NOT USED.
4	I	LAL : CW limit A.	17	O	BDOUT : Motor B dir.
5	I	LAH : CCW limit A.	16	O	BEOUT : Motor B out.
6	I	MDA : Motor A dir.	15	O	NOT USED
7	I	MDB : Motor B dir.	14	O	NOT USED.
8	I	LBL : CW limit B.	13	O	ADOUT : Motor A dir.
9	I	LBH : CCW limit B.	12	O	AEOUT : Motor A out
10	-	0V	11	I	

### **E.5 : Bus control : GAL pinout**

Pin	I/O	Description	Pin	I/O	Description
1	I	Sensor Ok	20	-	5V
2	I	CPU power	19	O	System OK
3	I	Standby power	18	O	415V Relay Drive
4	I	415V in	17	O	CPU Relay Drive
5	I	Reset	16	O	CPU Error
6	I	Stop /Resume	15	O	Bus Error
7	I	Line return	14	O	System Error
8	I	TX / RX	13	O	System Reset
9	I	TX request	12	O	Line Error
10	-	0V	11	I	CPU Error in

**CAUTION :** Programming this gal will cause unpredictable results in system wide control.

### E.6 : Bus control : Terminal block connectors

Pin	Mains Power [11041]	Power out [11042]	Sensor inputs [11043]
1	240VAC LIVE	CPU power out ( 15V )	0V
2	240VAC NEUTRAL	Front Panel Fuse	Relay Disable
3	GROUND	CPU power Out ( 0V )	Line return
4	nc	Front Panel Fuse	Stop / Resume
5	nc	GROUND case	Reset
6	nc	nc	415V on
7	nc	nc	Standby Power
8	nc	nc	CPU Power
9	nc	nc	/ Sensor OK
10	nc	TXRX	nc
11	nc	TXREQ	nc
12	nc	Signal ground	5V

### E.7 : Bus control : RS232 pinout

Pin	RS 232 Port [11044]	IBM display [11045]
1	nc	nc
2	RX in	'Bus error'
3	TX out	'Line error.'
4	RTS	'System error'.
5	CTS	'Stop / resume'.
6	Signal ground	'CPU error'
7	DTR	'System reset'
8	nc	nc
9	nc	nc
10	nc	LED return
11	nc	nc
12	nc	nc
13	nc	nc
14	nc	nc.
15	nc	'TX request'
16	nc	'TXRX'
17	nc	'System ok'
18	nc	'415V power'
19	nc	'CPU power'
20	nc	'Standby power'
21	nc	nc
22	nc	nc
23	nc	LED return
24	nc	nc
25	nc	nc.

Note that it is here that the TX / RX signals are crossed over.

### E.8 : Bus control : Display front end

STANDBY POWER	CPU POWER	415V POWER	SYSTEM OK	TX RX	TX REQUEST
SYSTEM RESET	CPU ERROR	STOP RESUME	SYSTEM ERROR	LINE ERROR	BUS ERROR

### E.9 : DC axial motors

DC axial motors.	[52013] [52021] [52031]
Application:	DEC reference R.A. & DEC offset
manufacturer:	Evershed and Vignoles Ltd. London W4
Type:	FAG 101/N4/BD
Field Voltage:	12V / winding
armature voltage:	12V

### E.10 : Offset motor wiring

	OFFSET DEC (GRAY Multicore)	OFFSET R.A. (BLACK Multicore)
CCW LIMIT	DARK GREEN	DARK GREEN
CCW LIMIT	LIGHT BLUE	LIGHT GREEN
CW LIMIT	PINK	PINK
CW LIMIT	ORANGE	ORANGE
FIELD 1	BROWN	BROWN
FIELD 2	BLACK	BLACK
FIELD COMMON	BLUE	BLUE
ARMATURE	RED	RED
ARMATURE	PURPLE	PURPLE
ZERO POT V+	YELLOW	YELLOW
ZERO POT V-	GRAY	GRAY
ZERO POT WIPER	WHITE	WHITE

### E.11 : Reference R.A. : Slew motor

Slew motor [52001].	
Manufacturer	ANSLADO
Dealer	POWERDRIVE PSD Ltd.
Model	A16\ 060 2C A 224
Voltage	220VAC
Current	0.85/0.5 A
Power	0.12kW
Phase	3Phase
Operation	Delta configuration

### E.12 : R.A. Clamp

Motor 240VAC 3 phase delta

limit (upper only) : Magnetic sensor switch

Magnet (RS stock number 338-759) mounted on rotor arm.

2 sensors (RS stock number 339-213) mounted on main body of telescope

Clamp multibloc connections (reading left to right (1 to 6))

Multibloc [12001]	upper sensor	Lower sensor	output
1	blue		red
2	red	red	green
3	white	blue	black
4		white	
5 -nc-			
6 -nc-			

<b>Clamp motor [52002].</b>		
dealer	POWER DRIVE	
manufacturer	AEG	
type	AD 63 NZ Z 312	
number	2732686	
operation	3phase (Delta / Star)	
voltage	220 / 380 (Delta / Star)	V
current	1:37 / 0:75	A
power	0.18	KW
<b>clamp gearbox.</b>		
manufacturer	ROSSI MOTORDIDOTTORI	
type	MRV50PIIP	
power	0.18	KW
rate	29.8	RPM
<b>Clamp microswitch [62001].</b>		
Magnet	RS 338-759	
Sensors	RS 339-213	

**E.13 : Reference R.A. : Clutch unit**

<b>Clutch [52003]</b>	
Manufacturer	MATRIX
Operating voltage	24VDC
Clutch in / out sensor [62003]	Magnetic proximity / reed relay.
RS stock no.	338-743
<b>Clutch gearbox.</b>	
Manufacturer	ROSSI MOTTORIDUTTORI
Model	MR 2V 85 P11A
Built	82
P1	0.12kW
n2	0.92 /min

**E.14 : Sidereal drive**

<b>Fine / Sidereal Drive [52004]</b>	
Manufacturer	UNIMATIC ???
Dealer	UNIMATIC ENGINEERS Ltd GRANVILLE ROAD WORKS LONDON, ENGLAND NW2 2LN
Type	STEPPER
Voltage	32V

**E.15 : Mains power back connectors**

Pin	Connector [11011]	Connector [11012]	Connector [11013]	Connector [11014]	Connector [11015]
1	240VAC 1 phase switched out	240VAC 1 phase unswitched out	Main Dec armature in	240VAC 1 phase switched out	415VAC 3 phase delta in: BLUE
2	240VAC 1 phase unswitched out	1 phase NEUTRAL	Main dec armature out	1 phase NEUTRAL	415VAC 3 phase delta in: RED
3	1 phase NEUTRAL out	240VAC 1 phase unswitched out	Offset Ra armature in	240VAC 1 phase switched out	415VAC 3 phase delta in: YELLOW
4	240VAC 3 phase BLUE switched out	1 phase NEUTRAL	Offset Ra armature out	1 phase NEUTRAL	nc
5	240VAC 3 phase RED switched out	240VAC 1 phase unswitched out	Offset Dec armature in	240VAC 1 phase switched out	415VAC 3 phase delta out: BLUE Oil pump 1A
6	240VAC 3 phase YELLOW switched out	1 phase NEUTRAL	Offset Dec armature out	1 phase NEUTRAL	415VAC 3 phase delta out: RED Oil pump 1A
7	3 phase NEUTRAL out	240VAC 1 phase unswitched out		240VAC 1 phase switched out	415VAC 3 phase delta out: YELLOW Oil pump 1A
8	240VAC 3 phase BLUE in	1 phase NEUTRAL	24VDC out	1 phase NEUTRAL	nc
9	240VAC 3 phase RED in	240VAC 1 phase unswitched out	24VDC RETURN	240VAC 1 phase switched out	415VAC 3 phase delta out: BLUE Shutter 1A
10	240VAC 3 phase YELLOW in	1 phase NEUTRAL	+12VDC I/O (Distribution)	1 phase NEUTRAL	415VAC 3 phase delta out: RED Shutter 1A
11	3 phase NEUTRAL in	240VAC 1 phase unswitched out	0VDC I/O (Distribution)	240VAC 1 phase switched out	415VAC 3 phase delta out: YELLOW Shutter 1A
12	Earth	1 phase NEUTRAL	-12VDC I/O (Distribution)	1 phase NEUTRAL	nc

**E.16 : Fuse panel back connectors**

Pin	Connector [11021]	Connector [11022]	Connector [11023]	Connector [11024]	Connector 11025]
1	415VAC 3 phase delta in BLUE	415VAC 3 phase delta out 1A BLUE : Ra slew	nc	12VDC in BLUE	240VAC 1 phase in PINK : Switched
2	415VAC 3 phase delta in RED	415VAC 3 phase delta out 1A RED : Ra slew	nc	0VDC in BLACK	1 phase return in BLACK
3	415VAC 3 phase delta in YELLOW	415VAC 3 phase delta out 1A YELLOW : Ra slew	nc	12VDC out 1A BLUE : Offset ra	240VAC 1 phase in PINK : Unswitched
4	nc	415VAC 3 phase delta out 1A BLUE : Ra clamp	nc	0VDC out 1A BLACK : Offset ra	1 phase return in BLACK
5	nc	415VAC 3 phase delta out 1A RED : Ra clamp	nc	12VDC out 1A BLUE : Offset dec	240VAC 1 phase out 1A PINK : Desk switched
6	nc	415VAC 3 phase delta out 1A YELLOW : Ra clamp	nc	0VDC out 1A BLACK : Offset dec	1 phase return out BLACK
7	nc	415VAC 3 phase delta out 1A BLUE : Dec slew	nc	12VDC out 1A BLUE : Main dec	240VAC 1 phase out 1A PINK : Scope switched
8	nc	415VAC 3 phase delta out 1A RED : Dec slew	nc	0VDC out 1A BLACK : Main dec	1 phase return out BLACK
9	nc	415VAC 3 phase delta out 1A YELLOW : Dec slew	nc	50VAC 1 phase in PINK	240VAC 1 phase out 1A PINK : Desk unswitched
10	nc	415VAC 3 phase delta out 1A BLUE : Dec clamp	nc	50VAC 1 phase in PINK	1 phase return out BLACK
11	nc	415VAC 3 phase delta out 1A RED : Dec clamp	nc	50VAC 1 phase out 1A PINK : focus motors	240VAC 1 phase out 1A PINK : Scope unswitched
12	nc	415VAC 3 phase delta out 1A YELLOW : Dec clamp	nc	50VAC 1 phase out 1A PINK : focus motors	1 phase return out BLACK

### E.17 : Low voltage shelf

The low voltage shelf is situated in the main power cabinet under the EHT unit. It holds the transformers for +12VDC, -12VDC, -12VDC, PMT heaters and 24VDC, 50VDC supplies, both situated in the Cromenco power supply box.

Pin	Low voltage connector	[11030]
1	240VAC 1 phase LIVE unswitched in	
2	240VAC NEUTRAL	
3	240VAC 1 phase LIVE switched in	
4	240VAC NEUTRAL	
5	EARTH	
6	EARTH	
7	+12V out	
8	return	
9	-12V out	
10	return	
11	-12V out	
12	return	

**E.18 : R.A. reference output**

Bit	FFF0h]		FFF1h	
STATE	0	1	0	1
0			SID GUIDE-	CANCEL
1			SID SET+	CANCEL
2			SID SET-	CANCEL
3			SID GIUDE+	CANCEL
4				
5	CLUTCH OUT	CLUTCH IN		
6	CLAMP ON	CLAMP OFF	CLAMP OUT	CLAMP IN
7	SLEW ON	SLEW OFF	SLEW CCW	SLEW CW

**E.19 : R.A. reference inputs**

Bit	FFF0h [15302]	FFF1h [15303]	FFF2h [15304]	FFF3h [15305]	FFF4h [15306]	FFF5h [15307]
0	low encoder	low encoder	1	high encoder	ccw limit [1]	1
1	low encoder	low encoder	1	high encoder	cw limit [1]	1
2	low encoder	low encoder	1	high encoder	clamp out [1]	1
3	low encoder	1	1	high encoder	0	1
4	low encoder	1	1	high encoder	ccw limit [1]	1
5	low encoder	clutch in [0]	1	1	cw limit [1]	1
6	low encoder	clamp overload [0]	1	1	0	1
7	low encoder	-	1	1	0	1

**E.20 : DEC. reference output**

	FFF0h		FFF1h	
STATE	0	1	0	1
0				
1	ENABLE	DISABLE		
2	FINE CCW	FINE CW		
3	BRAKE ON	BRAKE OFF		
4				
5				
6	CLAMP ON	CLAMP OFF	CLAMP IN	CLAMP OUT
7	SLEW ON	SLEW OFF	SLEW CCW	SLEW CW

**E.21 : DEC. reference inputs**

Bit	FFF0h [16302]	FFF1h [16303]	FFF2h [16304]	FFF3h [16305]	FFF4h [16306]	FFF5h [16307]
0	high encoder	1	low encoder	low encoder	1	1
1	high encoder	1	low encoder	low encoder	1	1
2	high encoder	1	low encoder	low encoder	fine low [1]	1
3	high encoder	1	low encoder	1	fine high [1]	1
4	high encoder	1	low encoder	1	clamp out [1]	1
5	high encoder	1	low encoder	1	clamp in [1]	1
6	1	1	low encoder	1	CCW limit [1]	1
7	1	fine centre	low encoder	1	CW limit [1]	1

**E.22 : R.A. offset outputs**

Bit	FFF0h		FFF1h	
STATE	0	1	0	1
0			DIR RA+	DIR RA-
1			BRAKE OFF	BRAKE ON
2			UNUSED	UNUSED
3			DISABLE	ENABLE
4				
5				
6				
7				

**E.23 : R.A. offset inputs**

Bit	FFF0h [17302]	FFF1h [17303]	FFF2h [17304]	FFF3h [17305]	FFF4h [17306]	FFF5h [17307]
0	1	1	Centre	1	Limit CW [0]	
1	1	1	Phase 0	1	Limit CCW [0]	1
2	1	1	Phase 90	1	1	1
3	1	1	SYNC [1]	1	1	1
4	1	1	1	1	1	1
5	1	1	1	1	1	1
6	1	1	1	1	1	1
7	1	1	1	1	1	1

**E.24 : DEC. offset outputs**

Bit	FFF0h		FFF1h	
STATE	0	1	0	1
0			DIR DEC +	DIR DEC-
1			BRAKE OFF	BRAKE ON
2			UNUSED	UNUSED
3			DISABLE	ENABLE
4				
5				
6				
7				

**E.25 : DEC. offset inputs**

Bit	FFF0h [18302]	FFF1h [18303]	FFF2h [18304]	FFF3h [18305]	FFF4h [18306]	FFF5h [18307]
0	1	1	Centre	1	Limit CW [0]	1
1	1	1	Phase 0	1	Limit CCW [0]	1
2	1	1	Phase 90	1	1	1
3	1	1	SYNC [1]	1	1	1
4	1	1	1	1	1	1
5	1	1	1	1	1	1
6	1	1	1	1	1	1
7	1	1	1	1	1	1

**E.26 : Dome outputs**

Bit	FFF0h		FFF1h	
STATE	0	1	0	1
0	SLEW CCW	INHIBIT	SLEW CW	INHIBIT
1	F. SET CCW	INHIBIT	F. SET CW	INHIBIT
2	S. SET CCW	INHIBIT	S. SET CW	INHIBIT
3	GUIDE CCW	INHIBIT	GUIDE CW	INHIBIT
4				
5				
6				
7				

**E.27 : Dome inputs**

Bit	FFF0h [19302]	FFF1h [19303]	FFF2h [19304]	FFF3h [19305]	FFF4h [19306]	FFF5h [19307]
0	1	1	phase 0	1	1	1
1	1	1	phase 90	1	1	1
2	1	1	Sync	1	1	1
3	1	1	1	1	1	1
4	1	1	1	1	1	1
5	1	1	1	1	1	1
6	1	1	1	1	1	1
7	1	1	1	1	1	1

### E.28 : Encoders : Reference R.A. / DEC. coarse / fine

Ra reference encoders: Coarse/fine encoders linked via 25way D plug. Wire colors are constant.

