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.
Contents

Table of 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 ........................................................................  84
4.8 : Move monitor : movemon ...........................................................................  84

A distributed control system for the St Andrews twin photometric telescope : Page 2 of 335
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: TCLLINK 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: TCLINK title text .......................................................................... 108
Table 16: TCLINK 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 11th 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 11th 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 11th 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 ..............................................

---

A distributed control system for the St Andrews twin photometric telescope : Page 9 of 335
<table>
<thead>
<tr>
<th><strong>Glossary</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ATOM</strong></td>
<td>A 16 bit integer array whose entries points to character strings</td>
</tr>
<tr>
<td><strong>ATIS</strong></td>
<td>Automatic Telescope Instruction Set</td>
</tr>
<tr>
<td><strong>BUS</strong></td>
<td>A generic group of signals connected between two or more structures via a common physical route</td>
</tr>
<tr>
<td><strong>C700</strong></td>
<td>C compiler version 7.00. This is the first Microsoft compiler to support the Windows 3.0 / 3.1 operating system</td>
</tr>
<tr>
<td><strong>C</strong></td>
<td>Programming language used to create Windows modules</td>
</tr>
<tr>
<td><strong>C++</strong></td>
<td>Object oriented version of C</td>
</tr>
<tr>
<td><strong>CCW</strong></td>
<td>Counter clockwise rotation</td>
</tr>
<tr>
<td><strong>CFX</strong></td>
<td>Control file executor</td>
</tr>
<tr>
<td><strong>CLK</strong></td>
<td>Clock</td>
</tr>
<tr>
<td><strong>CPU</strong></td>
<td>Central processing unit, e.g. microcontroller IC</td>
</tr>
<tr>
<td><strong>CR</strong></td>
<td>Carriage return (0x0D) or capacitor resistor timing circuit</td>
</tr>
<tr>
<td><strong>CRC</strong></td>
<td>Cyclic redundancy check. A polynomial generated by the SIO to enabling error checking</td>
</tr>
<tr>
<td><strong>CTS</strong></td>
<td>RS232 Protocol: clear to send</td>
</tr>
<tr>
<td><strong>CW</strong></td>
<td>Clockwise rotation</td>
</tr>
<tr>
<td><strong>DEC</strong></td>
<td>Declination</td>
</tr>
<tr>
<td><strong>DCD</strong></td>
<td>RS232 Protocol: data carrier detect</td>
</tr>
<tr>
<td><strong>DCS</strong></td>
<td>Manufacturer of IBM PC clone used in TPT</td>
</tr>
<tr>
<td><strong>DDE</strong></td>
<td>Dynamic data exchange</td>
</tr>
<tr>
<td><strong>DOS</strong></td>
<td>Disk operating system. Standard text based operating system for IBM PC clones; current version is 6.0</td>
</tr>
<tr>
<td><strong>DTR</strong></td>
<td>RS232 Protocol: data terminal ready</td>
</tr>
<tr>
<td><strong>DX</strong></td>
<td>Non multiplexed data bus</td>
</tr>
<tr>
<td><strong>EPROM</strong></td>
<td>Erasable programmable read only memory. An IC that stores a program for use by a microcontroller</td>
</tr>
<tr>
<td><strong>FET</strong></td>
<td>Field effect transistor</td>
</tr>
<tr>
<td><strong>GAL</strong></td>
<td>Generic array logic device: An IC that can be programmed to emulate logic functions</td>
</tr>
<tr>
<td><strong>GND</strong></td>
<td>Local ground</td>
</tr>
<tr>
<td><strong>GUI</strong></td>
<td>Graphical user interface</td>
</tr>
<tr>
<td><strong>IBM</strong></td>
<td>International business machines. Manufacturer and standard specifier of PC computers</td>
</tr>
<tr>
<td><strong>IC</strong></td>
<td>Integrated circuit</td>
</tr>
<tr>
<td><strong>IDC</strong></td>
<td>Indirect connector</td>
</tr>
<tr>
<td><strong>INT</strong></td>
<td>Interrupt, usually related to a physical line which interrupts the CPU</td>
</tr>
<tr>
<td><strong>INTEL</strong></td>
<td>Manufacturer of 80x86 processor ICs</td>
</tr>
<tr>
<td><strong>I/O</strong></td>
<td>Input / output</td>
</tr>
<tr>
<td><strong>JEDEC</strong></td>
<td>Jedec code (kernel format): used in programming GAL devices</td>
</tr>
<tr>
<td><strong>LAN</strong></td>
<td>Local area network</td>
</tr>
<tr>
<td><strong>LED</strong></td>
<td>Light emitting diode</td>
</tr>
<tr>
<td><strong>LSB</strong></td>
<td>Least significant bit</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>MAX386</td>
<td>A memory management driver from Qualitas required by Microsoft’s C700 compiler</td>
</tr>
<tr>
<td>MB</td>
<td>Mega byte or mega baud</td>
</tr>
<tr>
<td>MSB</td>
<td>Most significant bit</td>
</tr>
<tr>
<td>NTT</td>
<td>New technology telescope</td>
</tr>
<tr>
<td>nod</td>
<td>Link switch in Microsoft’s C700 compiler that ignores default libraries named in object files</td>
</tr>
<tr>
<td>OLE</td>
<td>Object linking and embedding</td>
</tr>
<tr>
<td>PC</td>
<td>Personal computer, generic name for IBM type clones</td>
</tr>
<tr>
<td>PCB</td>
<td>Printed circuit board</td>
</tr>
<tr>
<td>PIO</td>
<td>Parallel input / output unit</td>
</tr>
<tr>
<td>PLC</td>
<td>Process logic controller</td>
</tr>
<tr>
<td>PWB</td>
<td>Microsoft’s integrated environment for the C700 compiler</td>
</tr>
<tr>
<td>R.A.</td>
<td>Right ascension</td>
</tr>
<tr>
<td>RAM</td>
<td>Random access memory</td>
</tr>
<tr>
<td>RDU</td>
<td>Remote driver unit. Used to describe a microcontroller unit</td>
</tr>
<tr>
<td>RFI</td>
<td>Radio Frequency Interference</td>
</tr>
<tr>
<td>RISC</td>
<td>Reduced instruction set computer</td>
</tr>
<tr>
<td>RST</td>
<td>Reset, usually applied to the physical line to reset a CPU</td>
</tr>
<tr>
<td>RX</td>
<td>Receive</td>
</tr>
<tr>
<td>SCMA</td>
<td>Smart communications macro assembler</td>
</tr>
<tr>
<td>SCSI</td>
<td>Small computer systems interface</td>
</tr>
<tr>
<td>SDL C</td>
<td>Synchronous data link control. An IBM propriety packet protocol</td>
</tr>
<tr>
<td>SIO</td>
<td>Serial input output unit. This allows the computer communicate to the outside world, usually using RS232 protocol</td>
</tr>
<tr>
<td>SID</td>
<td>Short for sidereal</td>
</tr>
<tr>
<td>SMART</td>
<td>Manufacturer of test equipment</td>
</tr>
<tr>
<td>STACK</td>
<td>A type of data storage, usually very fast and memory efficient</td>
</tr>
<tr>
<td>SVGA</td>
<td>Super versatile graphics adapter (1024x760 pixels, 256 plane)</td>
</tr>
<tr>
<td>SX</td>
<td>Multiplexed data bus</td>
</tr>
<tr>
<td>SYNC</td>
<td>A timing pulse to synchronise data transfer</td>
</tr>
<tr>
<td>TCL</td>
<td>Telescope control language</td>
</tr>
<tr>
<td>TI</td>
<td>Texas instruments</td>
</tr>
<tr>
<td>TPT</td>
<td>Twin photometric telescope</td>
</tr>
<tr>
<td>TTL</td>
<td>Transistor transistor logic</td>
</tr>
<tr>
<td>TX</td>
<td>Transmit</td>
</tr>
<tr>
<td>TXREQ</td>
<td>The network access arbitration line</td>
</tr>
<tr>
<td>TXRX</td>
<td>The network data transfer line</td>
</tr>
<tr>
<td>VCC</td>
<td>Voltage collector collector (power supply +)</td>
</tr>
<tr>
<td>VGA</td>
<td>Versatile graphics adapter (640x480, 16 plane)</td>
</tr>
<tr>
<td>WDT</td>
<td>Watch dog timer. Used to detect fatal program crashes</td>
</tr>
<tr>
<td>WHM</td>
<td>William Herschel Telescope</td>
</tr>
<tr>
<td>WIMP</td>
<td>Windows, icons, mouse pointing technology</td>
</tr>
<tr>
<td>Z80</td>
<td>The processor family name of which the microcontrollers are the latest generation</td>
</tr>
</tbody>
</table>

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

A distributed control system for the St Andrews twin photometric telescope : Page 24 of 335
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°x6° 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](image)

**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 anticollision 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.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Centralised</th>
<th>Distributed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor type</td>
<td>80486DX @ 66 MHz</td>
<td>8031 @ 25 MHz</td>
</tr>
<tr>
<td>Processor application</td>
<td>Numerical, graphics</td>
<td>Process control</td>
</tr>
<tr>
<td>Processor cost</td>
<td>High (&gt;£1000)</td>
<td>Low (£20)</td>
</tr>
<tr>
<td>External devices</td>
<td>High (48 I/O lines)</td>
<td>High (48 I/O Lines)</td>
</tr>
<tr>
<td>Initial development time</td>
<td>Medium</td>
<td>V. long</td>
</tr>
<tr>
<td>Controller Language</td>
<td>Assembler C/C++</td>
<td>C, Assembler, Occam</td>
</tr>
<tr>
<td>Console Language</td>
<td>C/C++ (DOS/UNIX)</td>
<td>C/C++ (WINDOWS/UNIX)</td>
</tr>
<tr>
<td>Ideal programming paradigm</td>
<td>Sequential / single task</td>
<td>Multitasking, parallel processing</td>
</tr>
<tr>
<td>Ideal operation system</td>
<td>DOS / CP\M</td>
<td>UNIX / WINDOWS</td>
</tr>
<tr>
<td>Process loop time</td>
<td>Fast</td>
<td>V. Fast</td>
</tr>
<tr>
<td>System speed</td>
<td>Fast</td>
<td>slow (Network bandwidth limited)</td>
</tr>
<tr>
<td>Applications</td>
<td>High speed robots, Small PLCs, standalone dedicated processes</td>
<td>Physically large site control. Intelligent cooperative automatons. Generic control block</td>
</tr>
</tbody>
</table>

**Table 1: System architecture characteristics**

A distributed control system for the St Andrews twin photometric telescope : Page 40 of 335
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.
<table>
<thead>
<tr>
<th>Network</th>
<th>Max. nodes</th>
<th>Attributes *</th>
<th>Bandwidth B / M</th>
<th>Max. Line B / M</th>
<th>Collision Avoidance</th>
<th>Mode</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>20mA</td>
<td>-</td>
<td>S</td>
<td>-</td>
<td>-</td>
<td>Software</td>
<td>Half Duplex</td>
<td>Teleprinters</td>
</tr>
<tr>
<td>RS 232/V.34</td>
<td>2</td>
<td>IS</td>
<td>19.2K@15</td>
<td><a href="mailto:1.2K@1.2K">1.2K@1.2K</a></td>
<td>-</td>
<td>Full Duplex</td>
<td>Standard serial link.</td>
</tr>
<tr>
<td>RS 422</td>
<td>10</td>
<td>BS</td>
<td>10M@10</td>
<td><a href="mailto:100@1.2K">100@1.2K</a></td>
<td>-</td>
<td>Simplex</td>
<td>Remote display monitors</td>
</tr>
<tr>
<td>RS 423</td>
<td>2</td>
<td>USD</td>
<td>100K@10</td>
<td>3@1K</td>
<td>-</td>
<td>Full Duplex</td>
<td>Now upgraded to RS 449</td>
</tr>
<tr>
<td>RS 485</td>
<td>32</td>
<td>DBS</td>
<td>10M@40</td>
<td>100K@1100</td>
<td>Software</td>
<td>Half Duplex</td>
<td>Microcontroller networks</td>
</tr>
<tr>
<td>GPIB/IEEE 488</td>
<td>256</td>
<td>DP</td>
<td>4M@100</td>
<td><a href="mailto:100K@1.1K">100K@1.1K</a></td>
<td>Hardware</td>
<td>Half duplex</td>
<td>Hewlett Packard systems</td>
</tr>
<tr>
<td>SCSI</td>
<td>8/256</td>
<td>DP</td>
<td>-</td>
<td>-</td>
<td>Hardware</td>
<td>Half Duplex</td>
<td>Peripheral I/O, Hard disk</td>
</tr>
<tr>
<td>CAMAC/IEEE-583</td>
<td>2&lt;sup&gt;24&lt;/sup&gt;</td>
<td>PS</td>
<td>1M</td>
<td>-NA-</td>
<td>Hardware</td>
<td>Half Duplex</td>
<td>Minicomputer I/O standard</td>
</tr>
<tr>
<td>Ethernet/IEEE-802.3</td>
<td>2&lt;sup&gt;16&lt;/sup&gt;</td>
<td>S</td>
<td>10M</td>
<td>≈1M@3K</td>
<td>Software</td>
<td>Half Duplex</td>
<td>LANs, Multiple workstations</td>
</tr>
<tr>
<td>Fibre optic</td>
<td>2</td>
<td>S</td>
<td>40M</td>
<td>1K</td>
<td>Hardware</td>
<td>Half Duplex</td>
<td>Hostile environments</td>
</tr>
</tbody>
</table>

* 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).

<table>
<thead>
<tr>
<th>Line</th>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>TXRX</td>
<td>Data transmit and receive</td>
</tr>
<tr>
<td>Request for transmit</td>
<td>TXREQ</td>
<td>Data line status (request transmission / transmission pending)</td>
</tr>
</tbody>
</table>

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:

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

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).

<table>
<thead>
<tr>
<th>&lt;TARGET&gt;</th>
<th>&lt;HOST&gt;</th>
<th>&lt;COMMAND&gt;</th>
<th>&lt;DATA&gt;</th>
<th>&lt;TERMINATOR&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 Bit ASCII</td>
<td>8 Bit ASCII</td>
<td>8 Bit ASCII</td>
<td>n * 8 Bit ASCII</td>
<td>ASCII(0x5A)</td>
</tr>
<tr>
<td>'A'-'Y'</td>
<td>'A'-'Y'</td>
<td>'A'-'Y'</td>
<td>n&lt;27</td>
<td>'Z'</td>
</tr>
</tbody>
</table>

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

The target address is an 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:

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

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'.

A distributed control system for the St Andrews twin photometric telescope: Page 54 of 335
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 / echant 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).
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.

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](image)

**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 20μM 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 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](image)

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.
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.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Set -</th>
<th>Guide +</th>
<th>Track</th>
<th>Guide +</th>
<th>Set +</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shaft rotation</td>
<td>-ve</td>
<td>Off</td>
<td>+ve</td>
<td>+ve</td>
<td>+ve</td>
</tr>
<tr>
<td>Phase quadrature</td>
<td>A&lt;B</td>
<td>Undefined</td>
<td>A&gt;B</td>
<td>A&gt;B</td>
<td>A&gt;B</td>
</tr>
<tr>
<td>Waveform</td>
<td><img src="image" alt="Waveform Diagram" /></td>
<td><img src="image" alt="Waveform Diagram" /></td>
<td><img src="image" alt="Waveform Diagram" /></td>
<td><img src="image" alt="Waveform Diagram" /></td>
<td><img src="image" alt="Waveform Diagram" /></td>
</tr>
</tbody>
</table>

Table 6: Relationship between mode selected and shaft rotation
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.
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
microcontroller. As before, this was tested on the prototype Z84015 microcontroller but suffered from the low RAM size and susceptibility to program failure. This generalisation used up significant memory in redundant code.

2. Use common routines for calculating the physical algorithm and system routines, and install device drivers for each microcontroller. While this is reverting to individual programming it compromises the memory hungry common programming which would be the obvious goal with the tailored programming used in single processor environments.

4.3 : Software structure

To recap, the software developed for the microcontrollers used a common kernel with individual routines, specific to the application. The program can be split into two routines.

Interrupt routines: Serial input / output characters rx_int, encoder update

ctc_int

Main loop routines: Move, receive and transmit monitoring

The two interrupt routines enable background updating of the receiver and encoders, providing asynchronous support for tasks requiring servicing faster than the main program loop time.

4.4 : Receiver interrupt : rx_int

All network characters sent from host to target node are present at every SIO receiver input pin. The receiver interrupt is polled every time the SIO receives a
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: etc_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).
<table>
<thead>
<tr>
<th>Axis:</th>
<th>Coarse encoder resolution:</th>
<th>Fine encoder resolution:</th>
</tr>
</thead>
<tbody>
<tr>
<td>R.A. REFERENCE</td>
<td>24</td>
<td>2048 (2^{11})</td>
</tr>
<tr>
<td>DEC REFERENCE 36</td>
<td>2048 (2^{11})</td>
<td></td>
</tr>
</tbody>
</table>

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).
etc

Divide interrupt to correct frequency

Interrupt not ready

Yes

Read fine encoder

No

Exit

Old encoder - New encoder = update

Update < -512

Yes

Read coarse encoder

No

Update > 512

Set 'System ok' flag

Decrement coarse memory location

Yes

No

Merge coarse and fine encoders into present location

0xF 0xE 0xD 0xC 0xB | 0xA 0x9 0x8 0x7 0x6 0x5 0x4 0x3 0x2 0x1 0x0
Coarse encoder | Fine Encoder

Offset = Future - Present

Old encoder = New encoder

Exit

Flowchart 4: Encoder merging routine
4.6: Main loop routines

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
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.

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.
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 predefined 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.

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

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.

Diagram 17: Network view of command processing
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>Manual / automatic move</td>
</tr>
<tr>
<td>D</td>
<td>Read 16 bit data value from memory</td>
</tr>
<tr>
<td>P</td>
<td>Program 16 bit data value to RAM</td>
</tr>
<tr>
<td>H</td>
<td>Hard reset</td>
</tr>
<tr>
<td>R</td>
<td>Reset encoders</td>
</tr>
<tr>
<td>C</td>
<td>Centre axis using centring detent</td>
</tr>
</tbody>
</table>

**Table 8: Microcontroller commands**

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

**Table 9: Microcontroller actuator parameters**

---

A distributed control system for the St Andrews twin photometric telescope: Page 88 of 335
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.

<table>
<thead>
<tr>
<th>MCU axis</th>
<th>Main actuators used</th>
</tr>
</thead>
<tbody>
<tr>
<td>R.A. reference:</td>
<td>Clamp, clutch, slew motor</td>
</tr>
<tr>
<td>DEC reference:</td>
<td>Clamp, slew motor</td>
</tr>
<tr>
<td>Other:</td>
<td>TTL logic levels</td>
</tr>
</tbody>
</table>

Table 10 : Actuators used for each axis


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, clamppoff

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.

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

Table 11: Available memory addresses

A distributed control system for the St. Andrews twin photometric telescope : Page 95 of 335
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.

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Task / user restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS2</td>
<td>Pre-emptive multitasking, single user</td>
</tr>
<tr>
<td>Windows</td>
<td>Non pre-emptive multitasking, single user</td>
</tr>
<tr>
<td>DOS</td>
<td>Single task, single user</td>
</tr>
<tr>
<td>UNIX</td>
<td>Pre-emptive multitasking, Multi user</td>
</tr>
</tbody>
</table>

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 this 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 split into different distinct tasks.
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.
<table>
<thead>
<tr>
<th>Error text</th>
<th>Error type</th>
<th>Description</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executing File</td>
<td>Information</td>
<td>Started TCLINK Module</td>
<td></td>
</tr>
<tr>
<td>Executing command &lt;Command&gt;</td>
<td>Information</td>
<td>Command requested</td>
<td></td>
</tr>
<tr>
<td>Terminating module</td>
<td>Information</td>
<td>Module terminating (end of program)</td>
<td></td>
</tr>
<tr>
<td>Invalid initialisation file</td>
<td>Fatal</td>
<td>Cannot read</td>
<td>Check : INITIATE.TCL for SCL_TCLQUICK_CONFIG entry, path existence.</td>
</tr>
<tr>
<td>Module interlock unavailable</td>
<td>Fatal</td>
<td>Cannot execute TCLINK module</td>
<td>Check : INITIATE.TCL for SCL_TCLINK_EXE entry, path existence.</td>
</tr>
<tr>
<td>System fragmented</td>
<td>Fatal</td>
<td>Cannot find DDE address of TCLINK module</td>
<td>Commonly linked with Module Interlock unavailable. If ONLY error, Windows environment unstable. Restart.</td>
</tr>
</tbody>
</table>

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).

<table>
<thead>
<tr>
<th>Axis selected:</th>
<th>Reference tube</th>
<th>Offset tube</th>
<th>Dome</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available speeds:</td>
<td>Slew</td>
<td>Set</td>
<td>Selected by push-button</td>
</tr>
<tr>
<td></td>
<td>Set</td>
<td>Guide</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Guide</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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).
<table>
<thead>
<tr>
<th>Title text</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCLLINK:INTERLINK</td>
<td>Module idle</td>
</tr>
<tr>
<td>TCLLINK:INTERLINK[..]</td>
<td>Transmitting statement [..] from DDE link</td>
</tr>
<tr>
<td>TCLLINK:UPDATE</td>
<td>Initiating batch file</td>
</tr>
<tr>
<td>TCLLINK:...</td>
<td>Executing batch file ..</td>
</tr>
</tbody>
</table>