Pin	Colour	Coarse Encoders	Fine Encoders
1	BLUE/BLACK	D0	D10/MSB
2	WHITE/BLUE	D1	D9
3	YELLOW/GREEN	D2	D8
4	PURPLE	D3	D7
5	RED/BROWN	D4	D6
6	RED/BLACK	D5	D5
7	WHITE/GREEN	nc	D4
8	WHITE/RED	nc	D3
9	YELLOW	nc	D2
10	BLACK	nc	D1
11	WHITE	nc	D0/LSB
12	BROWN	nc	spare
13	ORANGE/D.GREEN	nc	spare
14	PINK	nc	spare
15	RED	nc	output ground
16	L.GREEN	nc	circuit ground
17	ORANGE	nc	spare
18	D.BLUE	nc	+5VDC
19	D.GREEN	nc	spare
20	YELLOW/RED	nc	spare
21	YELLOW/L.GREEN	nc	spare
22	ORANGE/L.GREEN	+5VDC	spare
23	GREY	+5VDC	Case ground
24	RED/BLUE	0VDC	Case input
25	RED/L.GREEN	0VDC	Reverse count

### E.29 : Dome hydraulic pump

<b>Hydraulic Pump.[53061]</b>	
Manufacturer:	Vickers, England
Model:	V 210-5-1A-12
ref:	H4 2396/1/2
<b>Pump motor [52061].</b>	
Manufacturer:	Brook Motors Ltd.Huddersfield
Model:	Y322732
Frame:	C 215 /6E
Current:	5A avg. star(> 13A delta)
Voltage:	415VAC
Phase:	3 phase Star ( previously) Delta
RPM	950

### E.30 : Dome valves

<b>Dump valve [52062].</b>	
Manufactuer:	Flui - trol
Operating voltage:	24VDC
model:	A37A
<b>Direction valves [52063] [52064].</b>	
Manufactuer:	Sperry gyroscope co. LTD, Brentford, Essex.
operating voltage:	24VDC
coil resistance:	40ohms
max current:	600mA
max flow rate:	2.5 G.P.M.
max pressure:	3000 P.S.I.
Part number:	21172-0
model:	3025-L

RED + WHITE	Dump valve
BACK + WHITE	Dump valve
GREEN + WHITE	Direction valves return
YELLOW + WHITE	CW direction valve
BROWN + WHITE	nc
BLUE + WHITE	CCW direction valve

**E.31 : Interface units (general) : Multibloc connectors**

Pin	[1x016] coarse encoder.	[1x017] coarse encoder.	[1x018] fine encoder.	[1x019] fine encoder.	[1x015] cpu power / comms.	[1x014] limits / outputs.	[1x011] Motor power.	[1x013] High voltage limits.
1	0V	0V	0V	0V	15V CPU power input	0V	Motor 1 : 230VAC 3 phase in BLUE	0V
2	bit 1	bit 11	bit 1	bit 11	rx /tx input	output 1	Motor 1 : 230VAC 3 phase in BLUE	Motor 1 CW limit
3	bit 2	bit 12	bit 2	bit 12	txrequest	output 2	Motor 1 CCW limit	Motor 1 CCW limit
4	bit 3	bit 13	bit 3	bit 13	0V	output 3	Motor 2 CW limit	Motor 2 CW limit
5	bit 4	bit 14	bit 4	bit 14	nc	output 4	Motor 2 CCW limit	Motor 2 CCW limit
6	bit 5	bit 15	bit 5	bit 15	nc	output 5	Motor 3 CW limit	Motor 3 CW limit
7	bit 6	bit 16	bit 6	bit 16	nc	output 6	Motor 3 CCW limit	Motor 3 CCW limit
8	bit 7	Buffer Test ** (Connect to 0V)	bit 7	Buffer Test ** (Connect to 0V)		output 7	Motor 4 CW limit	Motor 4 CW limit
9	bit 8	Buffer Test ** (Connect to 0V)	bit 8	Buffer Test ** (Connect to 0V)	nc	output 8	Motor 4 CCW limit	Motor 4 CCW limit
10	bit 9	nc	bit 9	nc	nc	Sidereal Clk	nc	nc
11	bit 10	nc	bit 10	nc	nc	50Hz Sync	nc	nc
12	5V	5V	5V	5V	nc	5V	5V	5V

**\*\* CONNECTING THESE CAN DAMAGE UNIT IF DIRECTIONS ARE NOT PROPERLY OBSERVED**

The buffer tests are alternative inputs to bits 8,16 of the inputs.

**E.32 : Manual override : Connectors**

Pin	[1x308]	Colour	[10030]	[1x313]	Colour	[10030]
1	Manual select	Black	10	5V Digital	nc	nc
2	Fine enable	Brown	9	led anode	Green	17
3	Clamp enable	Red	8	led anode	Yellow	16
4	Clutch enable	Orange	7	Sidereal Clock	Orange	nc
5	Slew enable	Yellow	6	50Hz Sync In	Red	nc
6	Fine direction	Green	5	Fine Guide Ramp	Brown	14
7	Clamp direction	Blue	4	nc	nc	nc
8	Clutch direction	Purple	3	nc	nc	nc
9	Slew direction	Grey	2	nc	nc	nc
10	0V	White	1	0V	Black	15

**E.33 : Board connectors**

Pin	[1x101]	[1x313]
1	Power in	5V Digital
2	Power in	Manual override LED 'computer' anode
3	nc	Manual override LED 'Manual' anode
4	nc	Sidereal Clock out
5	0v	50Hz sync in
6	TXRX	Fine guide ramp inhibit
7	0v	nc
8	TX REQ	nc
9	0v	nc
10	0v	0V

Pin	[1x308], [1x801]	[1x306], [1x307], [1x309]
1	5v Digital	5V Digital
2	Fine enable	Limit input
3	Clamp enable	Limit input
4	Clutch enable	Limit input
5	Slew enable	Limit input
6	Fine direction	Limit input
7	Clamp direction	Limit input
8	Clutch direction	Limit input
9	Slew direction	Limit input
10	0v	0V

Pin	[1x102], [1x103], [1x104], [1x201], [1x301], [1x401]	Pin	[1x102], [1x103], [1x104], [1x201], [1x301], [1x401]
32	5V	31	0V
30	12V	29	0V
28	/NMI	27	/RST
26	/IO READ	25	/IO WRITE
24	/INT	23	/IO BOARD
22	/CLOCK BOARD	21	/POWER BOARD
20	nc	19	Data 7
18	Data 6	17	Data 5
16	Data 4	15	Data 3
14	Data 2	13	Data 1
12	Data 0	11	I/O Address 0
10	I/O Address 1	9	I/O Address 2
8	I/O Address 3	7	nc
6	/WATCH DOG OUT	5	Fine guide out
4	50Hz Sync in	3	Sidereal Drive out
2	/RXTX	1	/TX REQUEST

Pin	[1x302], [1x303], [1x304], [1x305], [1x502], [1x503]	[1x310], [1x601], [1x602]
1	5V Digital	5V Digital
2	Data bit 7	FFF0h bit 4
3	Data bit 6	FFF0h bit 5
4	Data bit 5	FFF0h bit 6
5	Data bit 4	FFF0h bit 7
6	Data bit 3	FFF1h bit 4
7	Data bit 2	FFF1h bit 5
8	Data bit 1	FFF1h bit 6
9	Data bit 0	FFF1h bit 7
10	0V	0V

Pin	[1x311]	[1x312]
1	12V relay out	12V relay out
2	Motor enable A out	Motor enable C out
3	Clamp A+B enable in	Clamp C+D enable in
4	Motor enable B out	Motor enable D out
5	Buffer enable in (See right)	Buffer enable in (Connect to pin 6)
6	5V digital	5V digital
7	Motor direction A	Motor direction C
8	Clamp A+B direction in	Clamp C+D direction in
9	Motor direction B	Motor direction D
10	0V	0V

Pin 5 was brought out to enable hot limit / safety cutouts to be integrated into the board. Connect to pin 6 (5V digital).

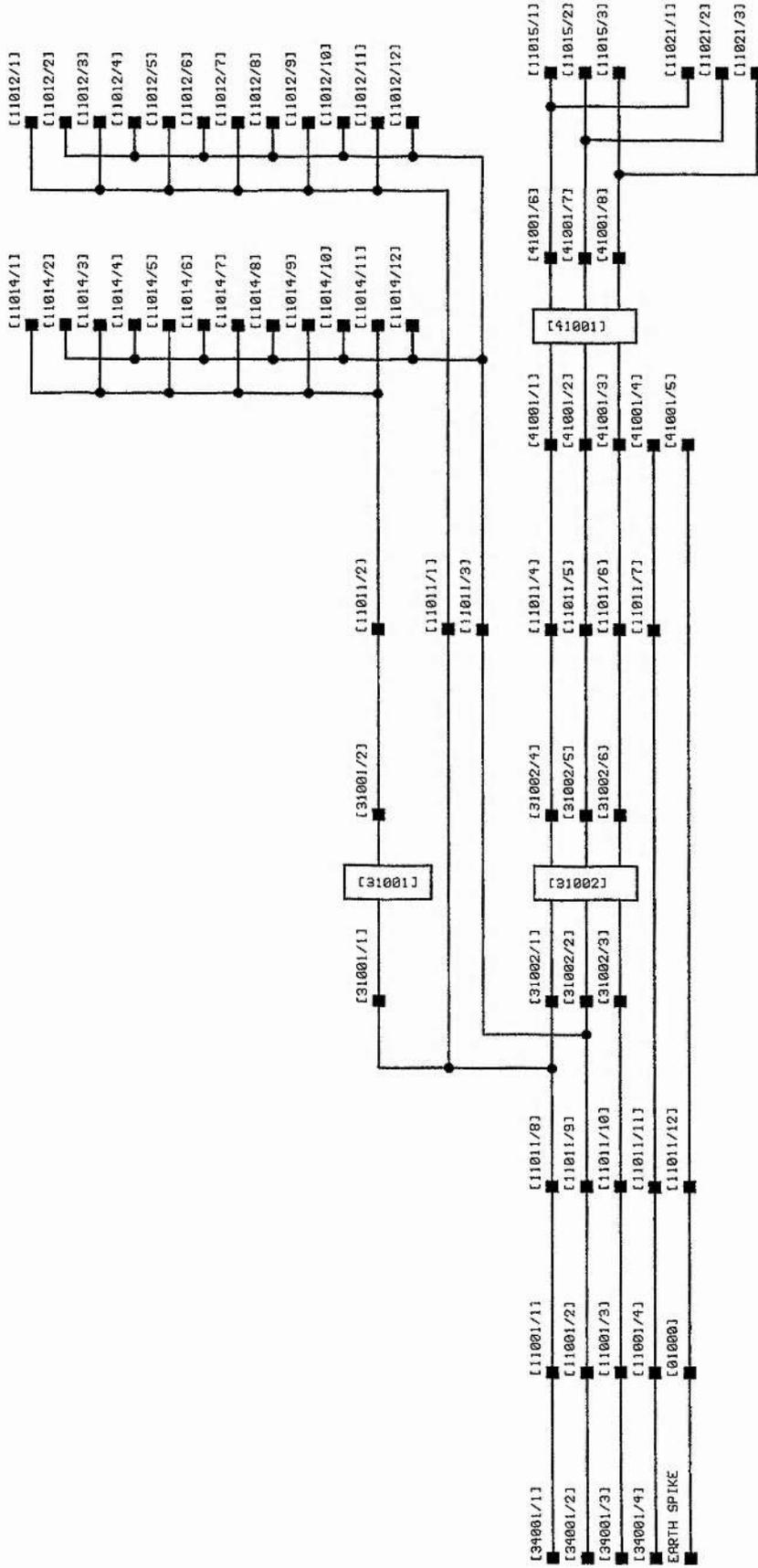
Pin	[1x701]	[1x704]	[1x705]	Int. color
1	5V	5V	0V	yellow / purple
2	/Sensor OK	CPU disable RLY A	nc	
3	CPU power	Clamp AB	Power in	grey
4	Standby power	415V disable RLY B	TX / RX	green
5	415V on	0V	TX request	blue
6	Reset	0V	CTS in	orange
7	Stop / resume	CPU disable RLY C	RTS out	red
8	Line return	Clamp CD	RX in	brown
9	Relay disable	415V disable RLY D	TX out	black
10	0V	0V	nc	

Pin	[1x702] LED bank A out	Ribbon	[11045]	[1x703] LED bank B	Ribbon	[11045]
1	5V	Black	25	5V	Black	14
2	CPU power relay	Brown	8	CPU error	Brown	6
3	System OK	Red	17	nc	Red	11
4	Standby power	Orange	20	System error	Orange	4
5	415V motor relay	Yellow	9	Stop / resume	Yellow	5
6	415V OK	Green	18	System reset	Green	7
7	CPU OK	Blue	19	Bus error	Blue	2
8	TX	Purple	16	Line error	Purple	3
9	TX request	Grey	15	System reset	Grey	13
10	0V	white	23	0V	White	10

## **Appendix F : Schematics**

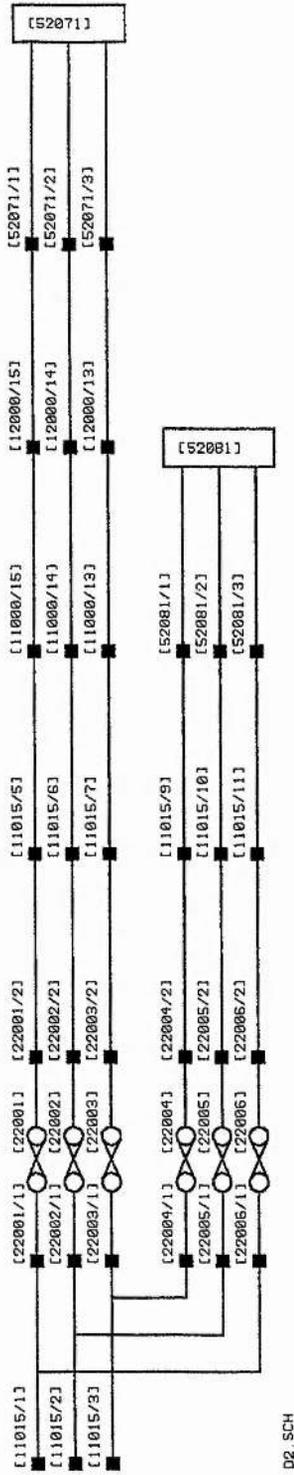
This appendix holds all schematics designed for the twin photometric telescope. Refer to appendix E.1 for listing of device codes.