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

Table 16: TCLINK error codes
<table>
<thead>
<tr>
<th>Command</th>
<th>Command Field 1</th>
<th>Command Field 2</th>
<th>Command Field 3</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Ref</td>
<td>&lt;Hrs&gt; &lt;Min&gt; &lt;Sec&gt;</td>
<td>Points telescope to given location</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Deg&gt; &lt;Min&gt; &lt;Sec&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Off</td>
<td></td>
<td>&lt;Hrs&gt; &lt;Min&gt; &lt;Sec&gt;</td>
<td>Offsets offset telescope to location relative to reference telescope</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Deg&gt; &lt;Min&gt; &lt;Sec&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dome</td>
<td></td>
<td>&lt;Degrees&gt;</td>
<td>Sets dome to &lt;Degrees&gt; azimuth</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 &lt; Degrees &lt; 359</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delay</td>
<td></td>
<td>&lt;Time&gt;</td>
<td>Delays execution of next command for &lt;Time&gt; Seconds</td>
<td></td>
</tr>
<tr>
<td>Wait</td>
<td></td>
<td></td>
<td>Suspends batch file and prompts user to resume</td>
<td></td>
</tr>
<tr>
<td>Filter</td>
<td>Ref</td>
<td>&lt;UMF_REFERENCE_FILTER&gt;</td>
<td>Sets the reference filter</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Position&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 &lt; Position &lt; 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Off</td>
<td>&lt;UMF_OFFSET_FILTER&gt;</td>
<td>Sets the Offset filter</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;Position&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 &lt; Position &lt; 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aperture</td>
<td>Action</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Both</td>
<td><code>&lt;UMF_BOTH_FILTER&gt;</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Position&gt;</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0&lt;Position&lt;7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ref</td>
<td><code>&lt;UMFREFERENCE_APERTURE&gt;</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Position&gt;</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0&lt;Position&lt;7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Off</td>
<td><code>&lt;UMF_OFFSET_APERTURE&gt;</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Position&gt;</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0&lt;Position&lt;7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Both</td>
<td><code>&lt;UMF_BOTH_APERTURE&gt;</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Position&gt;</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0&lt;Position&lt;7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Centre</td>
<td>Dec_Ref</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Centres Reference DEC tangent arm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ra_Off</td>
<td>Centres Offset R.A. tangent arm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dec_Off</td>
<td>Centres Offset DEC tangent arm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reset</td>
<td>Ra_Ref</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Locks Reference R.A. encoders</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dec_Ref</td>
<td>Locks Reference DEC encoders</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Camera</td>
<td>00</td>
<td>00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>----</td>
<td>----</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Ra_Off</strong></td>
<td>Locks Offset R.A. encoder</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dec_Off</strong></td>
<td>Locks Offset DEC encoder</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dome</strong></td>
<td>Locks DOME encoder</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>All</strong></td>
<td>Locks all encoders</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Camera</strong></td>
<td><strong>Ref</strong></td>
<td>Selects Reference camera</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Off</strong></td>
<td>Selects Offset camera</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Finder_Ref</strong></td>
<td>Selects Reference finder camera</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Finder_Off</strong></td>
<td>Selects Offset finder camera</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>&lt; Time&gt;</strong></td>
<td>Sets camera exposure time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>0 &lt; Time &lt; 5000</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>System</strong></td>
<td><strong>Ra_Ref</strong></td>
<td>Sets reference R.A. composite encoder offset</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt; Encoder Offset&gt;</td>
<td>0 ≤ Encoder Offset ≤ 65535</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Ra_Off</strong></td>
<td>Sets reference DEC composite encoder offset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt; Encoder Offset&gt;</td>
<td>0 ≤ Encoder Offset ≤ 65535</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dec_Ref</strong></td>
<td>Sets offset R.A. encoder offset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt; Encoder Offset&gt;</td>
<td>0 ≤ Encoder Offset ≤ 65535</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dec Off</td>
<td>&lt; Encoder Offset &gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 ≤ Encoder Offset ≤ 65535</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sets offset DEC encoder offset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>&lt; Network Command &gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sends a low level command to the network</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LED</td>
<td>&lt; Level &gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 ≤ Level ≤ 15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sets tube I.R. brightness</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sky</td>
<td>&lt; Step &gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 &lt; Step &lt; 200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sets sky position</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Star</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Resets aperture to star position</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abort Batch</td>
<td>Terminates the current batch file</td>
</tr>
<tr>
<td>Abort Source</td>
<td>Clears current commands line edit box</td>
</tr>
<tr>
<td>Stop</td>
<td>Stops the telescope. Identical to Stop on TCLMAN</td>
</tr>
</tbody>
</table>

**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).

<table>
<thead>
<tr>
<th>Colour</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLACK</td>
<td>Variable valid and accepted</td>
</tr>
<tr>
<td>RED</td>
<td>Network error in processing last variable update</td>
</tr>
<tr>
<td>GREEN</td>
<td>Transmitting variable update. Waiting for request acknowledge</td>
</tr>
<tr>
<td>BLUE</td>
<td>Processing variable update. Waiting for process acknowledge</td>
</tr>
</tbody>
</table>

**Table 19 : Variable colour status**
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 timeout 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).
<table>
<thead>
<tr>
<th>Error text</th>
<th>Error Type</th>
<th>Description</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>System fragmented</td>
<td>Fatal</td>
<td>Cannot find DDE address of TLLINK module.</td>
<td>See TCLQUICK</td>
</tr>
<tr>
<td>Module Interlock Unavailable</td>
<td>Fatal</td>
<td>Cannot execute TLLINK module.</td>
<td>See TCLQUICK</td>
</tr>
<tr>
<td>Executing module</td>
<td>Information</td>
<td>Executing TCLCTRL.</td>
<td></td>
</tr>
<tr>
<td>Interactive</td>
<td>Information</td>
<td>Batch file has ended. Control returned to the operator.</td>
<td></td>
</tr>
<tr>
<td>Executing drop file</td>
<td>Information</td>
<td>Batch file .. started.</td>
<td></td>
</tr>
<tr>
<td>Terminating module</td>
<td>Information</td>
<td>Module terminating (end of program).</td>
<td></td>
</tr>
<tr>
<td>Interlink Message &lt;..&gt;</td>
<td>Information</td>
<td>DDE message .. received from TLLINK.</td>
<td></td>
</tr>
<tr>
<td>Abort Command</td>
<td>Information</td>
<td>'Abort Command' push-button pressed. Edit box cleared.</td>
<td></td>
</tr>
<tr>
<td>Stop</td>
<td>Information</td>
<td>'Stop' push-button pressed. Telescope stopped.</td>
<td></td>
</tr>
<tr>
<td>Interactive Command &lt;..&gt;</td>
<td>Information</td>
<td>Processing command ..</td>
<td></td>
</tr>
</tbody>
</table>

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 an 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.

---

A distributed control system for the St Andrews twin photometric telescope : Page 125 of 335
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


(5) Reddish V. C., September 1966, Sky and Telescope, 124


(9) Golyas Yu. E., Tikhonov V. A., 1983, UDC, 681, 139


(11) Boyd L. J., 1984, IAPP symposium

(12) Treffers R. T., 1985, PASP, 97, 446


(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


(23) Sinnott R. W., 1983, Sky and Telescope, April, 311


(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


(43) Swainston F., 1991, Systems Approach to Programmable Controllers, Nelson (Thomas), Melbourne


(47) Forsythe W., Goodall R. M., 1991, Digital Control, Macmillan education, Basingstoke


(49) Hartley M. G., 1962, An Introduction to Electronic Analogue Computers


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

<table>
<thead>
<tr>
<th>Filename:</th>
<th>CPU.PLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>CPU memory and I/O logic decoder</td>
</tr>
<tr>
<td>Device used:</td>
<td>GAL 16V8 25nS / Replacement code RS 655-745</td>
</tr>
</tbody>
</table>

/*
/*  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
/*

PIN 1 = CLK ; /* 10MHz CPU 0 deg clock */
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
/*

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

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


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


/* 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 distributed control system for the St Andrews twin photometric telescope: Page 137 of 335
A.2 : Motor enable and direction encoder

<table>
<thead>
<tr>
<th>Filename:</th>
<th>MOTOR.PLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Motor enable / direction encoder limit cut-out</td>
</tr>
<tr>
<td>Device used:</td>
<td>GAL 16V8 25nS / Replacement code RS 655-745</td>
</tr>
</tbody>
</table>

/* */
/* 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 = AEOUT ; /* Motor A driver out */
PIN 13 = ADOUT ; /* Motor A direction out */
PIN 14 = NCA ; /* Not connected */
PIN 15 = NCB ; /* Not connected */
PIN 16 = BEOUT ; /* Motor B driver out */
PIN 17 = BDOUT ; /* Motor B direction out */
PIN 18 = NCC ; /* Not connected */
PIN 19 = NCD ; /* Not connected */
/* */
/* Finally, define the power supply and the control */
/* */

A distributed control system for the St Andrews twin photometric telescope : Page 138 of 335
/* 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 ;
AOUT = MEA & !LAL & !LAH
| MEA & MDA & !LAH
| MEA & !MDA & !LAL ;
BDOUT = MEB & MDB & !LBH ;
BOUT = MEB & !LBL & !LBH
| MEB & MDB & !LBH
| MEB & !MDB & !LBL ;

/* End of MOTOR,PLD source code */
A.3 : Network controller and driver

<table>
<thead>
<tr>
<th>Filename</th>
<th>BUS_CTRL.PLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Network monitor and driver</td>
</tr>
<tr>
<td>Device used</td>
<td>GAL 16V8 25nS / Replacement code RS 655-745</td>
</tr>
</tbody>
</table>

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

DEVICE 16v8;
TITLE BUS_CTRL;
REVISION 002;
AUTHOR RICHARD GEARNS;
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) */
PIN 19 = SYSOK; /* 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 line 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:

<table>
<thead>
<tr>
<th>Extension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.H</td>
<td>Header files including definitions, macro declarations, excluding all source code</td>
</tr>
<tr>
<td>*.PRE</td>
<td>Compiler pre-processing directives</td>
</tr>
<tr>
<td>*.REL</td>
<td>Z80 source code which has been compiled separately, but is common to all programs</td>
</tr>
</tbody>
</table>
B.1: Include file for character definition

<table>
<thead>
<tr>
<th>Filename:</th>
<th>character.h</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Include file for use in Z80 microcontrollers</td>
</tr>
</tbody>
</table>

;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

<table>
<thead>
<tr>
<th>Filename:</th>
<th>map_io.h</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Include file for use in Z80 microcontrollers</td>
</tr>
</tbody>
</table>

; This file sets up the io locations and offsets;

```assembly
code
base_io   .equ  0ff0h ; define the base address of the io channels
code
encoder_io .equ  0ff2h ; 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

<table>
<thead>
<tr>
<th>Filename:</th>
<th>memory.h</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Include file for use in Z80 microcontrollers</td>
</tr>
</tbody>
</table>

; 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 + 030Dh) ; 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 + 031Dh) ; 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
course .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

A distributed control system for the St Andrews twin photometric telescope : Page 146 of 335
moveflag .equ (ram + 032Ah) ; automate (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)

<table>
<thead>
<tr>
<th>Filename:</th>
<th>statement.h</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Include file for use in Z80 microcontrollers</td>
</tr>
</tbody>
</table>

; 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

<table>
<thead>
<tr>
<th>Filename:</th>
<th>rambo.h</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Include file for use in Z80 microcontrollers</td>
<td></td>
</tr>
</tbody>
</table>

; 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
crc_ctrl_2 .equ 012h
crc_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

<table>
<thead>
<tr>
<th>Filename:</th>
<th>setup_io.rel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Include file for use in Z80 microcontrollers</td>
</tr>
</tbody>
</table>

; 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_0,0011011 b)

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

    io_ctrl(ctc_ctrl_0,0,195)

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

    io_ctrl(ctc_ctrl_1,0001011 b)
    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_2,10000101 b)
    io_ctrl(ctc_ctrl_2,220)

; int., timer mode, /256, rising edge, autotrig, add const, free run, 1Khz

    io_ctrl(ctc_ctrl_3,0011011 b)

; 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.
;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(1,00001100b)

;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 & Offh
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: Autoneables 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. [xxxxxxxx01]

   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

<table>
<thead>
<tr>
<th>Filename</th>
<th>scma.pre</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Include file for use in Z80 microcontrollers</td>
</tr>
</tbody>
</table>

;This file sets up the scma assembler to optimise all variables
.title "Microcontroller unit ver. 3.1"
.nopage
. 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 PHOTOMETRIC 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

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
```

---

*A distributed control system for the St Andrews twin photometric telescope: Page 156 of 335*
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,(0FF0h) ; low byte of fine encoder from I/O
ld l,a ; and store (as part of twin reg. hl)
ld a,(0FF1h) ; 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,l ; and show that position is 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:
pop iy; exit routine here
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 (Off00h & ($ + Offh))
iow_setup:
#include "c: \design \scma \z80 \setup_io.rel"
ret

********************************************************************

.org (Off00h & ($ + Offh))

; Interrupt table in here
int_table:

int_table_ctc:
.byte unused & Offh
.byte (unused & Offh) > > 8
.byte unused & Offh
.byte (unused & Offh) > > 8
.byte ctc_int & Offh
.byte (ctc_int & Offh) > > 8
.byte unused & Offh
.byte (unused & Offh) > > 8
.byte unused & Offh
.byte (unused & Offh) > > 8
.byte unused & Offh
.byte (unused & Offh) > > 8
byte unused & Offh
byte (unused & Offh) > > 8

int_table_sio:
.byte rx_test & Offh
.byte (rx_test & Offh) > > 8

********************************************************************

.org (Off00h & ($ + Offh))

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

A distributed control system for the St Andrews twin photometric telescope: Page 159 of 335
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
ld hl, (present) ; lock present & future together
ld (future), hl
jp print_text ; tx message to network

; ********************************************************************
; setup:
; ld a, OFFh ; 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 capacity decay in comms line
ld a,0 ; Set slew flag to 0
ld (slewflag),a ; and switch move to manual
ld (moveflag),a ; invalid ide encoder ok flag
ld (okflag),a ; clear limits
ld a,l ; 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 ; encoder reset, sync encoders
cp 'R'
jp z,starttest ; 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,OFFFEh ; 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,movemonl ; 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,slewoff ; 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
```

A distributed control system for the St Andrews twin photometric telescope
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); hl 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

A distributed control system for the St Andrews twin photometric telescope: Page 163 of 335
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'??
sclf ; 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 inppresl
jp ret_com

;********************************************************************

dispres:
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 sidereal drive
  ld  (base_io+direction),a
  ld  a,0ffh       ; switch clutch on
  ld  (base_io+motor),a
slewstart1:
  ld  a,(0fff1h)   ; is clutch on?
  bit 5,a
  jp  nz, slewstart1
  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,00ffh      ; 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:
Id a,(0fff4h) ; is clamp out?
bit 5,a
jp nz,clampoff1 ; yes, so exit
ld hl,(delay_over) ; delay to overcome overload
call delay
ld a,(0fff1h) ; is overload sensor active?
bit 6,a
jp z,clampoff3 ; yes? knock out clamp and repeat
ld a,0fffh ; no, so switch clamp motor on
ld (base_io+motor),a
jp clampoff2

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

clewstop:
ld a,0fffh ; stop slew motor
ld (base_io+motor),a
ld hl,(delay_mech) ; wait for relays
call delay
ld a,0fffh ; reset contacts
ld (base_io+direction),a
call clampon ; lock clamp on
ld a,0fffh ; 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,0fffh ; 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,0fffh ; no, so switch on clutch
ld (base_io+motor),a
jp clampon2
clamp_on1:
    ld     hl, (delay_over) ; wait for inertia to build up
    call   delay
clamp_on3:
    ld     a, (Off1h) ; 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, Offh ; it has hit the end, so stop motor
    ld     (base_io+ motor), a
    ld     hl, (delay_mech) ; wait for relays
    call   delay
    ld     a, 0fh ; 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

A distributed control system for the St Andrews twin photometric telescope : Page 168 of 335
clear1:
  ld  (ix+0),a
  inc  ix
  djnz  clearl
  ret

,*************************************************************************

tx_request:
  ld  iy,tx_buffer

,*************************************************************************

; 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) ; delay for a preset time
  call  delay
  sio_ctrl(5,01101000b) ; set txrx control inactive
  ld  hl,(delay_cap) ; delay for capacitor
  call  delay
  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_l:
sio_in(0) ; get tx empty (bit status)
bit 2,a ; empty ??
jp z,print_l ; 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

A distributed control system for the St Andrews twin photometric telescope ; Page 170 of 335
C.2 : Source code for DEC. axis microcontroller

<table>
<thead>
<tr>
<th>Filename:</th>
<th>dec.z80</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Source code for DEC. axis Z80 microcontroller</td>
</tr>
</tbody>
</table>

; ST. ANDREWS TWIN PHOTOMETRIC 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"

; processor startup code
.start .org 0000h
  di ; 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

.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,'T'      ; 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

***********

cto_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)
lد a, (0FFF2h) ; get low byte of fine encoder from input
ld l,a ; and store (as part of twin reg. hl)
lد a, (0FFF3h) ; now get high byte of fine encoder
and 007h ; mask out all but low 3 bits
ld h,a ; and store
dl (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)
lد 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:
lد 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:
lد a, (OFFFOh) ; 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 is 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:

pop iy ; exit routine here
pop ix
pop hl ; clear up and exit
pop de
pop bc
pop af
ei ; and enable interrupts
reti ; and exit

A distributed control system for the St Andrews twin photometric telescope : Page 175 of 335
ei
ret

;********************************************************************
#include "c:\design\scma\z80\setup_io.rel"
ret

;********************************************************************

org (0ff00h & ($ + 0ffh))
io_setup:

.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 & ($ + 0ffh))

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

call tx_request

call print_text ; tx message to network

ld hl,14904d ; Delay startup 5 sec
call delay

A distributed control system for the St Andrews twin photometric telescope: Page 176 of 335
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 printtext ; tx message to network

;*********************************************************
;setup:
ld a,OFFh ; 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,0000FFh ; 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  ; soft reset, reset encoders
cp 'R'  ; 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('T',delay_inert)
select('A',delay_arb)
select('U',delay_out)
select('K',okflag)
select('D',intcount)
ld iy,0FFF Eh ; 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,movemonl ; 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) ; hl 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,OFCh ; preload with guide +
cp '3' ; is it guide + ('3')
jp z,manmove ; yes, so exit
ld b,OFCh ; 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,OFFh ; 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

;*****************************************************************************
sleon:
; 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 1

;*****************************************************************************
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 0

;*****************************************************************************
guideon:
seton:
    ld b,OFCh ; preload b with fine motor guide+
    ld a,(offset_dir) ; get direction
    cp 0 ; which direction??
    jp nz,setonl ; c w , so start cw motor
    ld b,OFDh ; ccw, so load b with guide-
setonl:
    ld a,b ; swap for load
    ld (base_io+direction),a
    ret

;*****************************************************************************
fineoff:
    ld a,OFFH ; switch all fine inputs off
    ld (base_io+direction),a
    ret

;*****************************************************************************
centrego:
    call centre ; centre tangent arm
    jp ret_com ; and clear rx buffer

;*****************************************************************************
centre:
    ld a,(0FF1h) ; 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,(OFFFlh) ; 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

********************************************************************

inpress:
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)
inpress1:
ld a,(ix+0) ; load in bit to be tested
cp '1' ; is it a '1'??
scf ; preload with a carry set
jp z,inpress1 ; and if it is, start rolling
and a ; else clear flag
inpressl:
rl (iy+0) ; rotate carry into memory location
rl (iy+1) ; and through into high byte
inc ix ; next rx holding buffer
djnz inpressl
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

A distributed control system for the St Andrews twin photometric telescope : Page 182 of 335
ld b,8 ; set loop to shift 8 bits

; dispres4:
sla a ; shift top bit into carry
jp nc,dispers3 ; 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,OFFh ; load clamp direction OUT
ld (base_io+direction),a ; and store
ld hl,(delay_mech) ; wait for contacts
call delay ; cold start,

; 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,OFFh ; now stop motor

; (will automatically stop anyway)
ld (base_io+motor),a ; and store
Id hl,(delay_mech) ; wait for contacts
jp delay ; and return

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

********************************************************************************
clampon:
id a,0BFh ; load clamp direction IN
id (base_io+direction),a ; and store
id hl,(delay_mech) ; wait for contacts
call delay ; wait..
clampon1:
id a,0BFh ; now start up motor
id (base_io+motor),a ; and store
id a,(0FFF4h) ; get clamp in microswitch
bit 5,a ; is it set??
jp z,clampon1 ; no, so repeat
id a,0FFh ; now stop motor
(id will automatically stop anyway)
(id base_io+motor),a ; and store
id hl,(delay_mech) ; wait for contacts
call delay
ret ; and return

********************************************************************************
limitmon:
id a,(0FFF4h) ; get limits in
and 00Ch ; mask off all but fine limits
call nz,centre ; if hit, recentre 'scope
id a,(0FFF4h) ; get limits in again
and OC0h ; mask off all but axis limits
ret z ; none hit, so exit
id a,0 ; clear moveflag
id (moveflag),a ; and store
call slewstop ; stop slewing
call fineoff ; stop fine motors
id a,(0FFF4h) ; get limits in
and 0CCh ; mask off all other limits
id 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__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
djmz clear1
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

tx_request:
ld iy, tx_buffer

TX_REQ2:
ld a, (ix+0)
cp chr_null

A distributed control system for the St Andrews twin photometric telescope : Page 185 of 335
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 arbitation, 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

A distributed control system for the St Andrews twin photometric telescope : Page 186 of 335
ld ix,tx_buffer ; like *
ld (ix+0),a ; that message is sent
ld hl,(delay_out) ; set txrx control inactive

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_t2:
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 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

;******************************************************************** 9

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"

A distributed control system for the St Andrews twin photometric telescope: Page 187 of 335
C.3 : Source code for OFFSET axis microcontrollers

<table>
<thead>
<tr>
<th>Filename:</th>
<th>OFFSET.Z80</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Source code for offset axis Z80 microcontrollers</td>
</tr>
</tbody>
</table>

; 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"

; Processor startup code
.start .org 0000h
  di ; interrupts from table
  im 2
  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

.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,r.rx_test1 ; and loop if not yet available

.rx_test6:
  ld ix,r.rx_buffer ; get start of buffer
  ld iy,r.rx_cursor ; get cursor pointer
  ld a,(r.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_buffer
  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
push bc
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,
ld hl,(present) ; so determine whether to set or clear
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
ld a,(newencoder) ; now compare either side to the new Phase
and 03h ; load a with new encoder, to enable
; compare with ix+?
cp (ix+2) ; mask out all but phase quad information
jp z,ctcint5 ; in memory
jp z,ctcint6 ; is the dome going cw (+ve)
cp (ix+0) ; yes, so jump to cw routine
jp z,ctcint7 ; is dome going ccw (-ve)
dl a,0 ; yes, so jump to ccw routine.
l d (okflag),a ; if it's not going cw/ccw (as above) -
jp ctcint1 ; it's bugged.
l d (present),hl ; so clear ok flag -
l d hl,(present) ; show that encoder is knackered
jp ctcint2 ; and exit (will calculate offset,
l d (offset),hl ; but ok flag overrides)
now compare either side to the new Phase
ctcint5:
l d a,01h ; set the direction = cw
ld (dome_direction),a ; and save in encoder direction,
inc hl ; for use by interrupt routine
ld hl,(present) ; get current position
ld (present),hl ; and increment
jp ctcint6 ; and store back again
ctcint6:
l d a,0 ; jump to exit saving old encoder
ld (dome_direction),a ; set the direction = ccw
ld hl,(present) ; and save in encoder direction,
dec hl ; for use by interrupt routine
ld (present),hl ; get current position
ctcint7:
l d hl,(present) ; and decrement
ld (present),hl ; and store back again
ld (offset),hl ; 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 ??
jp ctcint10
ctcint9:
l d hl,(future) ; here swap future and present registers
ld de,(present) ; and here too
and a ; clear carry flag
sbc hl,de ;
l d (offset),hl ; save offset in memory
ld a,1 ; and invert direction
ld (offset_dir), a

ctcin10:

dcctl:

dcctl1:

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
ret ; and exit

;****************************

io_setup:

; Interrupt table in here

int_table:

int_table_cctl:

.byte unused & 0ffh
.byte (unused &0ff0h) >> 8
.byte unused & 0ffh
.byte (unused &0ff0h) >> 8
.byte ctc_int & 0ffh
.byte (ctc_int &0ff0h) >> 8
.byte unused & 0ffh
.byte (unused &0ff0h) >> 8
.byte unused & 0ffh
.byte (unused &0ff0h) >> 8
.byte unused & 0ffh
.byte (unused &0ff0h) >> 8

int_table_sio:

.byte rx_test & 0ffh
.byte (rx_test &0ff0h) >> 8

;****************************

start_main:

call setup ; setup memory, ports default values

; enable interrupts

ei

sio_ctrl(5, 01101000b) ; testd
sio_ctrl(0, 020h) ; enable rx on next character

A distributed control system for the St Andrews twin photometric telescope
ld ix,text_on
call tx_request

main_loop:
call movemon
call limitmon
call rx_mon
call print_text
jp main_loop

;**************************************************************************

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,OFCh ; switch on motor, +ve direction.
ld (base_io+direction),a

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,OFCh ; switch on motor, +ve direction.
ld (base_io+direction),a

setup:
ld a,OFFh ; 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

A distributed control system for the St Andrews twin photometric telescope : Page 194 of 335
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,00FFh ; 85 mS arbitration delay
ld (delay_arb),hl ; for arbitration delay
ld hl,00FFh ; 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'

A distributed control system for the St Andrews twin photometric telescope: Page 195 of 335
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??
ld  a,'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 bytes)
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',OFFFOh)
select('W',OFFF2h)
select('H',OFFF4h)
select('M',delay_mech)
select('I',delay_inert)
select('A',delay_arb)
select('U',delay_out)
select('K',okflag)
select('D',intcount)
ld iy,OFFFEh
  ; 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

A distributed control system for the St Andrews twin photometric telescope : Page 196 of 335
ld ix, text_rtn
cl call tx_request
cl call print_text
jp testcode

;************************************************************************
movemon:
lb a, (moveflag) ; is automove flag set??
cp 0 ; automatic if! = 0
rb ret z ; it isn't, so exit - axis can move manually
lb a, (okflag) ; is unit OK??
cp 0 ; buggered if 0
jp z, movemon1 ; yes, so stop all motors
lb hl, (offset) ; is slew hysteresis > offset
lb 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
lb a, 0 ; reset automove flag
lb (moveflag), a ; save in register
lb ix, text_ok ; send message saying drive stopped
call tx_request ; and transmit

movemon1:
lb a, OFFh ; else switch off everything
lb (base_io + direction), a
ret

;************************************************************************
limitmon:
lb a, (0FFF4h) ; get limits from input
and 3 ; and mask out all others...
cp 3 ; are all bits set? - no limits hit
rb ret z
lb ix, text_hit
cl call tx_request
call print_text
jp starttest ; limit hit, recentre

;************************************************************************
slewon:
lb a, (offset_dir) ; get offset direction
cp 00h ; which direction is it going in??
jp z, slewon1 ; CCW, so next routine
lb a, OFCh ; get slew output
lb (base_io + direction), a ; and load CW direction
ret ; and exit

slewon1:
lb a, 0FDh ; Motor CCW
lb (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
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
   jr nz,centrel ; this way, so run motor
   ld c,0FDh ; no, other way, so reload c with correct dir
centrel:
   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???
   jr z, centrel ; no, so continue
   ld a,0FFh ; yes, centre has been reached
   ld (base_io+direction),a; so stop motor and exit
   ret

A distributed control system for the St Andrews twin photometric telescope: Page 198 of 335
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 ; next bit in tx buffer
djnz dispres3 ; repeat 8 times
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 μS
; delay time = 2.5 + 335.5*HL μS
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

A distributed control system for the St Andrews twin photometric telescope : Page 201 of 335
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_l:
sio_in(0) ; get tx empty (bit status)
bit 2,a ; empty ??
jp z,print_l ; 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 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"

A distributed control system for the St Andrews twin photometric telescope : Page 202 of 335
C.4 : Source code for DOME axis microcontroller

<table>
<thead>
<tr>
<th>Filename:</th>
<th>dome.z80</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Source code for dome axis Z80 microcontroller</td>
</tr>
</tbody>
</table>