# F.1 : Mains input, power switches and 3 phase isolator transformer



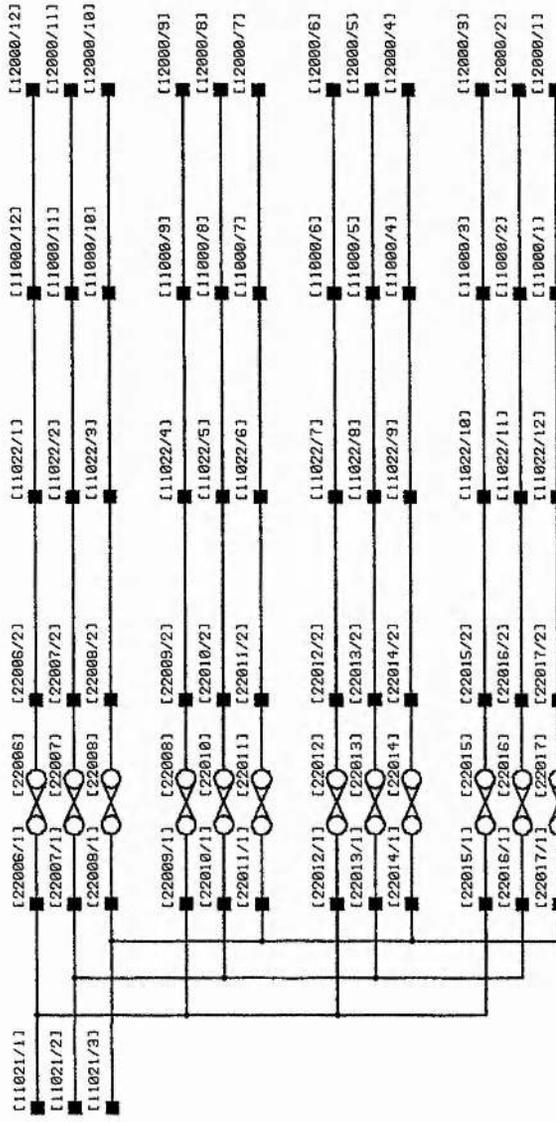
D1 SCH

## F.2 : Oil pump and dome shutter power supply



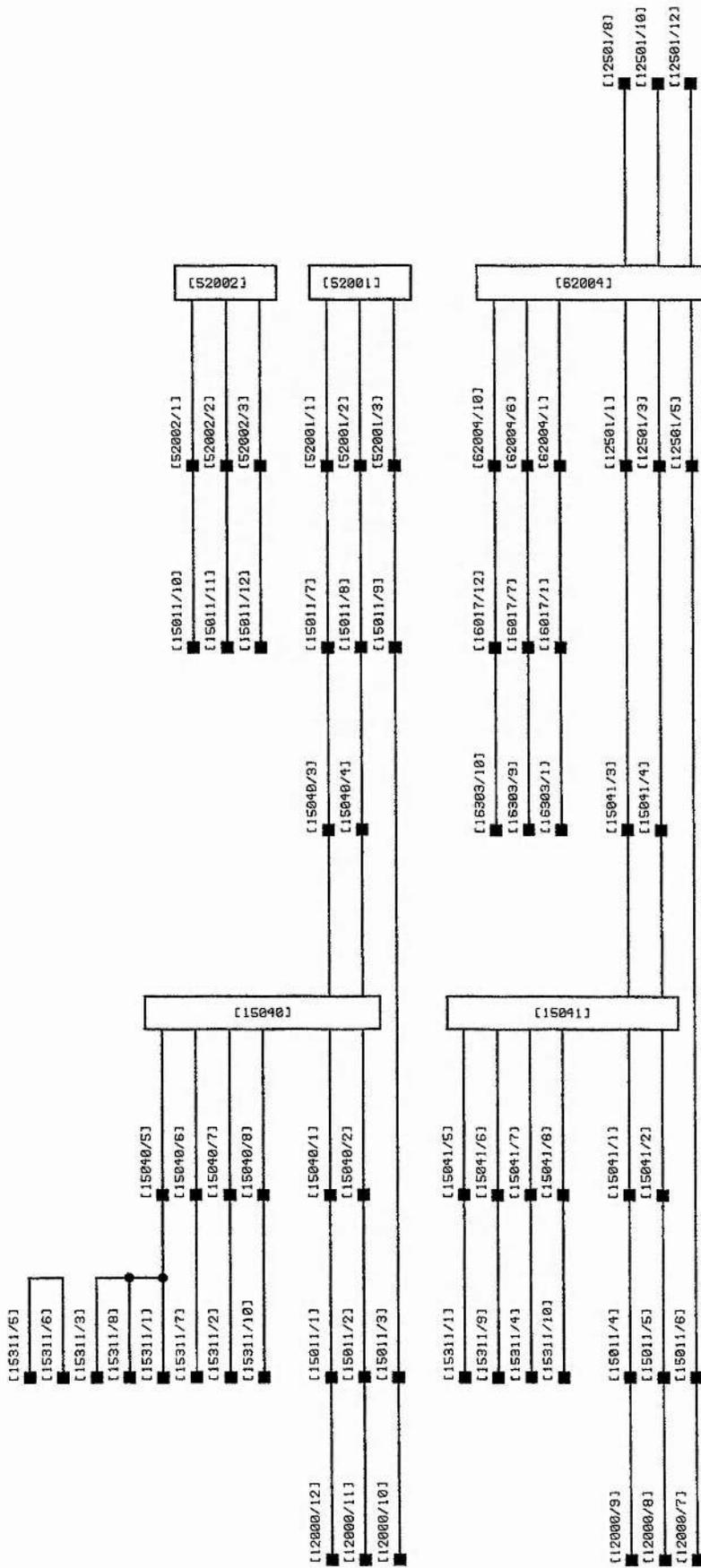
D2. SCH

### F.3 : 3 Phase motor power supply distribution



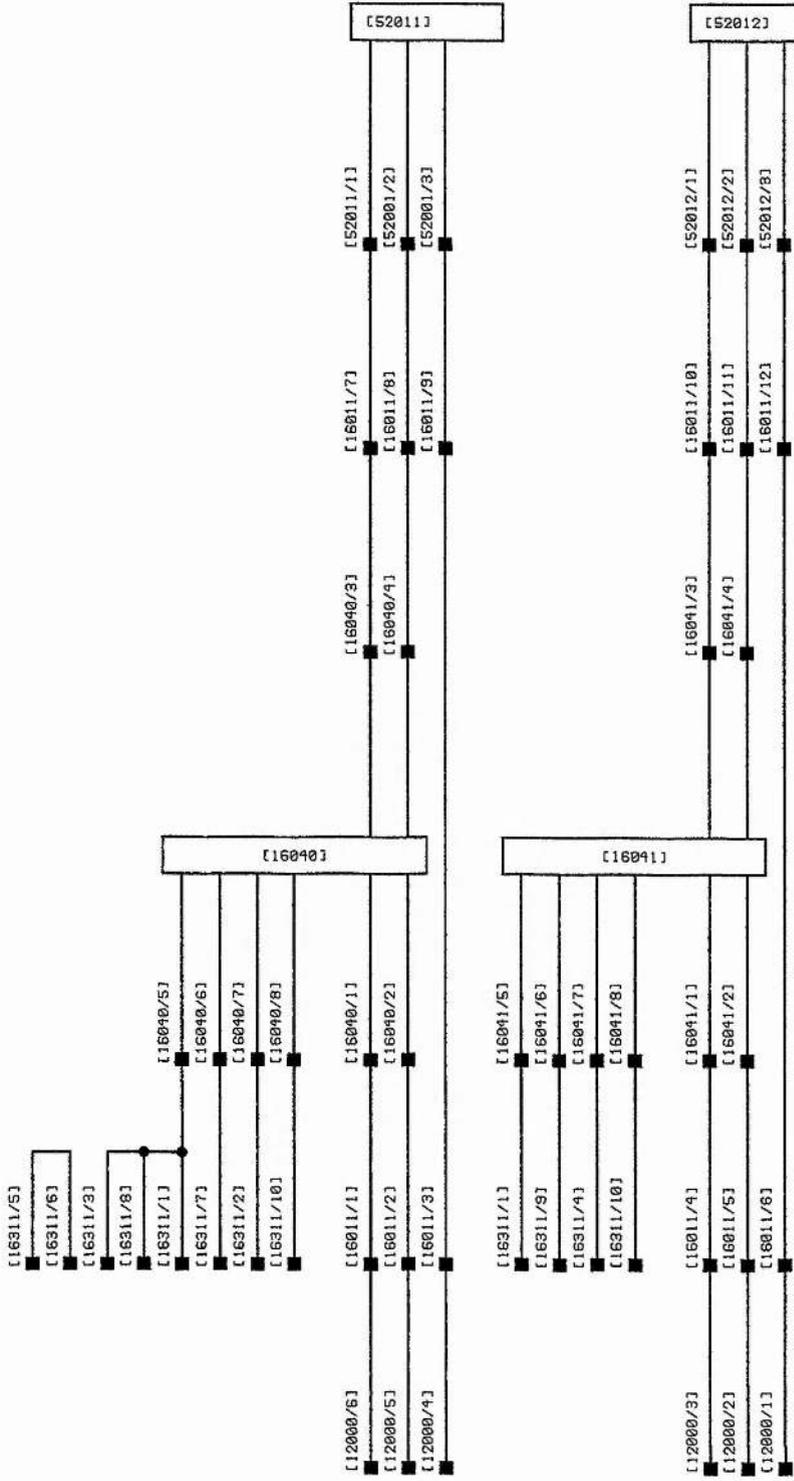
D3 SCH

# F.4 : R.A. slew and R.A. clamp switchgear



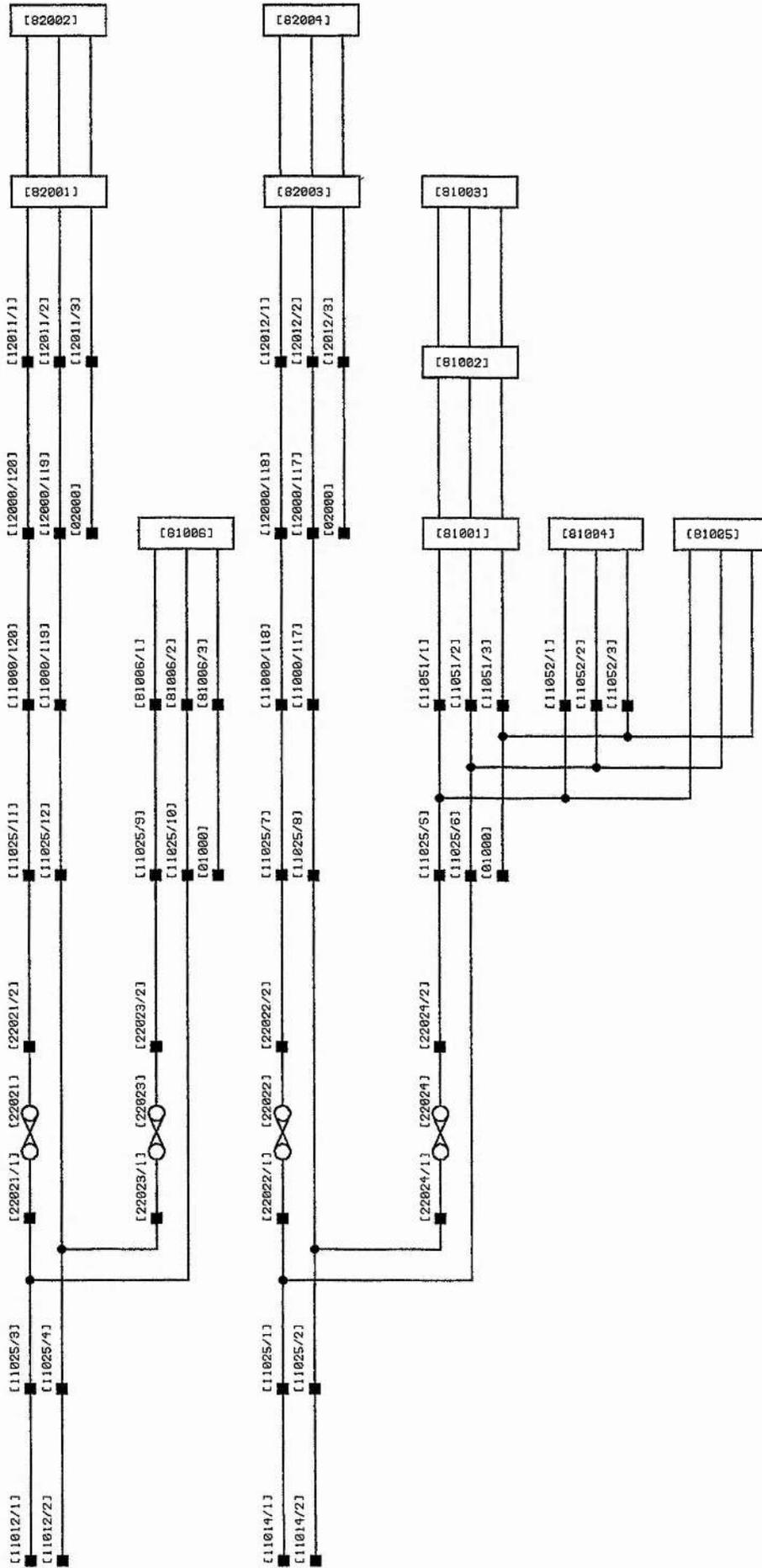
D38 SCH

## F.5: DEC. slew and DEC. clamp switchgear



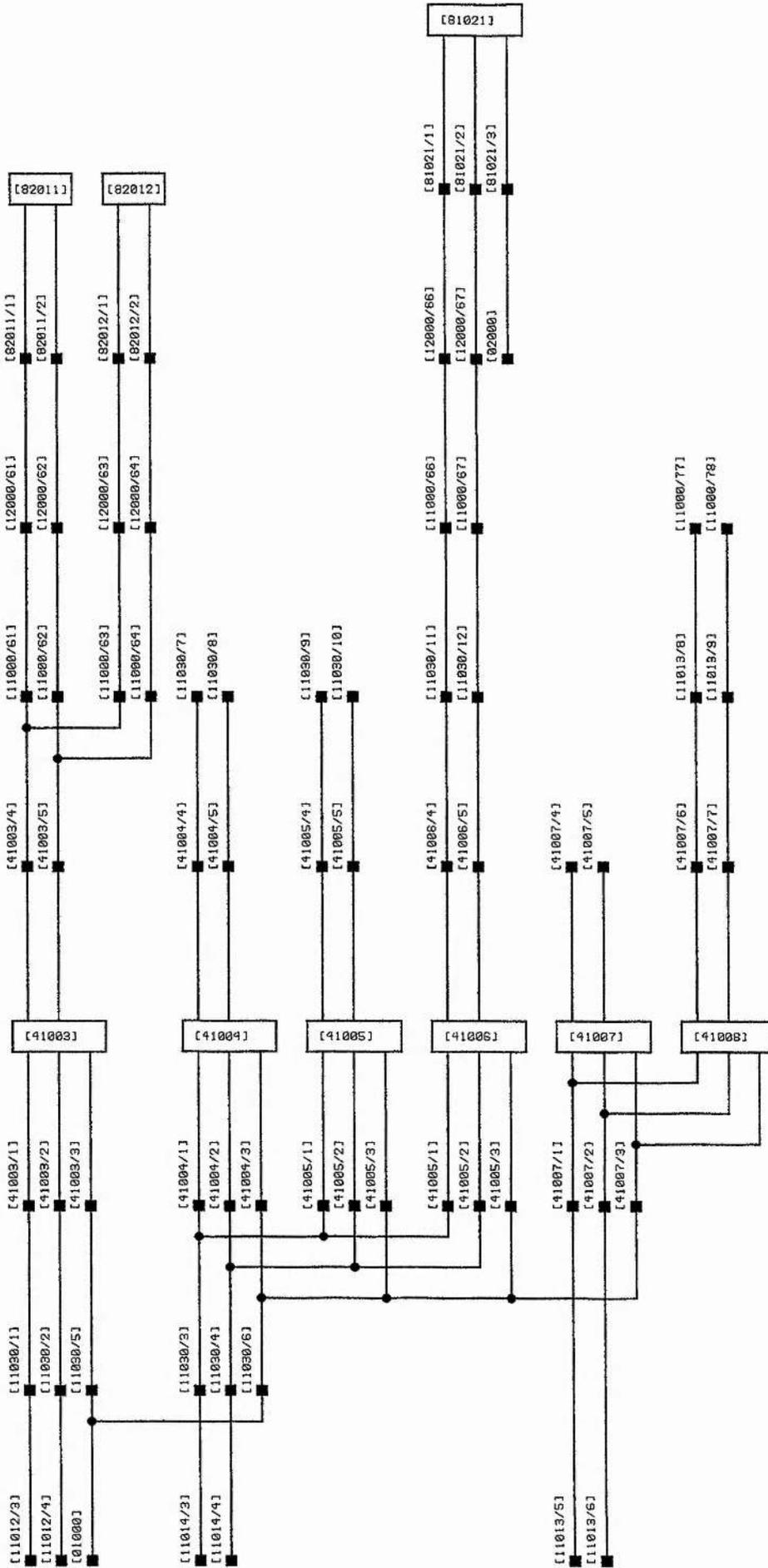
037 SCH

# F.6 : 1 Phase supply distribution



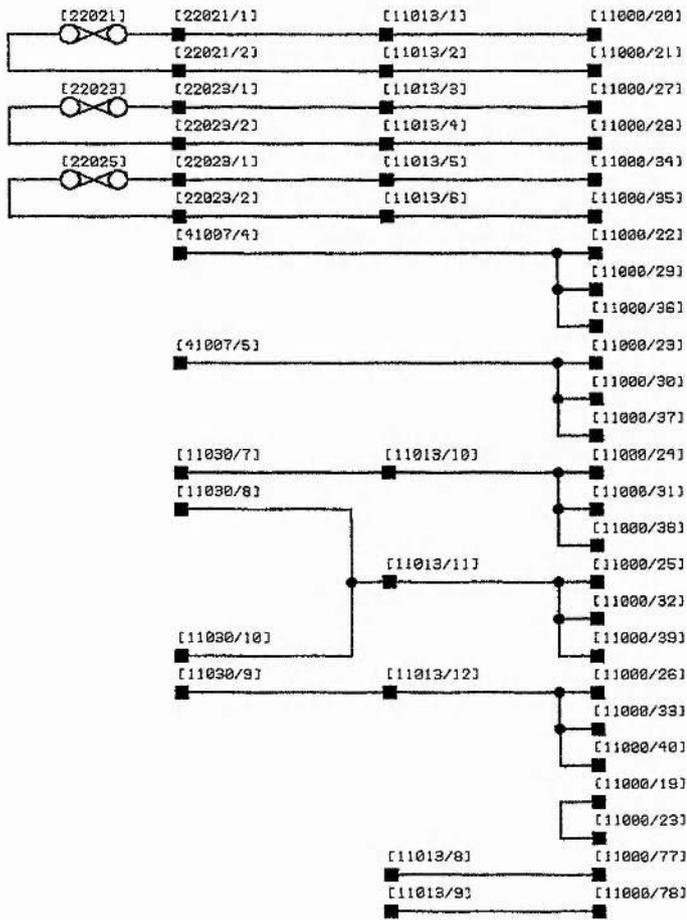
D9 SCH

## F.7 : Low voltage transformers and distribution



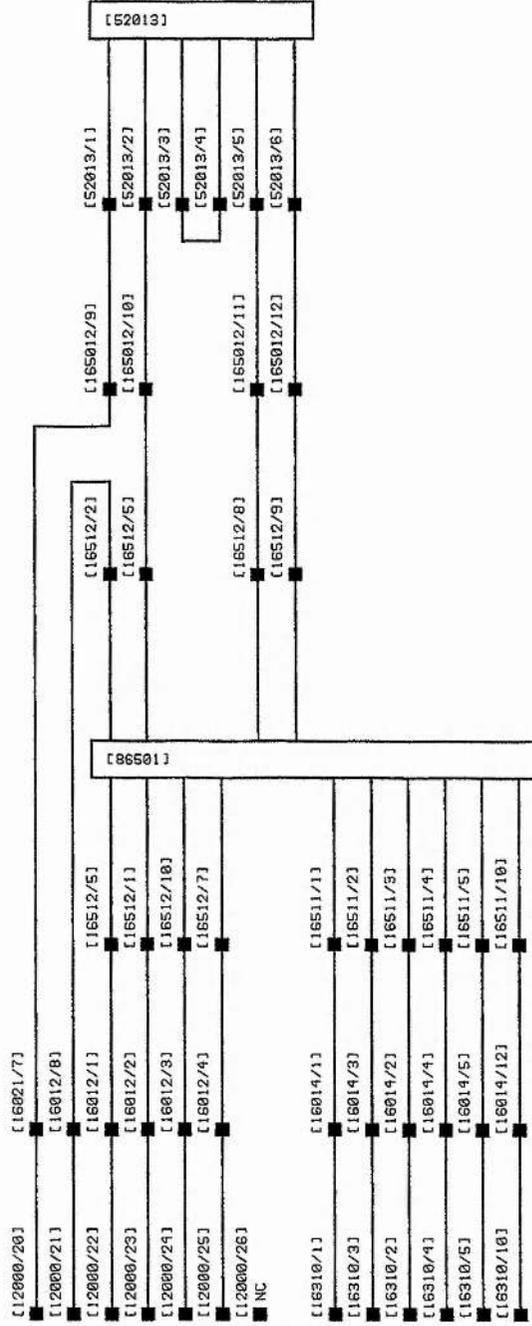
05 SCH

## F.8 : Low voltage spine distribution



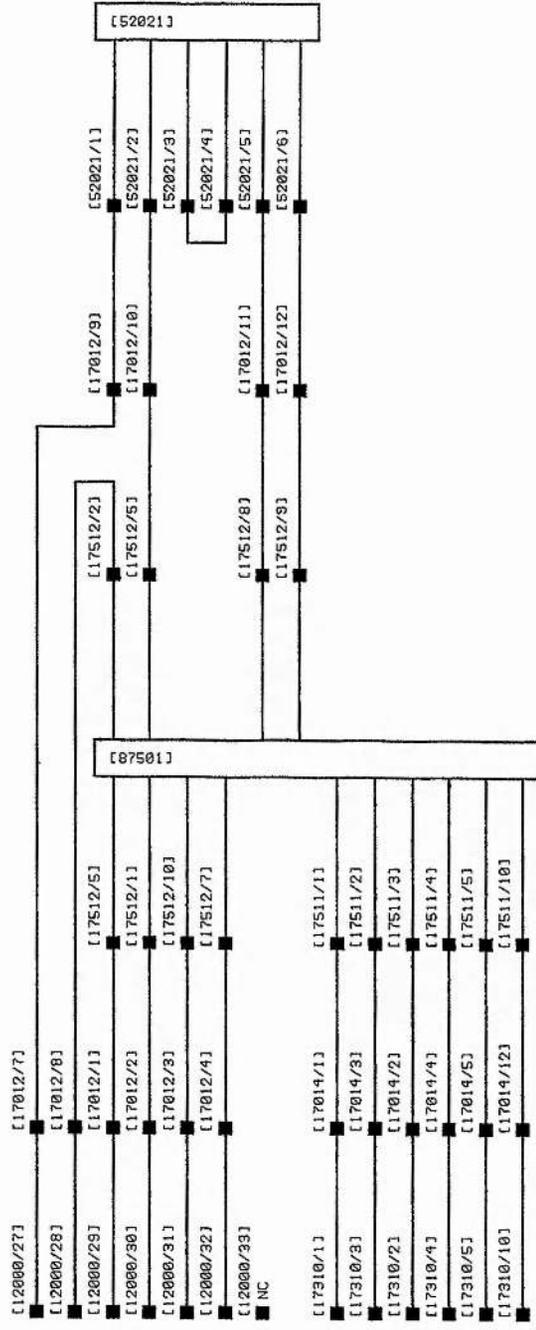
D6. SCH

## F.9 : Reference DEC. fine motor



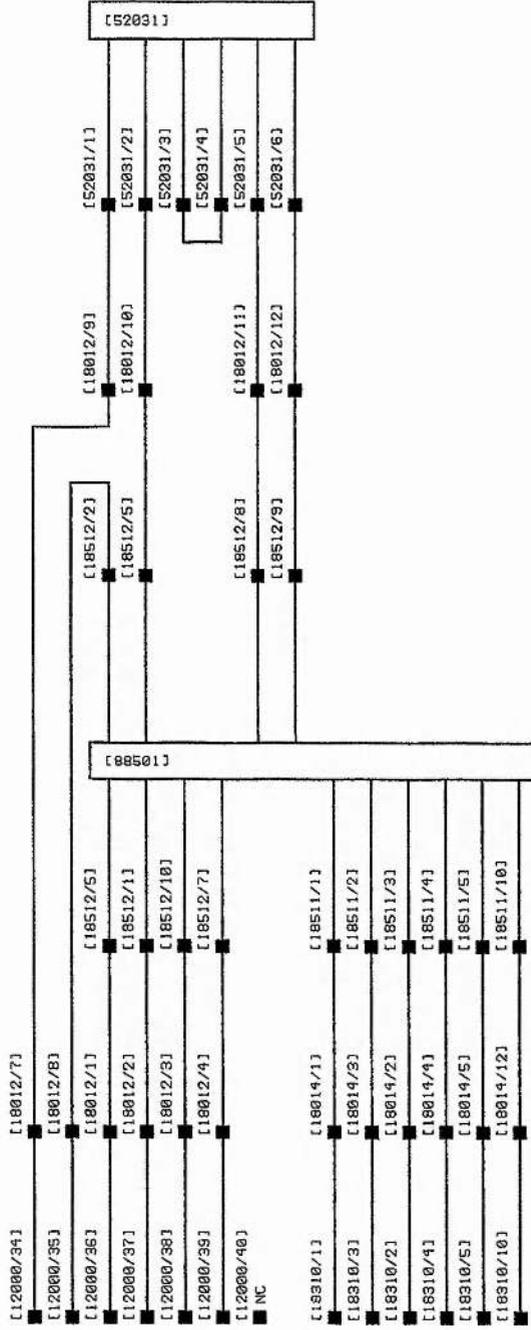
DT SCH

# F.10 : Offset R.A. fine motor



D8 SCH

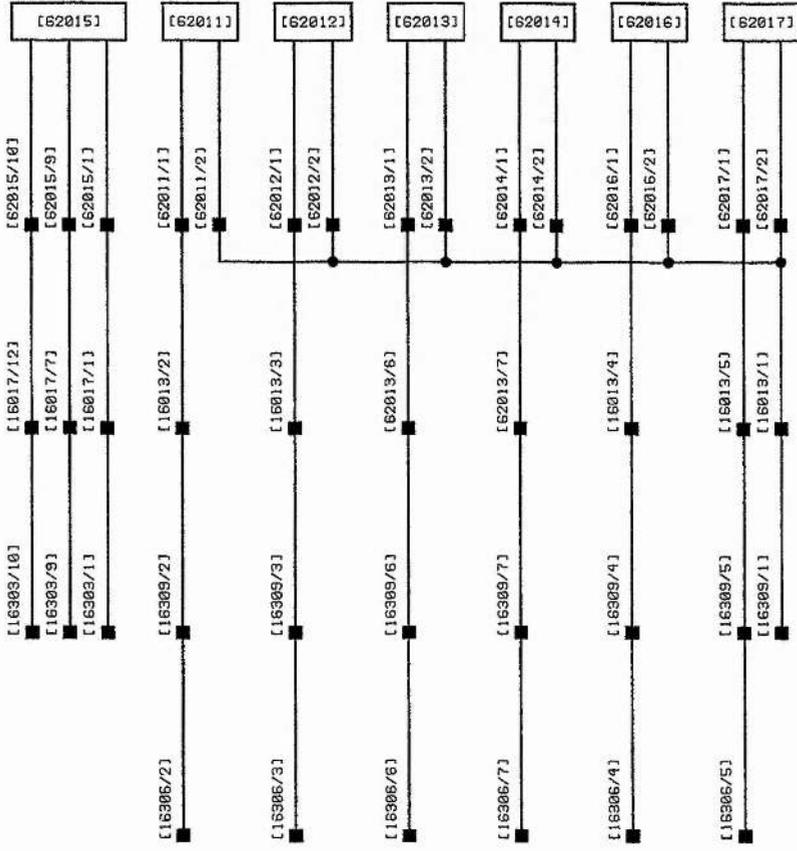
## F.11 : Offset DEC. fine motor



DS SCH

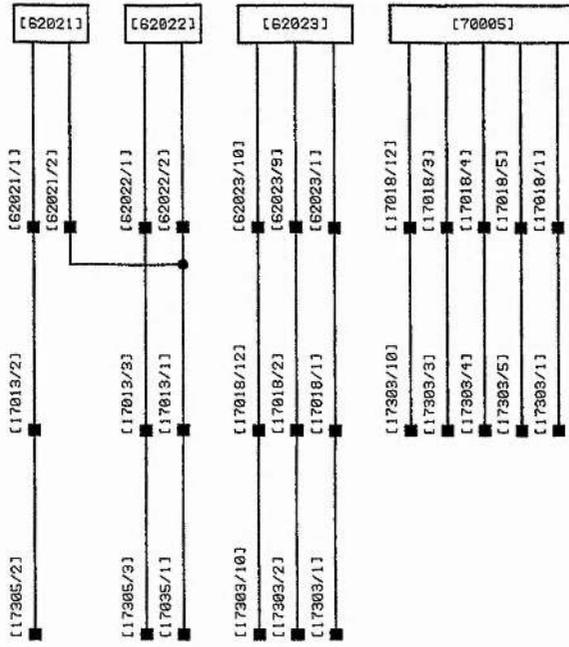


## F.13 : Reference DEC. limits



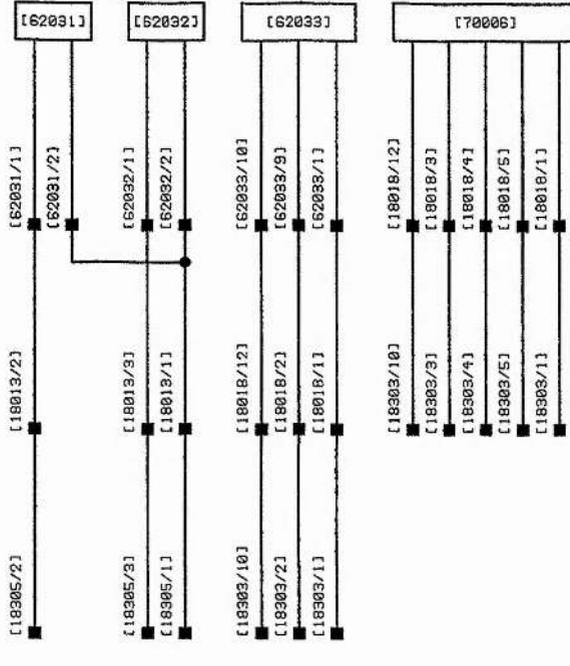
D11 SCH

## F.14 : Offset R.A. limits and encoder



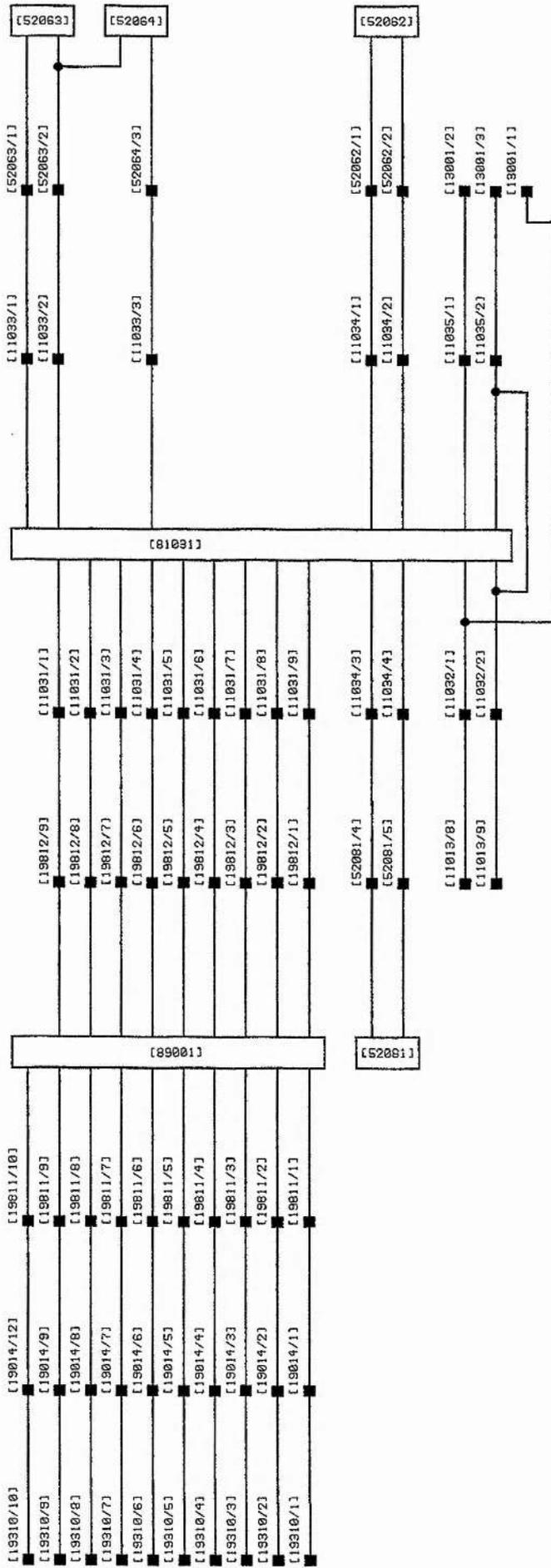
012 SCH

## F.15 : Offset DEC. limits and encoder



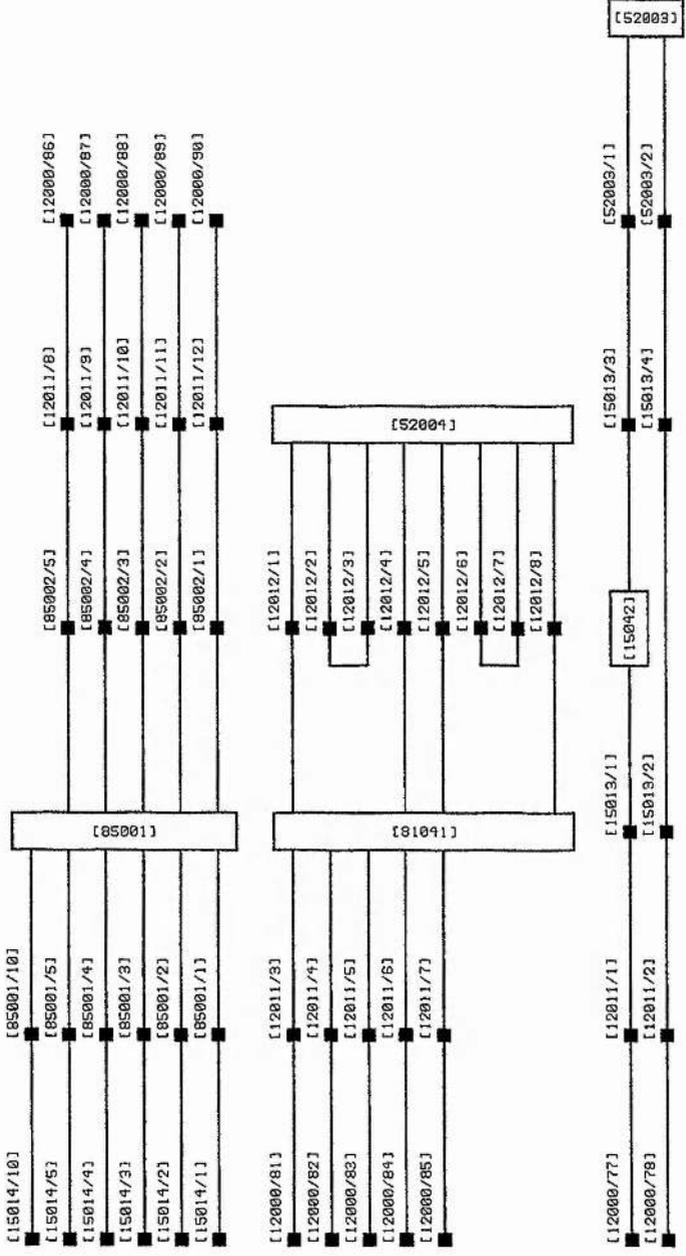
D13 SCH

# F.16 : Dome rotation opto isolator and controller



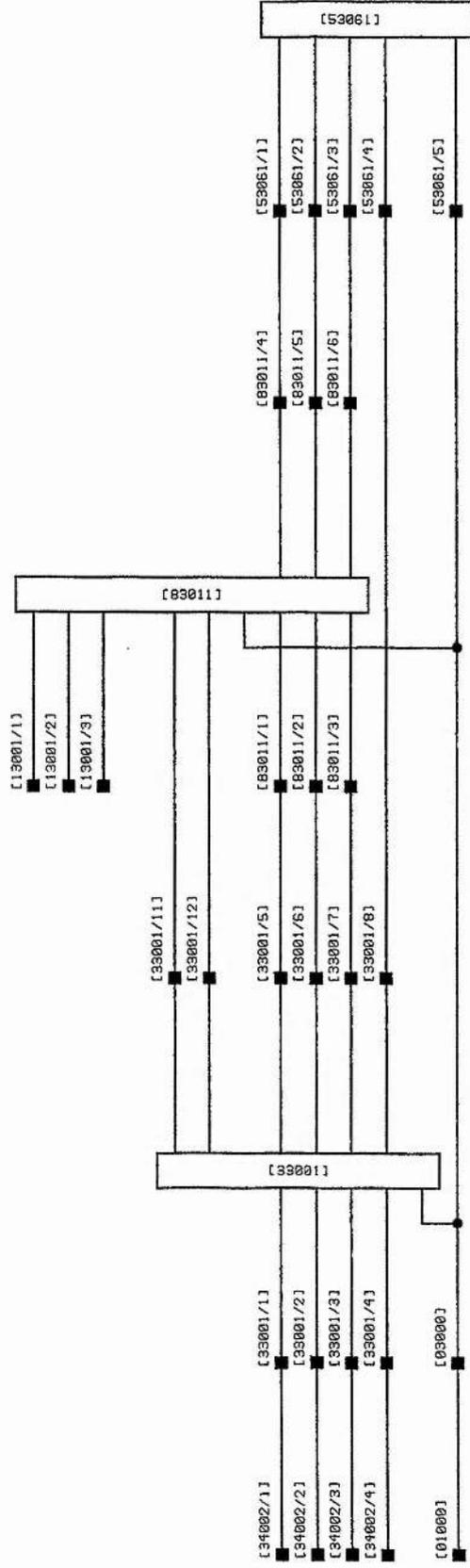
D14 SCH

**F.17 : Sidereal motor driver, R.A. sidereal opto isolator, driver and R.A. clutch**



D15 SCH

**F.18 : Hydraulic pump 3 phase supply and switchgear**



016 SCH

## F.19 : Reference R.A. coarse and fine encoders

[70002/1]	[15017/2]	[15303/6]