```
; ST. ANDREWS TWIN PHOTOMETRIC 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
    ; Processor startup code
    di
    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

.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
```

A distributed control system for the St Andrews twin photometric telescope : Page 204 of 335
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:

A distributed control system for the St Andrews twin photometric telescope : Page 205 of 335
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
(jc 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
calcint3:
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
calcint2:
; Encoder has incremented/decremented, so look up old encoder
; in data table, and see which side of it is the new reading
; 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
calcint4:
inc ix ; next phase quad data table location
cp (ix+1) ; Data entry == old_encoder ??
jp nz,calcint4 ; no so repeat (max 4 times):
; can only be 00 01 11 10
; data table pointer (ix) now pointing to old Phase
ld a,(newencoder) ; load a with new encoder,
and 03h ; mask out all but phase quad
; information in memory
cp (ix+2) ; is the dome going cw (+ve)
; yes, so jump to cw routine

; if it's not going cw/ccw (as above)
; it's buggered.

; if it's not going cw/ccw (as above)
; it's buggered.

; show that encoder is knackered

; and exit (will calculate offset,
; but ok flag overrides)

; set the direction = cw

; and save in encoder direction,
; for use by interrupt routine

; get current position

; and increment

; and store back again

; jump to exit saving old encoder

; set the direction = ccw

; and save in encoder direction.
; for use by interrupt routine

; get current position

; and decrement

; and store back again

; compare present to future

; to get direction of offset

; clear carry flag

; jump if future < present

; save offset in memory

; and set direction to ??

; here swap future and present registers

; and here too

; clear carry flag

; save offset in memory

; and invert direction

; and this extra bit to the calcoffset routine is due to the cyclic nature of dome.

; 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
reiti ; and exit

;**************************** unused:

;****************************

.org (off00h & ($ + offh))
io_setup:
#include "c:\design\scma\z80\setdo_io.rel"
ret

; 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
.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
ld a,OFFh
ld (okflag),a ; invalidate ok flag
ld a,OFFh
ld (base_io+motor),a
ld a,0FEh ; switch on dome CW fast!!!
ld (base_io+direction),a

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,OFFh
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,OFFh ; 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,OFFh ; shut off all motors, whatever state
ld (base_io+motor),a ; motors off
ld (base_io+direction),a ; direction relays off

A distributed control system for the St Andrews twin photometric telescope : Page 209 of 335
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 (1 deg)
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 ; and switch move to manual
ld (moveflag),a ; invalidiadte encoder ok flag
ld (okflag),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'

A distributed control system for the St Andrews twin photometric telescope : Page 210 of 335
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', OFFF0h)
    select('W', OFFF2h)
    select('H', OFFF4h)
    select('M', delay_mech)
    select('I', delay_inert)
    select('A', delay_arb)
    select('U', delay_out)
    select('K', okflag)
    select('D', intcount)
    ld iy, OFFF Eh ; sent to dud space
    ret ; and return. this MAY NOT
    ; be a good idea.

A distributed control system for the St Andrews twin photometric telescope: Page 211 of 335
ld a, 'P'
ld (rx_holding + 3), a
call dispres
call print_text
ld ix, text_rtm
call tx_request
call print_text
jp testcode
;j:-------------------------------------------------------------

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 ; buggered if 0
jp z, movemon1 ; yes, so stop all motors
ld hi, (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 hi, (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 hi, (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, OFFh ; else switch off everything
ld (base_io + motor), a
ld (base_io + direction), a
ret
;j:-------------------------------------------------------------
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, OFEh ; get slew output
ld (base_io + direction), a ; and load CW direction

A distributed control system for the St Andrews twin photometric telescope: Page 212 of 335
ld a,OFFh ; clear outputs to CCW direction
ld (base_io+motor),a ; and load
ret ; and exit

slewon1:
ld a,OFFh ; clear outputs to CW direction
ld (base_io+direction),a; and load
ld a,OFh ; get slew output to CCW
ld (base_io+motor),a ; and load
ret

;************************************************
seton:
ld a,(offset_dir) ; get offset direction
cp OOh ; which direction is it going in??
jp z,seton1 ; CCW, so next routine
ld a,OFFh ; get set output
ld (base_io+direction),a; and load CW valves
ld a,OFh ; clear outputs to CCW direction
ld (base_io+motor),a ; and load
ret ; and exit

seton1:
ld a,OFFh ; clear CW outputs
ld (base_io+direction),a ; and load
ld a,OFh ; get set output to CCW
ld (base_io+motor),a ; and load
ret

;************************************************
guideon:
ld a,(offset_dir) ; get offset direction
cp OOh ; which direction is it going in??
jp z,guideon1 ; CCW, so next routine
ld a,OFFh ; get guide output
ld (base_io+direction),a; and load CW valves
ld a,OFFh ; clear outputs to CCW
ld (base_io+motor),a ; and load
ret ; and exit

guideon1:
ld a,OFFh ; clear outputs to CW
ld (base_io+direction),a; and load
ld a,OFh ; 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
moveint1:
    ld a,00h ; yep, so return
    ld (moveflag),a ; automove has been canceled, so stop
    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,manmove1 ; yes, so exit
    ld b,0FDh ; preload b with slew direction CW
    cp '5' ; is it CW slew??
    jp z,manmove1 ; yes, so start
    ld b,0FBh ; preload b with slew direction CCW
    cp '6' ; is it CCW slew??
    jp z,manmove1 ; 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

;s;*****************************************************************
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

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

clear1:
  ld (ix+0), a
  inc ix
djnz clear1
  ret

;s;*****************************************************************
tx_request:
  ld iy, tx_buffer
tx_reql:
  ld a, (ix+0)
cp chr_null
  jp nz, tx_reql
  ld a, chr_null
  ld (iy+0), a
  ret

tx_reql:
  ld (iy+0), a
  inc ix
  inc iy
djnz tx_reql2

;s;*****************************************************************
; 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 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 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

A distributed control system for the St Andrews twin photometric telescope: Page 218 of 335
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 "00000000000000ACK"
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 Norton's 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

<table>
<thead>
<tr>
<th>Filename:</th>
<th>TCLCTRL.MAK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>make file for C700 compiler</td>
</tr>
</tbody>
</table>

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

link TCLCTRL, /align:16, NUL, /nod sibcsw 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**

<table>
<thead>
<tr>
<th>Filename:</th>
<th>TCLCTRL.DEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Definition file for C700 complier</td>
</tr>
</tbody>
</table>

; tcctrl.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

<table>
<thead>
<tr>
<th>Filename:</th>
<th>TCLCTRL.H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Include file for C700 compiler</td>
</tr>
</tbody>
</table>

//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 compiler

//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);

A distributed control system for the St Andrews twin photometric telescope : Page 224 of 335
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=0, 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)
{

A distributed control system for the St Andrews twin photometric telescope : Page 225 of 335
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,IDS_ABORT_SOURCE, hInstance,NULL);

hwndAbortCommand = CreateWindow("button","Abort Command",
WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
0,0,0,0,hwnd,IDS_ABORT_COMMAND, hInstance,NULL);

hwndStop = CreateWindow("button","Stop",
WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
0,0,0,0,hwnd,IDS_STOP, hInstance,NULL);

hwndEntry = CreateWindow("edit","STARTUP",
WS_CHILD | WS_VISIBLE | ES_LEFT,
0,0,0,0,hwnd,IDS_ENTRY, hInstance,NULL);

A distributed control system for the St Andrews twin photometric telescope : Page 226 of 335
IpfnEntryProc = MakeProcInstance( (FARPROC)EntryProc,hInstance);
IpfnOldEntryProc = (FARPROC)
               GetProcAddress(hwndEntry,GWL_WNDPROC);
SetWindowLong(hwndEntry,GWL_WNDPROC,(LONG) IpfnEntryProc);

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)
        {
            printf(ErrorFile,
                    "TCLCTRL:System Fragmented. Terminating Module\n")
            MessageBox(hwnd,}
if((hwndTclLink =FindWindow(NULL,"TCLLINK;INTERLINK"))
    = =NULL)
{
    MessageBox(hwnd,"TCLCTRL: Module Interlock unavailible"
        ;"TCLCTRL",MB_INFORMATION|MB_OK);
    fprintf(ErrorFile,"TCLCTRL: Module Interlock unavailible");
    PostQuitMessage(0);
}

GlobalAddAtom("INVALID");
fprintf(ErrorFile,"TCLCTRL:Executing ModuleXn");
return 0;

case WM_SETFOCUS:
    SetFocus(hwndEntry);
    return 0;

case WM_TIMER:
    GetSiderealQ;
    if(TimerFlag>0)
    {
        TimerFlag--;  
        return 0;
    }
    if(FileFlag==0 1 1  ModuleWait==0)retum 0;
    if(fgets(lpszFileEntry,79,Ffîle) = = =  NULL )
    {
        FileFlag=0;
        fclose(Ffile);
        fprintf(ErrorFile,"TCLCTRL:INTERACTIVE
");
        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);

A distributed control system for the St Andrews twin photometric telescope :  Page 228 of 335
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(File);
    FileFlag = 0;
    SetWindowText(hwnd, "TCLCTRL: UPDATING");
    hDrop = (HANDLE) wParam;
    DragQueryFile((HANDLE) wParam, 0, lpszFile, sizeof(lpszFile));
    DragFinish((HANDLE) wParam);
    if ((File = fopen(lpszFile, "rt")) == NULL) return 0;
    FileFlag = 1;
    strcpy(lpszTitle, "TCLCTRL: EXECUTING ");
    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) IParam, 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)

    
    
A distributed control system for the St Andrews twin photometric telescope : Page 230 of 335
iLength = GetWindowTextLength(hwndEntry) + 1;
GetWindowText(hwndEntry, lpszEntry, iLength);
SetWindowText(hwndEntry, "Processing...");
TranslateCommand(hwndEntry, lpszEntry);
fprintf(ErrorFile, "TCLCTRL: INTERACTIVE COMMAND < %s > 
", lpszEntry);
SetWindowText(hwndEntry, "");
break;
}

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

unsigned int TranslateCommand(HWND hwnd, char *lpszInput)
{
    char lpszCom1[20], lpszCom2[20], sl[40], lpszOutput[80];
    int d1, d2, d3, d4, d5, d6, iData;

    sscanf(lpszInput, "%s %s %d %d %d %d %d %d", &lpszCom1, &lpszCom2, &d1, &d2, &d4, &d5, &d6);
    sscanf(lpszInput, "%s %s %s", &lpszCom1, &lpszCom2, &sl);
    if(_strcmpi(lpszCom1, "DELAY") == 0)
    {
        TimerFlag = atol(lpszCom2) / 2;
        return 0;
    }
    if(_strcmpi(lpszCom1, "POINT") == 0)
    {
        if(_strcmpi(lpszCom2, "REF") == 0)
            TransmitMessage(PointRef(d1, d2, d3, d4, d5, d6));
        if(_strcmpi(lpszCom2, "OFF") == 0)
            TransmitMessage(PointOffset(d1, d2, d3, d4, d5, d6));
        if(_strcmpi(lpszCom2, "DOME") == 0)
            TransmitMessage(PointDome(d1));
        return 0;
    }
    if(_strcmpi(lpszCom1, "WAIT") == 0)
    {
        ModuleWait = 0;
        if(MessageBox(hwnd,
                        "TCLCTRL: \n\nSYSTEM SUSPENDED. CONTINUE ?",
                        "TCLCTRL MB_INFORMATION\nMB_OK") == IDOK)
            ModuleWait = 1;

        return 0;
    }
if(_strcmi(lpszCom1, "FILTER") == 0)
{
    if(_strcmi(lpszCom2, "REF") == 0)
    {
        strcpy(lpszOutput, GetMacro(s1, GetFileName("UMF_REFERENCE_FILTER")));  
        if(_strcmi(lpszOutput, "ZZ") == 0)return 0;  
        TransmitMessage(lpszOutput);  
        strcpy(varText[ID_FILREF].lpszString, s1);  
        SetWindowDisplay(ID_FILREF);  
        return 0;  
    }
    if(_strcmi(lpszCom2, "OFF") == 0)
    {
        strcpy(lpszOutput, GetMacro(s1, GetFileName("UMF_OFFSET_FILTER")));  
        if(_strcmi(lpszOutput, "ZZ") == 0)return 0;  
        TransmitMessage(lpszOutput);  
        strcpy(varText[ID_FILOFF].lpszString, s1);  
        SetWindowDisplay(ID_FILOFF);  
        return 0;  
    }
    if(_strcmi(lpszCom2, "BOTH") == 0)
    {
        strcpy(lpszOutput, GetMacro(s1, GetFileName("UMF_BOTH_FILTER")));  
        if(_strcmi(lpszOutput, "ZZ") == 0)return 0;  
        TransmitMessage(lpszOutput);  
        strcpy(varText[ID_FILREF].lpszString, s1);  
        SetWindowDisplay(ID_FILREF);  
        SetWindowDisplay(ID_FILOFF);  
    }
    return 0;  
}

if(_strcmi(lpszCom1, "APERTURE") == 0)
{
    if(_strcmi(lpszCom2, "REF") == 0)
    {
        strcpy(lpszOutput, GetMacro(s1, GetFileName("UMF_REFERENCE_APERTURE")));  
        if(_strcmi(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(sl, GetFileName("UMF_OFFSET_APERTURE")));  
  if(_strcmpi(lpszOutput, "ZZ") == 0) return 0;
  TransmitMessage(lpszOutput);
  strcpy(varText[ID_APEOFF].lpszString, si);
  SetWindowDisplay(ID_APEOFF);
  return 0;
}

if(_strcmpi(lpszCom2, "BOTH") == 0)
{
  strcpy(lpszOutput, GetMacro(sl, GetFileName("UMF_BOTH_APERTURE")));
  if(_strcmpi(lpszOutput, "ZZ") == 0) return 0;
  TransmitMessage(lpszOutput);
  strcpy(varText[ID_APEREF].lpszString, si);
  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("DIZDIZZZ");
  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("DIZDIZZZ");

A distributed control system for the St Andrews twin photometric telescope : Page 233 of 335
if(_strcmpi(lpszCom2, "DOME") == 0)
    TransmitMessage("FIZFIRZZ");
if(_strcmpi(lpszCom2, "ALL") == 0)
    TransmitMessage("AIZAIRZBIZBIRZCIZCIRZDI ZFIRZZ");
return 0;
}

if(_strcmpi(lpszCom1, "CAMERA") == 0)
{
    if((iVarData = atoi(lpszCom2)) > 0)
    {
        if(iVarData > 0 && iVarData < 5000)
        {
            strcpy(lpszOutput, "HII");
            strcat(lpszOutput, lpszCom2);
            strcat(lpszOutput, "ZZZ");
            TransmitMessage(lpszOutput);
            varText[ID_INT].iVarData = iVarData;
            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) dl;
if(_strcmpi(lpszCom2, "R.A._OFF") == 0) RaOffEnc = (unsigned int) dl;
if(_strcmpi(lpszCom2, "DEC_REF") == 0) DecRefEnc = (unsigned int) dl;
if(_strcmpi(lpszCom2, "DEC_OFF") == 0) DecOffEnc = (unsigned int) dl;
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("H1OZZ");
    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, int DecSec)
{
    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;
    }

    return 0;
}
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(RaTemp));
strcat(lpszOutput, "ZBIMGZAIZAIMPF");
strcat(lpszOutput, GetBinary(DecTemp));
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 = abs(varText[i].iData)/3600;
        iB = (abs(varText[i].iData) - abs(iA)*3600)/60;
        iC = abs(varText[i].iData) - abs(iA)*3600 - abs(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

<table>
<thead>
<tr>
<th>Filename:</th>
<th>TCLCTRL.C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Resource script for C700 compiler</td>
</tr>
</tbody>
</table>

```c
#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
```
D.6 : TCLLINK : Make file

<table>
<thead>
<tr>
<th>Filename:</th>
<th>TCLINK.MAK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Make file for C700 complier</td>
</tr>
</tbody>
</table>

TCLINK.exe : TCLINK.obj TCLINK.def TCLINK.res

link TCLINK, /align:16, NUL, /nod slibc sw libw shell,
TCLINK rc TCLINK.res

TCLINK.obj : TCLINK.c TCLINK.h TCLINK.ico
cl -c -Gsw -Ow -W2 -Zp TCLINK.c

TCLINK.res : TCLINK.rc TCLINK.h TCLINK.ico
rc -r TCLINK.rc
D.7 : TCLLINK : Definition file

<table>
<thead>
<tr>
<th>Filename:</th>
<th>TCLLINK.DEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Definition file for C700 compiler</td>
</tr>
</tbody>
</table>
D.8 : TCLINK : Include file

<table>
<thead>
<tr>
<th>Filename:</th>
<th>TCLINK.H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Include file for C700 compiler</td>
</tr>
</tbody>
</table>

//TCLINK 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 0x2BE

#endif
D.9 : TCLLINK : Source file

<table>
<thead>
<tr>
<th>Filename:</th>
<th>TCLLINK.C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Source file for C700 compiler</td>
</tr>
</tbody>
</table>

```c
// 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 lpszCmdLine, int nCmdShow)
{
    static char szAppName[] = "TCLLINK";
    HWND hwnd;
    MSG msg;
    WNDCLASS wndclass;