[70002/2]	[15016/11]	[15303/5]
[70002/3]	[15016/10]	[15303/4]
[70002/4]	[15016/9]	[15303/3]
[70002/5]	[15016/8]	[15303/2]
[70002/6]	[15016/7]	[15302/7]
[70002/7]	[15016/6]	[15302/6]
[70002/8]	[15016/5]	[15302/5]
[70002/9]	[15016/4]	[15302/4]
[70002/10]	[15016/3]	[15302/3]
[70002/11]	[15016/2]	[15302/2]
[70002/15]	[15016/1]	[15302/1]
[70002/16]	[15016/1]	[15302/1]
[70002/18]	[15016/12]	[15302/10]
[70002/23]	[15016/1]	[15302/1]
[70002/24]	[15016/1]	[15302/1]
[70002/25]	[15016/1]	[15302/1]

017 SCH

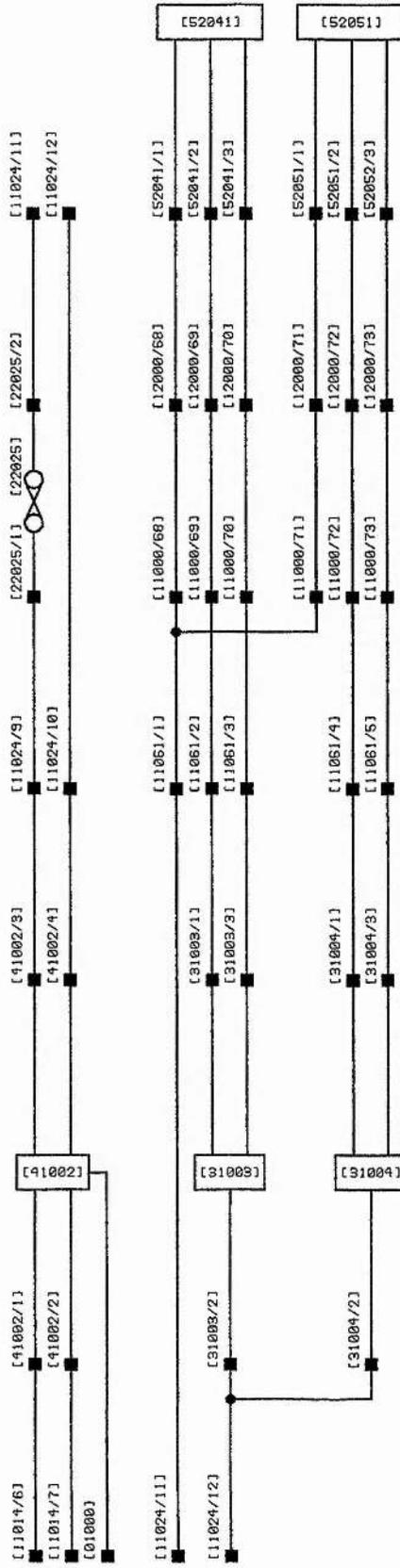
[70001/1]	[15018/2]	[15305/2]
[70001/2]	[15018/3]	[15305/3]
[70001/3]	[15018/4]	[15305/4]
[70001/4]	[15018/5]	[15305/5]
[70001/5]	[15018/6]	[15305/6]
[70001/22]	[15018/12]	[15305/10]
[70001/23]	[15018/12]	[15305/10]
[70001/24]	[15018/11]	[15305/11]
[70001/25]	[15018/11]	[15305/11]