    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, "TCLLINK:INTERLINK",
                         "WS_OVERLAPPEDWINDOW + WS_MINIMIZE, CW_USEDEFAULT, CW_USEDEFAULT",
                         ...)
```
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", &dcbnet) < 0)
        BreakOut(hwnd, "BuildCommDCB: COM4 failed", 1);
    if(SetCommState(&dcbnet) < 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\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 + 1] = lpszInput[CursorB + 1];
        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,IpszInput,80);
        CursorA =0;
        CursorB =0;
        InterlinkFlag=TRUE;
        FileFlag=FALSE;
        fprintf(ErrorFile,
"TCLLINK:INTERLINK:MESSAGE < %s > \n",
    lpszInput);
    
    
A distributed control system for the St Andrews twin photometric telescope : Page 247 of 335
return 0;
}

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':

A distributed control system for the St Andrews twin photometric telescope : Page 248 of 335
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 : TCLINK : Resource script

<table>
<thead>
<tr>
<th>Filename:</th>
<th>TCLINK.RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Resource script for C700 complier</td>
</tr>
</tbody>
</table>

#include <windows.h>
#include <shellapi.h>
#include "TCLINK.H"

TCLINK ICON TCLINK.ICO
D.11 : TCLQUICK : Make file

<table>
<thead>
<tr>
<th>Filename:</th>
<th>TCLQUICK.MAK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Make file for C700 compiler</td>
</tr>
</tbody>
</table>

tclquick.exe : tclquick.obj tclquick.def tclquick.res

link tclquick, /align:16, NUL, /nod slibcex libw, tclquick rc tclquick.res

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

tclquick.res : tclquick.rc tclquick.h tclquick.ico
   rc -r tclquick.rc
D.12 : TCLQUICK : Definition file

<table>
<thead>
<tr>
<th>Filename:</th>
<th>TCLQUICK.DEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Definition file for C700 compiler</td>
</tr>
</tbody>
</table>

;==================================================================
; 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 compiler |

//TCLQUICK include file
#ifndef H_TCLQUICK
#define H_TCLQUICK

#define MAX_QUICK_ENTRY 15

#endif
D.14 : TCLQUICK : Source file

<table>
<thead>
<tr>
<th>Filename:</th>
<th>TCLQUICK.C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Source file for C700 compiler</td>
</tr>
</tbody>
</table>

//Telescope Quick Control Module
#include <windows.h>
#include <stdio.h>
#include <string.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.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,"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_TCLINK_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 \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 4 - tm.tmExtCalacter;Leading;
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(ErrorPile,"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, IParam);
}
D.15 : TCLQUICK : Resource script

<table>
<thead>
<tr>
<th>Filename:</th>
<th>TCLQUICK.RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Resource script for C700 complier</td>
</tr>
</tbody>
</table>

#include <windows.h>
#include "tclquick.h"

TCLQUICK ICON TCLQUICK.ico
D.16 : TCLMAN : Make file

<table>
<thead>
<tr>
<th>Filename</th>
<th>TCLMAN.MAK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Make file for C700 compiler</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Filename:</th>
<th>TCLMAN.H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Include file for C700 compiler</td>
</tr>
</tbody>
</table>

//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
//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.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(BLACK_BRUSH) ;
        wndclass.lpszMenuName = szAppName;
    }
}
wndclass.lpszClassName  = szAppName;

RegisterClass(&wndclass);

hwnd = CreateWindow(szAppName,"TCLMAN:",
WS_OVERLAPPEDWINDOW | WS_CLIPSIBLING,
0,0,0,0,NULL,NULL,hInstance,NULL);

for(i=0; i<9; i++)hwndButton[i] =
   CreateWindow("button",lpszAtt[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 Atom;
   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"),
A distributed control system for the St Andrews twin photometric telescope : Page 263 of 335
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_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, T 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;
    z4 = 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;
    z4 = 1;
    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, IParam);

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 39: return "FIZFIM1ZZZ";
case 40: return "FIZFIM0ZZZ";
case 41: return "FIZFIM4ZZZ";
case 42: return "FIZFIM2ZZZ";
case 43: return "FIZFIM0ZZZ";
case 44: 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

<table>
<thead>
<tr>
<th>Filename:</th>
<th>TCLMAN.RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>Resource script for C700 compiler</td>
</tr>
</tbody>
</table>

#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
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.

<table>
<thead>
<tr>
<th>Entry</th>
<th>T (Device type)</th>
<th>L (Location)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>WIRE</td>
<td>ELSEWHERE</td>
</tr>
<tr>
<td>1</td>
<td>CONNECTOR</td>
<td>TELESCOPE WARM ROOM</td>
</tr>
<tr>
<td>2</td>
<td>FUSE</td>
<td>TELESCOPE COLD ROOM</td>
</tr>
<tr>
<td>3</td>
<td>SWITCH</td>
<td>PUMP ROOM</td>
</tr>
<tr>
<td>4</td>
<td>TRANSFORMER</td>
<td>ISOLATOR ROOM</td>
</tr>
<tr>
<td>5</td>
<td>ACTUATOR</td>
<td>R.A. REFERENCE UNIT</td>
</tr>
<tr>
<td>6</td>
<td>SENSOR / LIMIT UNIT</td>
<td>DEC REFERENCE UNIT</td>
</tr>
<tr>
<td>7</td>
<td>ENCODERS</td>
<td>R.A. OFFSET UNIT</td>
</tr>
<tr>
<td>8</td>
<td>UNDEFINED UNIT</td>
<td>DEC OFFSET UNIT</td>
</tr>
<tr>
<td>9</td>
<td>UNDEFINED UNIT</td>
<td>DOME UNIT</td>
</tr>
</tbody>
</table>

E.1: Device list

<table>
<thead>
<tr>
<th>Code</th>
<th>Physical device</th>
<th>Code</th>
<th>Physical Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>11000</td>
<td>Spine (in warm room)</td>
<td>0x000</td>
<td>Earth strap</td>
</tr>
<tr>
<td>12000</td>
<td>Spine (in cold room)</td>
<td>00001</td>
<td>Input power cable</td>
</tr>
<tr>
<td>11001</td>
<td>Input mains connector</td>
<td>01002</td>
<td>Input power link</td>
</tr>
<tr>
<td>11011</td>
<td>Switch panel connector 1</td>
<td>34000</td>
<td>Input isolator switch</td>
</tr>
<tr>
<td>11012</td>
<td>Switch panel connector 2</td>
<td>34001</td>
<td>Isolator switch (console)</td>
</tr>
<tr>
<td>11013</td>
<td>Switch panel connector 3</td>
<td>34002</td>
<td>Isolator switch (hydraulics)</td>
</tr>
<tr>
<td>11014</td>
<td>Switch panel connector 4</td>
<td>31001</td>
<td>Single phase power switch</td>
</tr>
<tr>
<td>11015</td>
<td>Switch panel connector 5</td>
<td>31002</td>
<td>Three phase power switch</td>
</tr>
<tr>
<td>11021</td>
<td>Fuse panel connector 1</td>
<td>31003</td>
<td>Reference focus motor switch</td>
</tr>
<tr>
<td>11022</td>
<td>Fuse panel connector 2</td>
<td>31004</td>
<td>Offset focus motor switch</td>
</tr>
<tr>
<td>11023</td>
<td>Fuse panel connector 3</td>
<td>41001</td>
<td>Three phase isolator transformer</td>
</tr>
<tr>
<td>11024</td>
<td>Fuse panel connector 4</td>
<td>41002</td>
<td>50VAC transformer</td>
</tr>
<tr>
<td>11025</td>
<td>Fuse panel connector 5</td>
<td>41003</td>
<td>PMT heater transformer</td>
</tr>
<tr>
<td>11030</td>
<td>Low voltage shelf connector</td>
<td>41004</td>
<td>+12VDC transformer</td>
</tr>
<tr>
<td>11041</td>
<td>Network box mains input</td>
<td>41005</td>
<td>-12VDC (A) transformer</td>
</tr>
<tr>
<td>11042</td>
<td>Network box power out</td>
<td>41006</td>
<td>-12VDC (B) transformer</td>
</tr>
<tr>
<td>11043</td>
<td>Network box sensor inputs</td>
<td>41007</td>
<td>50VDC transformer</td>
</tr>
<tr>
<td>11044</td>
<td>Network box RS232 Data I/O</td>
<td>41008</td>
<td>24VDC transformer</td>
</tr>
<tr>
<td>11045</td>
<td>Network box display</td>
<td>22001</td>
<td>Oil pump fuse</td>
</tr>
<tr>
<td>12001</td>
<td>Clamp sensor</td>
<td>22002</td>
<td>Oil pump fuse</td>
</tr>
<tr>
<td>Address</td>
<td>Description</td>
<td>Address</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------------</td>
<td>---------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>1x011</td>
<td>Unit x: 3 phase connector</td>
<td>22002</td>
<td>Oil pump fuse</td>
</tr>
<tr>
<td>1x012</td>
<td>Unit x: Fine motor</td>
<td>22003</td>
<td>Dome shutter fuse</td>
</tr>
<tr>
<td>1x013</td>
<td>Unit x: Input limit switches</td>
<td>22004</td>
<td>Dome shutter fuse</td>
</tr>
<tr>
<td>1x014</td>
<td>Unit x: Auxiliary outputs</td>
<td>22005</td>
<td>Dome shutter fuse</td>
</tr>
<tr>
<td>1x015</td>
<td>Unit x: Unit power and comms</td>
<td>22006</td>
<td>Ra slew fuse</td>
</tr>
<tr>
<td>1x016</td>
<td>Unit x: Input encoder bank 1</td>
<td>22007</td>
<td>Ra slew fuse</td>
</tr>
<tr>
<td>1x017</td>
<td>Unit x: Input encoder bank 2</td>
<td>22008</td>
<td>Ra slew fuse</td>
</tr>
<tr>
<td>1x018</td>
<td>Unit x: Input encoder bank 3</td>
<td>22009</td>
<td>Ra clamp fuse</td>
</tr>
<tr>
<td>1x019</td>
<td>Unit x: Input encoder bank 4</td>
<td>22010</td>
<td>Ra clamp fuse</td>
</tr>
<tr>
<td>1x020</td>
<td>Unit x: Manual override 2SDM</td>
<td>22011</td>
<td>Ra clamp fuse</td>
</tr>
<tr>
<td>10030</td>
<td>Manual overide 2SDM</td>
<td>22012</td>
<td>Dec slew fuse</td>
</tr>
<tr>
<td>1x01</td>
<td>[POWER] : Power / com in</td>
<td>22013</td>
<td>Dec slew fuse</td>