## F.20 : Reference DEC. coarse and fine encoders

[70004/1]	[16017/2]	[16305/6]	[70003/1]	[16018/2]	[16302/2]
[70004/2]	[16016/11]	[16305/5]	[70003/2]	[16018/3]	[16302/3]
[70004/3]	[16016/10]	[16305/4]	[70003/3]	[16018/4]	[16302/4]
[70004/4]	[16016/9]	[16305/3]	[70003/4]	[16018/5]	[16302/5]
[70004/5]	[16016/8]	[16305/2]	[70003/5]	[16018/6]	[16302/6]
[70004/6]	[16016/7]	[16304/7]	[70003/6]	[16018/7]	[16302/7]
[70004/7]	[16016/6]	[16304/6]	[70003/22]	[16018/12]	[16302/10]
[70004/8]	[16016/5]	[16304/5]	[70003/23]	[16018/12]	[16302/10]
[70004/9]	[16016/4]	[16304/4]	[70003/24]	[16018/11]	[16302/11]
[70004/10]	[16016/3]	[16304/3]	[70003/25]	[16018/11]	[16302/11]
[70004/11]	[16016/2]	[16304/2]			
[70004/15]	[16016/1]	[16304/1]			
[70004/16]	[16016/1]	[16304/1]			
[70004/18]	[16016/12]	[16304/10]			
[70004/23]	[16016/1]	[16304/1]			
[70004/24]	[16016/1]	[16304/1]			
[70004/25]	[16016/1]	[16304/1]			

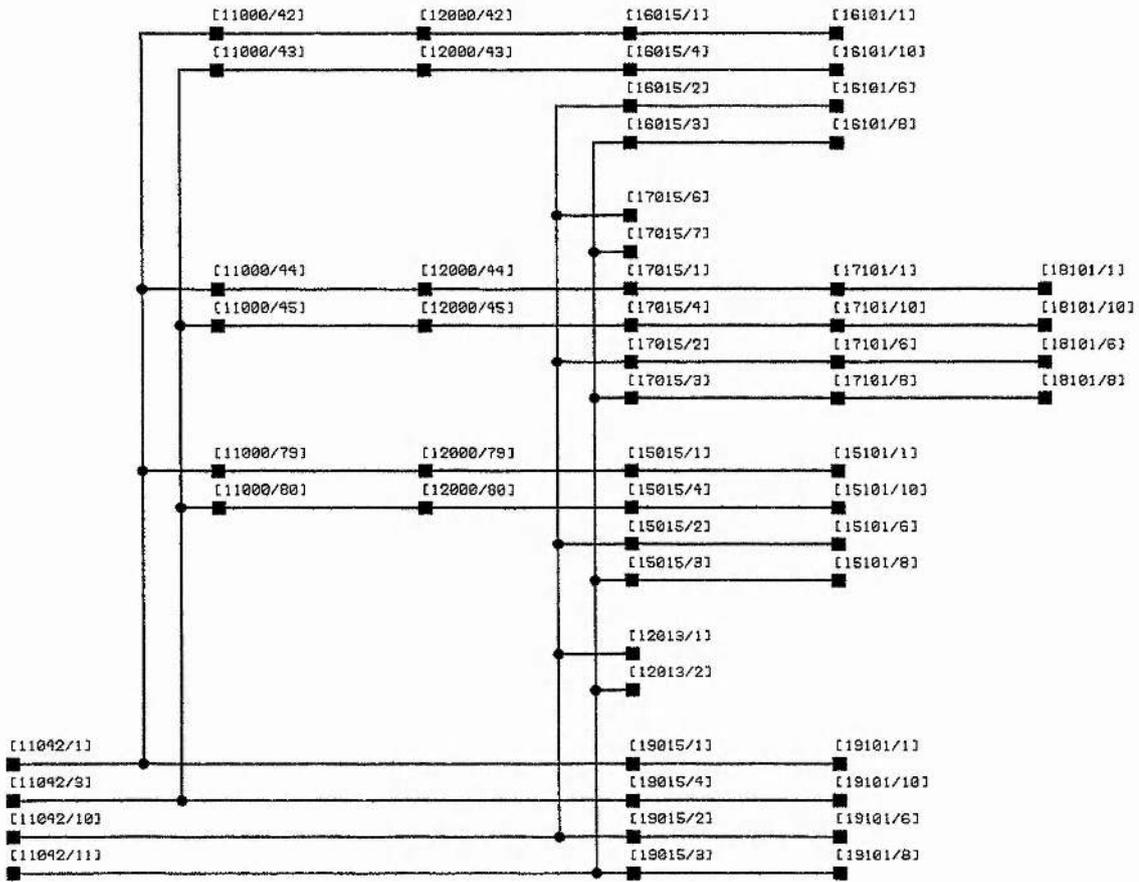
D18. SCH

## F.21 : Reference and offset focus motors and switchgear



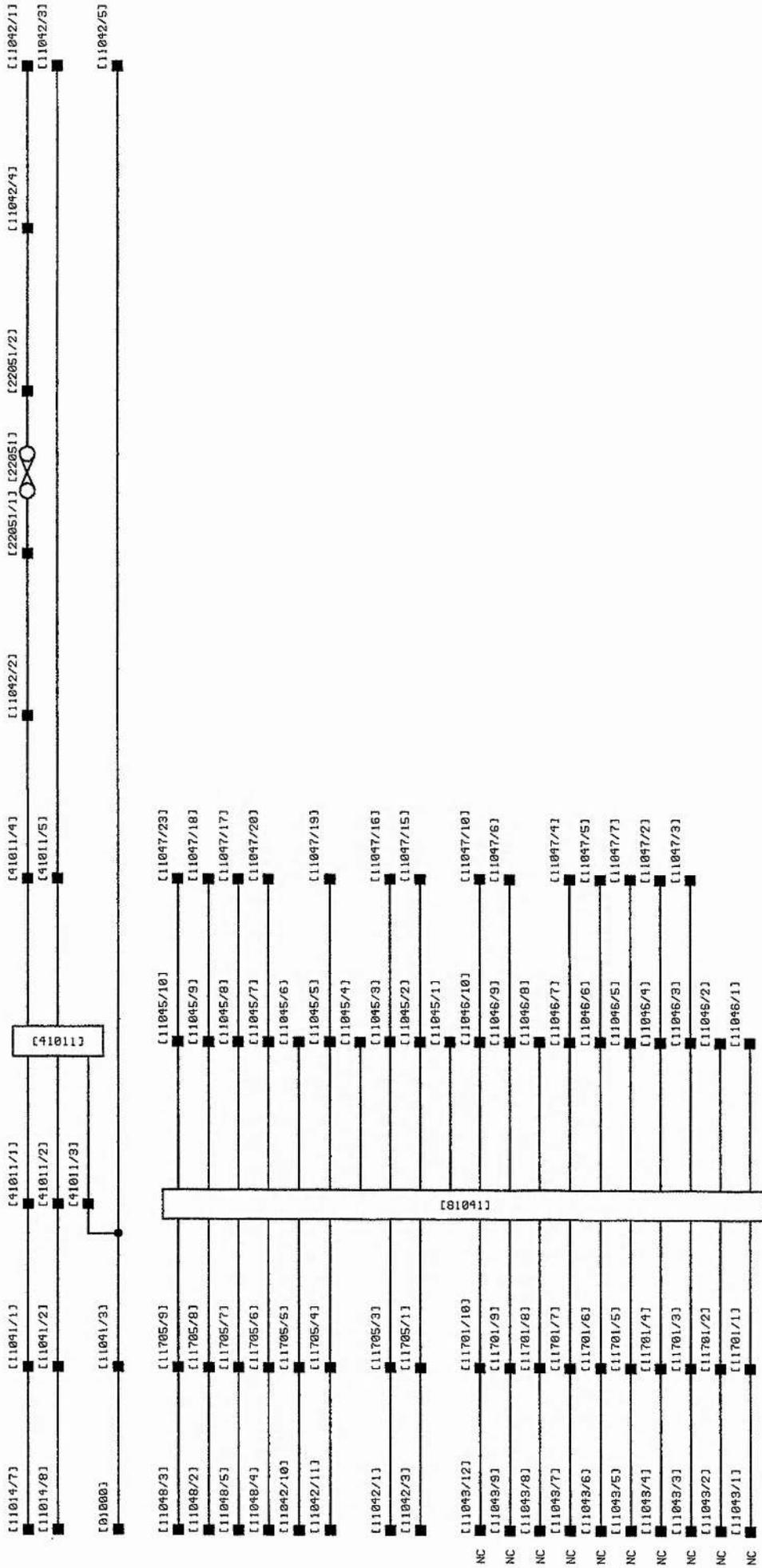
D19 SCH

## F.22 : Microcontroller power and communications

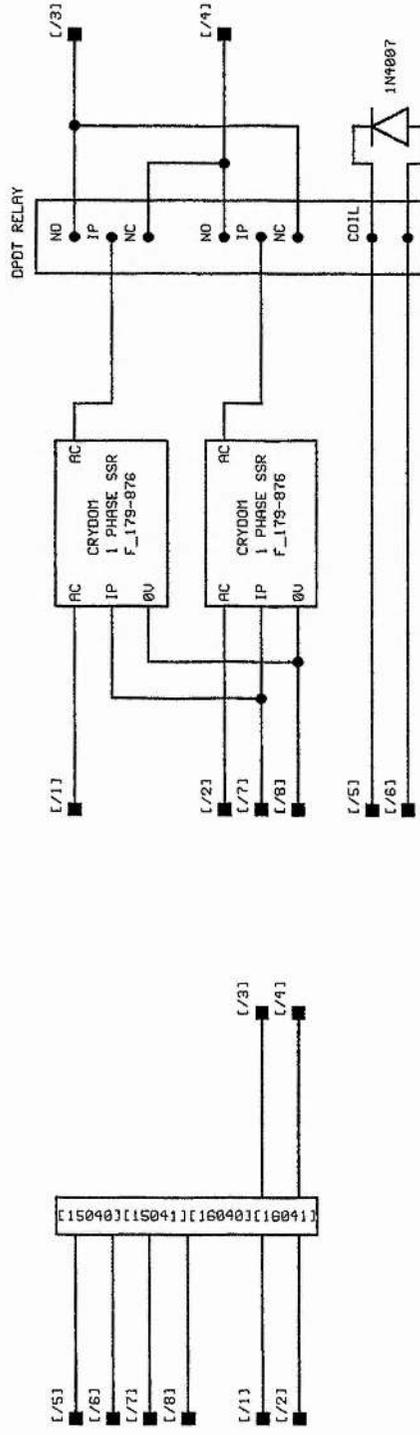


D20 SCH

# F.23 : Microcontroller power supply and monitor display

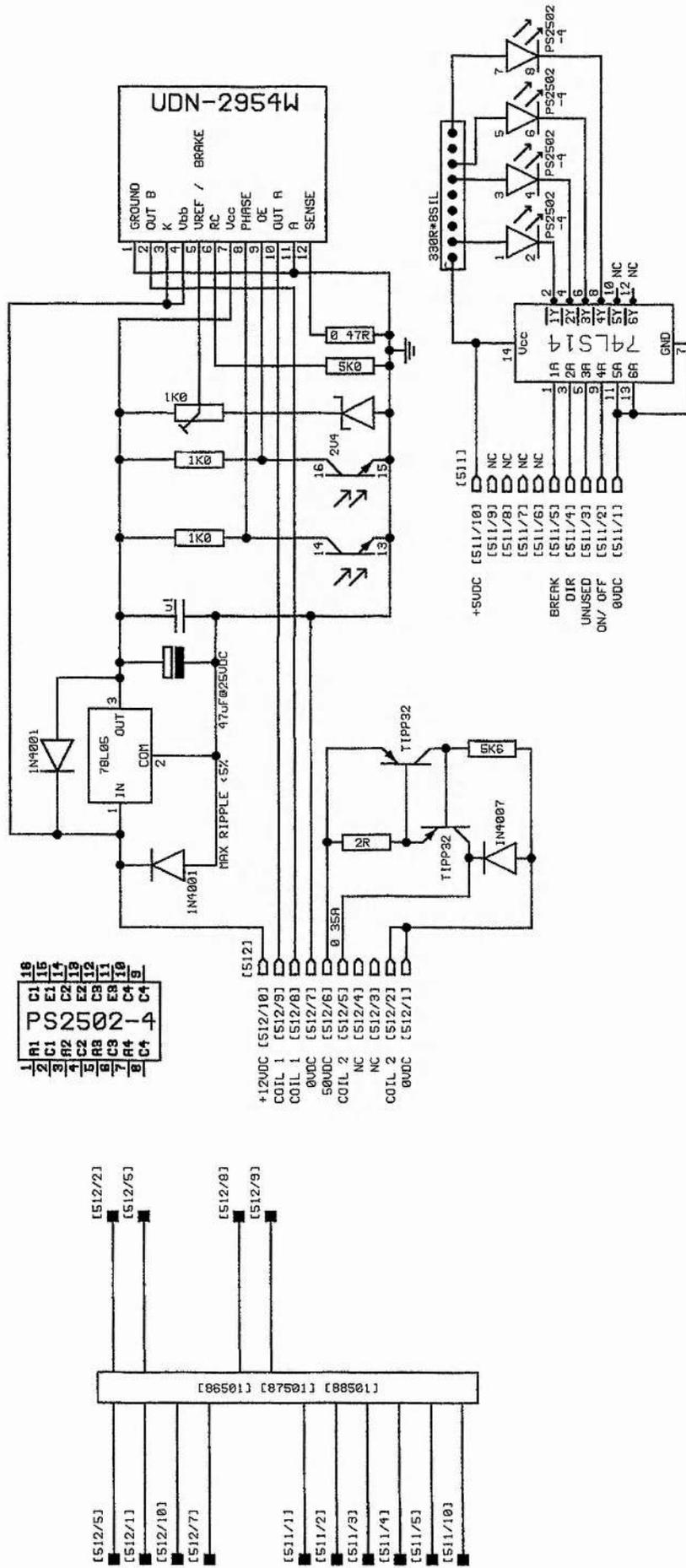


## F.24 : 3 Phase solid state relay unit



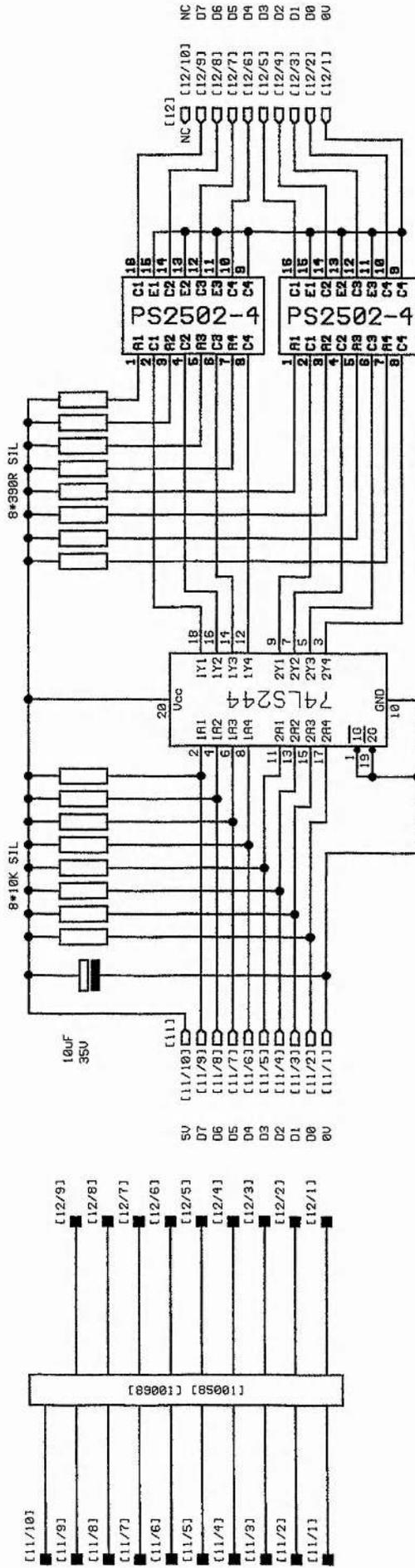
D22 SCH

## F.25 : Pulse width modulated motor driver

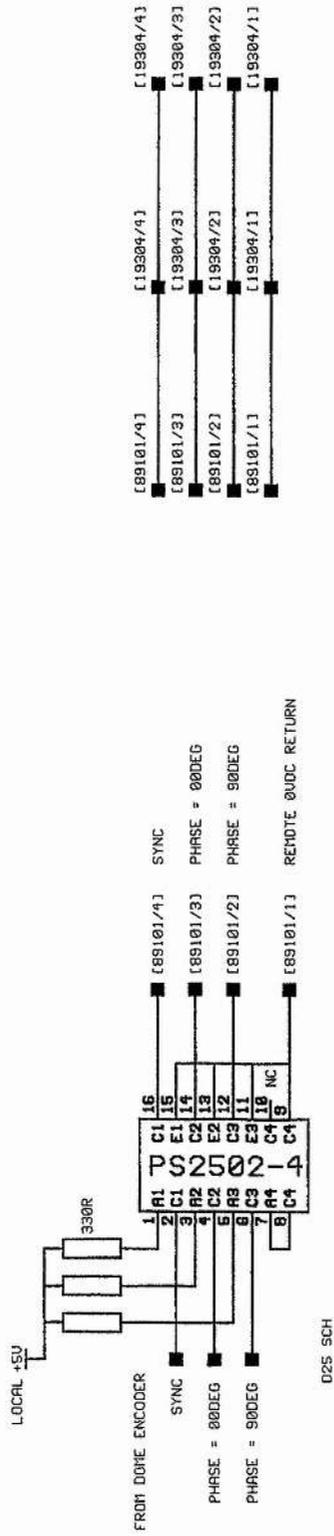


D23 SCH

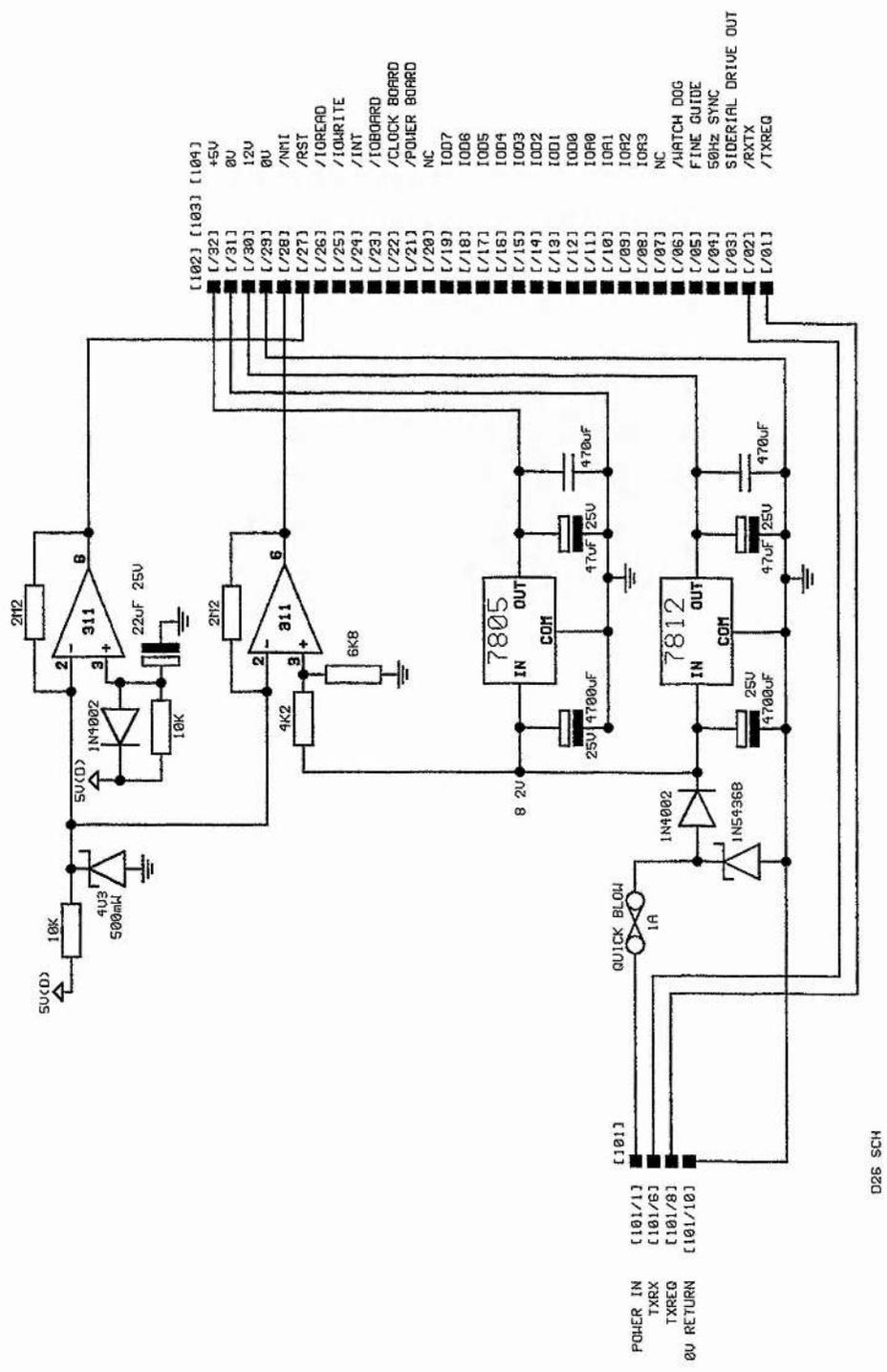
## F.26 : Sidereal and dome opto isolator



## F.27 : Dome encoder isolator

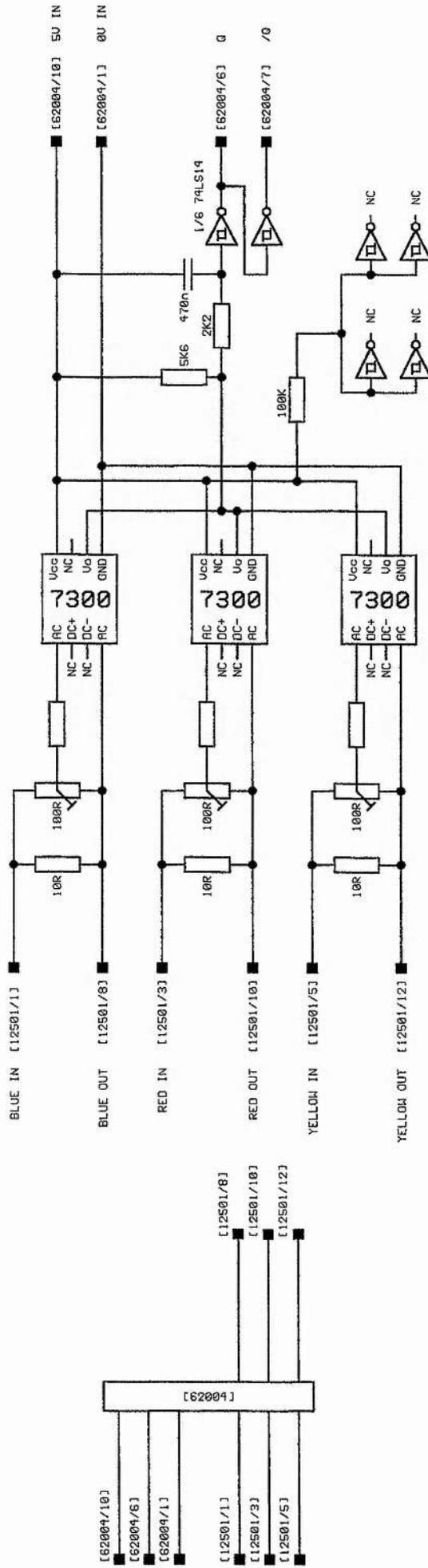


# F.28 : CPU power board



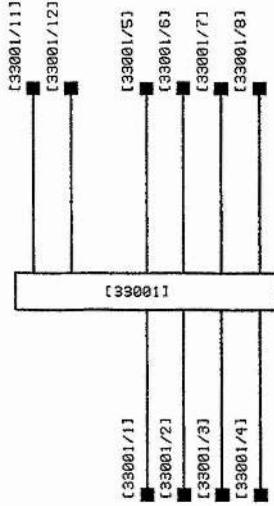
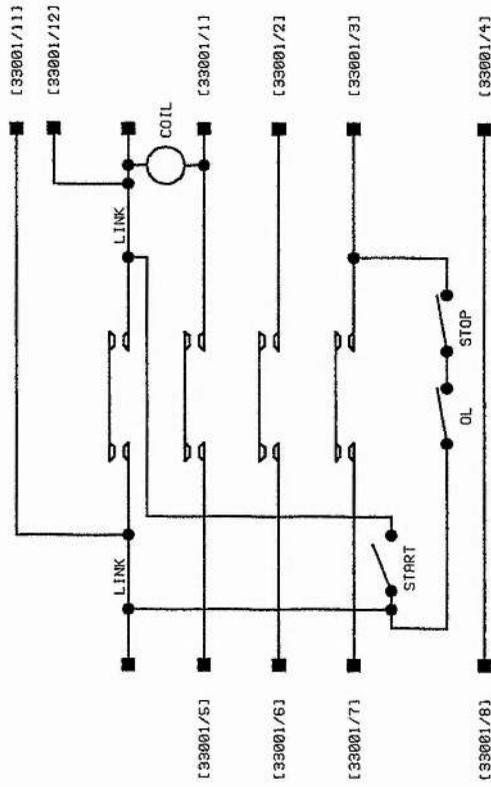
D26 SCH

## F.29 : 3 Phase clamp overload sensor



D27 SCH

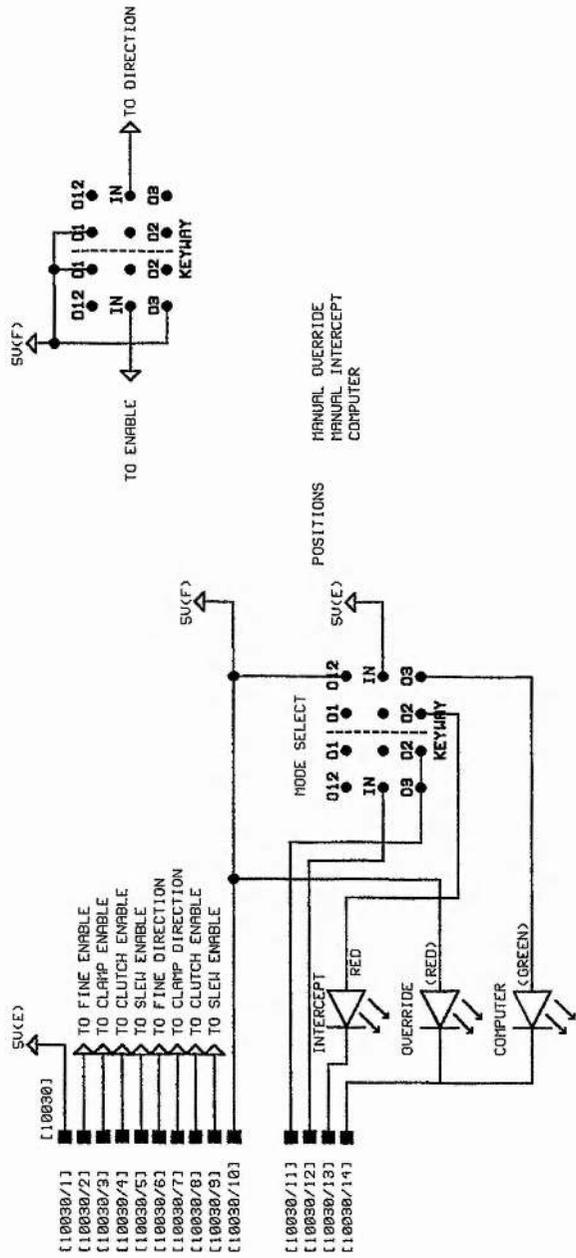
## F.30 : 3 Phase mechanical pump contactor



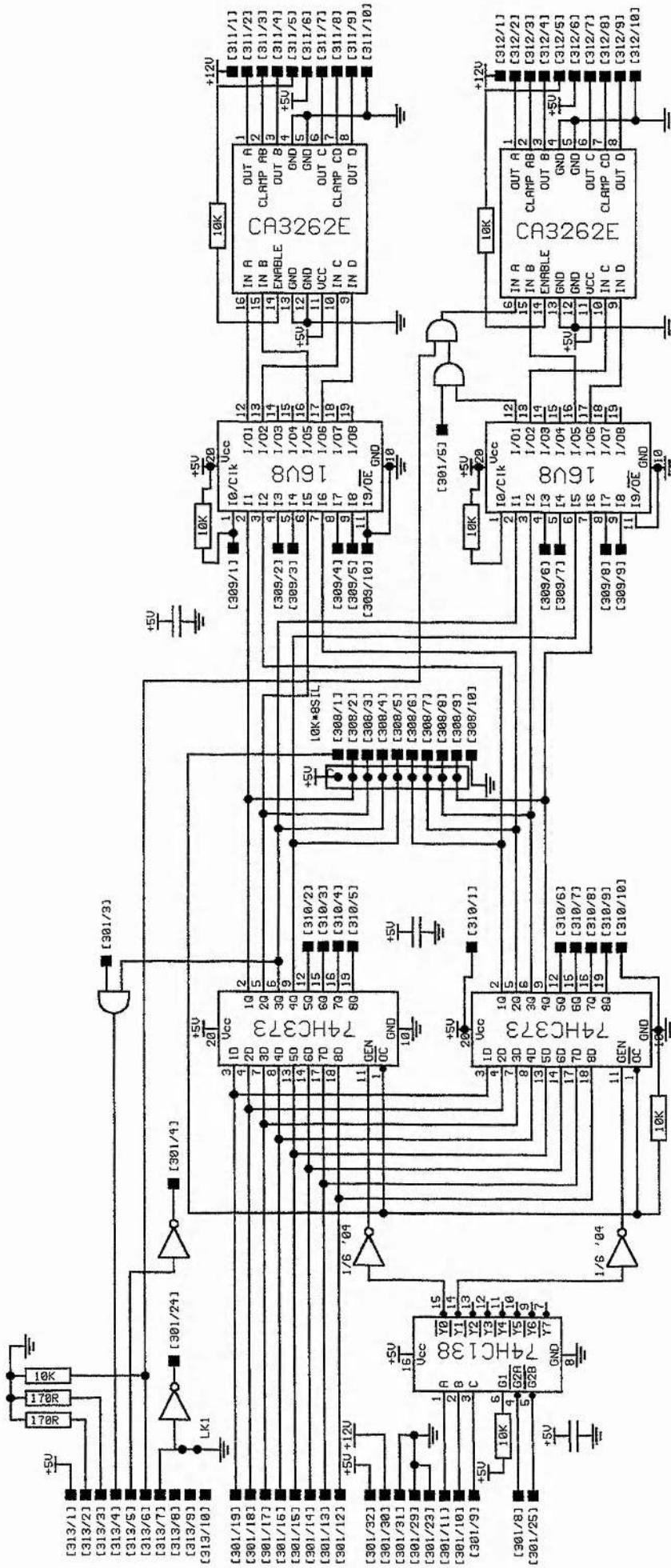
D28. SCH



## F.32 : Manual switch unit

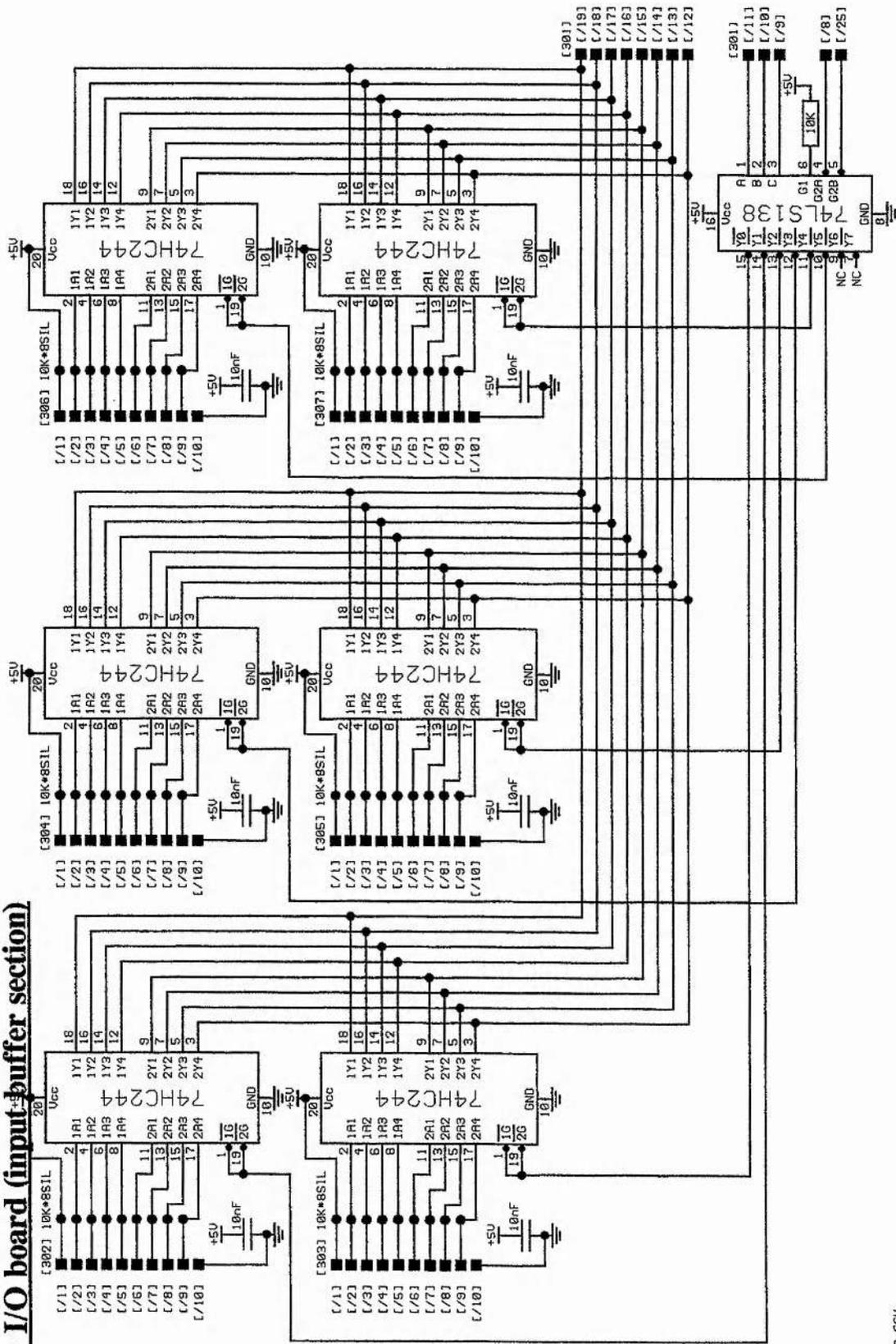


### F.33 : I/O board (output driver section)



031 SCH

### F.34 : I/O board (input-buffer section)

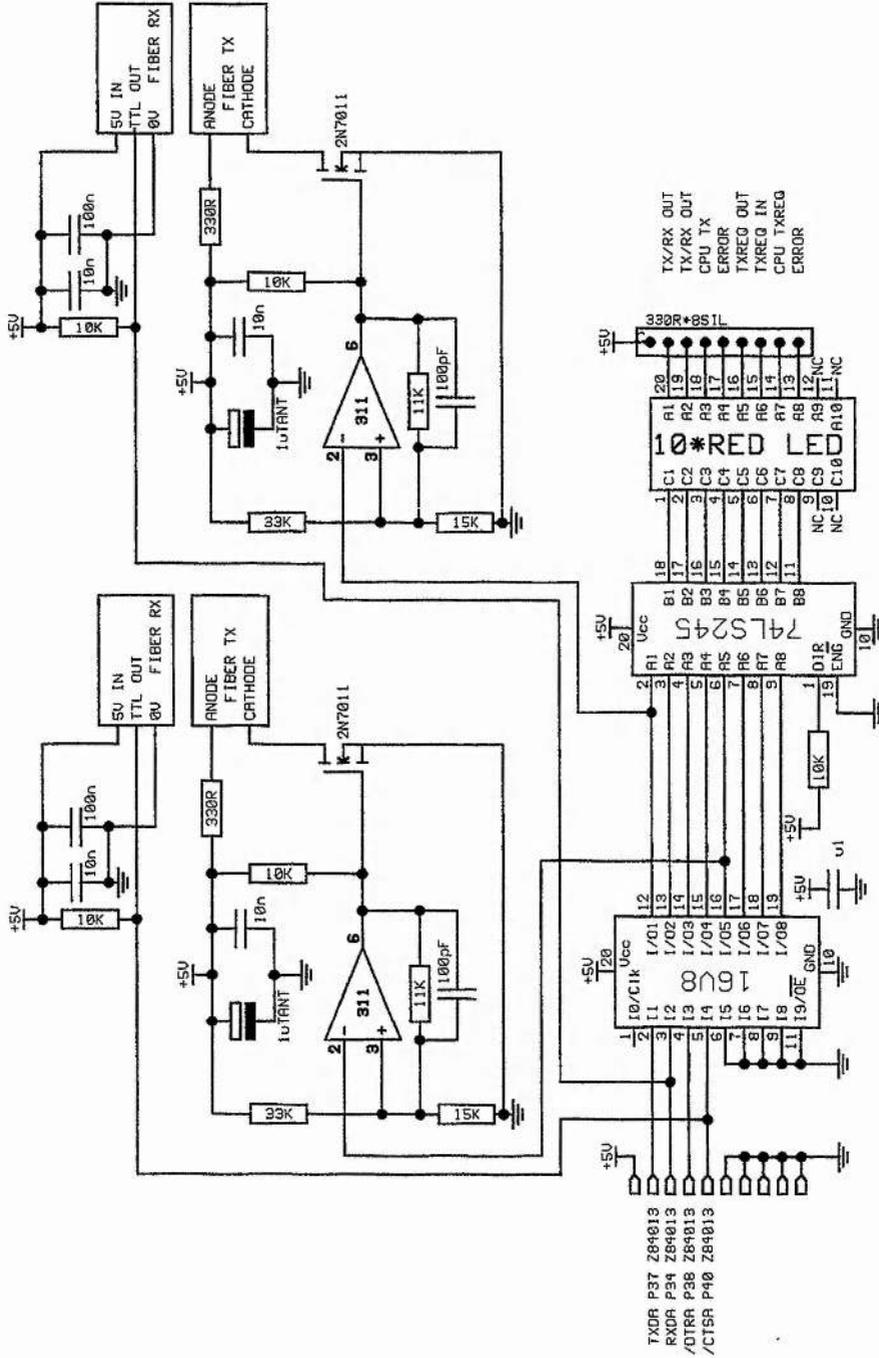


D32 SCH



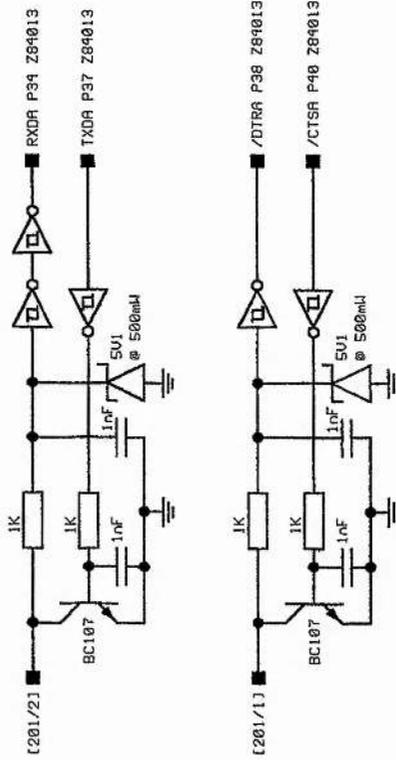


# F.37 : Proposed fibre optic transceiver unit



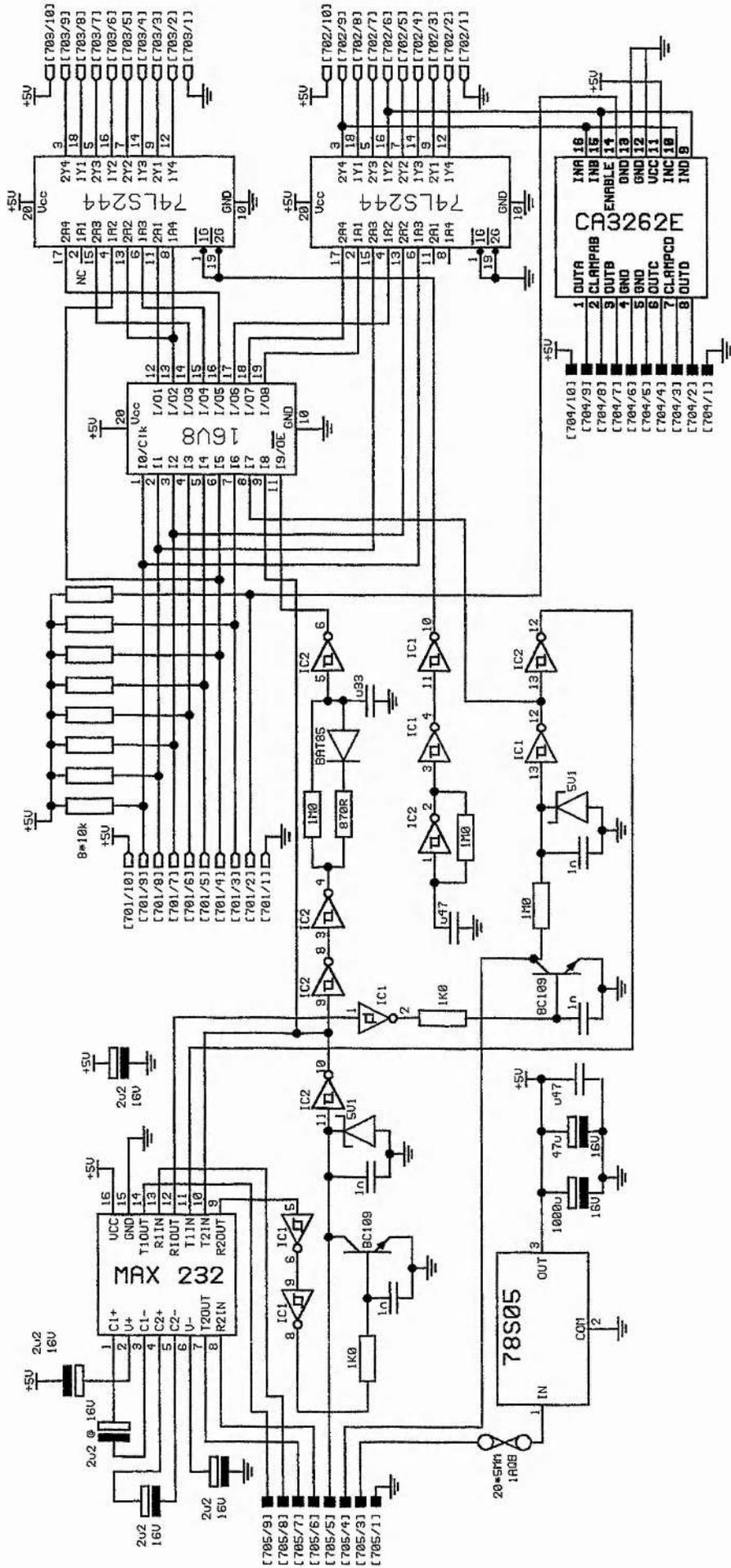
036 SCH

**F.38 : Transceiver used**



035 SCH

# F.39 : Network controller



DS9 SCH