</tr>
<tr>
<td>1x02</td>
<td>[POWER] : IO / clock bus</td>
<td>22014</td>
<td>Dec slew fuse</td>
</tr>
<tr>
<td>1x03</td>
<td>[POWER] : IO / clock bus</td>
<td>22015</td>
<td>Dec clamp fuse</td>
</tr>
<tr>
<td>1x04</td>
<td>[POWER] : IO / clock bus</td>
<td>22016</td>
<td>Dec clamp fuse</td>
</tr>
<tr>
<td>1x201</td>
<td>[CPU] : CPU bus</td>
<td>22017</td>
<td>Dec clamp fuse</td>
</tr>
<tr>
<td>1x301</td>
<td>[I/O] : IO bus</td>
<td>22021</td>
<td>DEC reference armature</td>
</tr>
<tr>
<td>1x302</td>
<td>[I/O] : MSB coarse encoder in</td>
<td>22022</td>
<td>DEC reference field</td>
</tr>
<tr>
<td>1x303</td>
<td>[I/O] : LSB coarse encoder in</td>
<td>22023</td>
<td>R.A. offset armature</td>
</tr>
<tr>
<td>1x304</td>
<td>[I/O] : MSB fine encoder in</td>
<td>22024</td>
<td>R.A. offset field</td>
</tr>
<tr>
<td>1x305</td>
<td>[I/O] : LSB fine encoder in</td>
<td>22025</td>
<td>DEC offset armature</td>
</tr>
<tr>
<td>1x306</td>
<td>[I/O] : Limit bank A in</td>
<td>22026</td>
<td>DEC offset field</td>
</tr>
<tr>
<td>1x307</td>
<td>[I/O] : Limit bank B in</td>
<td>62001</td>
<td>R.A. reference limit CW</td>
</tr>
<tr>
<td>1x308</td>
<td>[I/O] : Manual override</td>
<td>62002</td>
<td>R.A. reference limit CW</td>
</tr>
<tr>
<td>1x309</td>
<td>[I/O] : Limit input</td>
<td>62003</td>
<td>R.A. reference clutch in</td>
</tr>
<tr>
<td>1x310</td>
<td>[I/O] : Auxiliary output</td>
<td>62004</td>
<td>R.A. reference clamp in</td>
</tr>
<tr>
<td>1x311</td>
<td>[I/O] : Motor out A/B</td>
<td>62005</td>
<td>R.A. reference clamp out</td>
</tr>
<tr>
<td>1x312</td>
<td>[I/O] : Motor out C/D</td>
<td>62011</td>
<td>DEC reference limit CW</td>
</tr>
<tr>
<td>1x313</td>
<td>[I/O] : Timing functions</td>
<td>62012</td>
<td>DEC reference limit CCW</td>
</tr>
<tr>
<td>1x401</td>
<td>[CLOCK] : Clock bus</td>
<td>62013</td>
<td>DEC reference fine limit CW</td>
</tr>
<tr>
<td>1x402</td>
<td>[CLOCK] : Clock TTL input</td>
<td>62014</td>
<td>DEC reference fine limit CCW</td>
</tr>
<tr>
<td>1x501</td>
<td>[SERIAL] : Incremental encoder</td>
<td>62015</td>
<td>DEC reference centre detent</td>
</tr>
<tr>
<td>1x502</td>
<td>[SERIAL] : LSB out</td>
<td>62016</td>
<td>DEC clamp in</td>
</tr>
<tr>
<td>1x503</td>
<td>[SERIAL] : MSB out</td>
<td>62017</td>
<td>DEC clamp out</td>
</tr>
<tr>
<td>1x601</td>
<td>[PROTECT] : Input</td>
<td>62021</td>
<td>R.A. offset limit CW</td>
</tr>
<tr>
<td>1x602</td>
<td>[PROTECT] : Output</td>
<td>62022</td>
<td>R.A. offset limit CCW</td>
</tr>
<tr>
<td>1x701</td>
<td>[NETWORK] : Power sensor in</td>
<td>62023</td>
<td>R.A. offset centre detent</td>
</tr>
<tr>
<td>1x702</td>
<td>[NETWORK] : LED bank A out</td>
<td>62031</td>
<td>DEC offset limit CW</td>
</tr>
<tr>
<td>1x703</td>
<td>[NETWORK] : LED bank B out</td>
<td>62032</td>
<td>DEC offset limit CCW</td>
</tr>
<tr>
<td>1x704</td>
<td>[NETWORK] : Relay drive</td>
<td>62033</td>
<td>DEC offset centre detent</td>
</tr>
<tr>
<td>1x705</td>
<td>[NETWORK] : Comms / power</td>
<td>70001</td>
<td>R.A. reference coarse encoder</td>
</tr>
<tr>
<td>1x801</td>
<td>Override handset connector</td>
<td>70002</td>
<td>R.A. reference fine encoder</td>
</tr>
<tr>
<td>19811</td>
<td>Dome opto-isolator in</td>
<td>70003</td>
<td>DEC reference coarse encoder</td>
</tr>
<tr>
<td>19812</td>
<td>Dome opto-isolator out</td>
<td>70004</td>
<td>DEC reference fine encoder</td>
</tr>
</tbody>
</table>

A distributed control system for the St Andrews twin photometric telescope: Page 270 of 335
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15001</td>
<td>R.A. opto-isolator in</td>
<td>70005</td>
<td>R.A. offset incremental encoder</td>
</tr>
<tr>
<td>15002</td>
<td>R.A. opto-isolator out</td>
<td>70006</td>
<td>DEC offset incremental encoder</td>
</tr>
<tr>
<td>11031</td>
<td>Dome rotation unit CPU in</td>
<td>70007</td>
<td>Dome incremental encoder</td>
</tr>
<tr>
<td>11032</td>
<td>Dome rotation unit power</td>
<td>82001</td>
<td>Reference PMT coolers</td>
</tr>
<tr>
<td>11033</td>
<td>Dome CW/CCW valve output</td>
<td>82002</td>
<td>Offset PMT coolers</td>
</tr>
<tr>
<td>11034</td>
<td>Dump valve enable, output</td>
<td>82003</td>
<td>CCD camera control unit</td>
</tr>
<tr>
<td>11035</td>
<td>Pump motor on/off</td>
<td>82004</td>
<td>CUBE unit</td>
</tr>
<tr>
<td>13001</td>
<td>Low voltage input</td>
<td>81001</td>
<td>DCS PC (console)</td>
</tr>
<tr>
<td>12011</td>
<td>Sidereal drive door connector</td>
<td>81002</td>
<td>CUBE monitor</td>
</tr>
<tr>
<td>12012</td>
<td>Sidereal motor connector</td>
<td>81003</td>
<td>CCD camera video display</td>
</tr>
<tr>
<td>11051</td>
<td>Desk multibloc connectors</td>
<td>81004</td>
<td>Dome azimuth display</td>
</tr>
<tr>
<td>12011</td>
<td>Scope unswitched dual skt</td>
<td>81005</td>
<td>PMT desk</td>
</tr>
<tr>
<td>12012</td>
<td>Scope switch dual skt</td>
<td>81006</td>
<td>EHT power supply</td>
</tr>
<tr>
<td>52001</td>
<td>R.A. reference slew motor</td>
<td>82011</td>
<td>Reference window heater</td>
</tr>
<tr>
<td>52002</td>
<td>R.A. reference clamp</td>
<td>82013</td>
<td>Offset window heater</td>
</tr>
<tr>
<td>52003</td>
<td>R.A. reference clutch</td>
<td>81031</td>
<td>Dome rotation controller</td>
</tr>
<tr>
<td>52004</td>
<td>R.A. reference sidereal motor</td>
<td>81041</td>
<td>Sidereal drive amplifier</td>
</tr>
<tr>
<td>52011</td>
<td>Dec reference slew motor</td>
<td>85001</td>
<td>R.A. opto-isolator (sidereal i/p)</td>
</tr>
<tr>
<td>52012</td>
<td>Dec reference clamp</td>
<td>89001</td>
<td>Dome o/p opto-isolator</td>
</tr>
<tr>
<td>52013</td>
<td>Dec reference fine motor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52021</td>
<td>Ra offset fine motor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52031</td>
<td>Dec offset fine motor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52041</td>
<td>Reference focus motor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52051</td>
<td>Offset focus motor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>53061</td>
<td>Hydraulic pump motor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>53062</td>
<td>Hydraulic pump dump valve</td>
<td></td>
<td></td>
</tr>
<tr>
<td>53063</td>
<td>Hydraulic pump CW valve</td>
<td></td>
<td></td>
</tr>
<tr>
<td>53064</td>
<td>Hydraulic pump CCW valve</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### E.2: Spine Connector

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
<th>Voltage</th>
<th>Device</th>
<th>Fuse</th>
<th>Wire Label</th>
<th>Wire Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3 phase delta out: YELLOW</td>
<td>415VAC</td>
<td>Dec clamp</td>
<td>1A</td>
<td>C</td>
<td>GREEN</td>
</tr>
<tr>
<td>2</td>
<td>3 phase delta out: RED</td>
<td>415VAC</td>
<td>Dec clamp</td>
<td>1A</td>
<td>C</td>
<td>BLUE</td>
</tr>
<tr>
<td>3</td>
<td>3 phase delta out: BLUE</td>
<td>415VAC</td>
<td>Dec clamp</td>
<td>1A</td>
<td>C</td>
<td>RED</td>
</tr>
<tr>
<td>4</td>
<td>3 phase delta out: YELLOW</td>
<td>415VAC</td>
<td>Dec slew</td>
<td>1A</td>
<td>C</td>
<td>BROWN</td>
</tr>
<tr>
<td>5</td>
<td>3 phase delta out: RED</td>
<td>415VAC</td>
<td>Dec slew</td>
<td>1A</td>
<td>C</td>
<td>BLACK</td>
</tr>
<tr>
<td>6</td>
<td>3 phase delta out: BLUE</td>
<td>415VAC</td>
<td>Ra clamp</td>
<td>1A</td>
<td>C</td>
<td>WHITE</td>
</tr>
<tr>
<td>7</td>
<td>3 phase delta out: YELLOW</td>
<td>415VAC</td>
<td>Ra clamp</td>
<td>1A</td>
<td>C</td>
<td>YELLOW</td>
</tr>
<tr>
<td>8</td>
<td>3 phase delta out: RED</td>
<td>415VAC</td>
<td>Ra slew</td>
<td>1A</td>
<td>C</td>
<td>GREY</td>
</tr>
<tr>
<td>9</td>
<td>3 phase delta out: BLUE</td>
<td>415VAC</td>
<td>Ra clamp</td>
<td>1A</td>
<td>C</td>
<td>PURPLE</td>
</tr>
<tr>
<td>10</td>
<td>3 phase delta out: YELLOW</td>
<td>415VAC</td>
<td>Ra slew</td>
<td>1A</td>
<td>C</td>
<td>ORANGE</td>
</tr>
<tr>
<td>11</td>
<td>3 phase delta out: RED</td>
<td>415VAC</td>
<td>Ra slew</td>
<td>1A</td>
<td>C</td>
<td>GREEN</td>
</tr>
<tr>
<td>12</td>
<td>3 phase delta out: BLUE</td>
<td>415VAC</td>
<td>Ra slew</td>
<td>1A</td>
<td>C</td>
<td>PINK</td>
</tr>
<tr>
<td>13</td>
<td>3 phase delta out: YELLOW</td>
<td>415VAC</td>
<td>Oil pump</td>
<td>1A</td>
<td>D</td>
<td>GREEN</td>
</tr>
<tr>
<td>14</td>
<td>3 phase delta out: RED</td>
<td>415VAC</td>
<td>Oil pump</td>
<td>1A</td>
<td>D</td>
<td>BLACK</td>
</tr>
<tr>
<td>15</td>
<td>3 phase delta out: BLUE</td>
<td>415VAC</td>
<td>Oil pump</td>
<td>1A</td>
<td>D</td>
<td>WHITE</td>
</tr>
<tr>
<td>16</td>
<td>3 phase delta out: YELLOW</td>
<td>415VAC</td>
<td>Dome shutter</td>
<td>1A</td>
<td>D</td>
<td>RED</td>
</tr>
<tr>
<td>17</td>
<td>3 phase delta out: RED</td>
<td>415VAC</td>
<td>Dome shutter</td>
<td>1A</td>
<td>D</td>
<td>YELLOW</td>
</tr>
<tr>
<td>18</td>
<td>3 phase delta out: BLUE</td>
<td>415VAC</td>
<td>Dome shutter</td>
<td>1A</td>
<td>D</td>
<td>BLUE</td>
</tr>
<tr>
<td>19</td>
<td>(23 broken internally,) use 19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Field</td>
<td>12VDC (1 phase)</td>
<td>Reference DEC</td>
<td>[86501]</td>
<td>A</td>
<td>YELLOW</td>
</tr>
<tr>
<td>21</td>
<td>Field</td>
<td>0VDC RETURN</td>
<td>Reference DEC</td>
<td>[86501]</td>
<td>A</td>
<td>PINK</td>
</tr>
<tr>
<td>22</td>
<td>Armature</td>
<td>50VDC</td>
<td>Reference DEC</td>
<td>[86501]</td>
<td>A</td>
<td>BROWN</td>
</tr>
<tr>
<td>23</td>
<td>Armature</td>
<td>0VDC</td>
<td>Reference DEC</td>
<td>[86501]</td>
<td>A</td>
<td>BLUE</td>
</tr>
<tr>
<td>24</td>
<td>Totem power</td>
<td>+12VDC</td>
<td>Reference DEC</td>
<td>[86501]</td>
<td>A</td>
<td>B</td>
</tr>
</tbody>
</table>

A distributed control system for the St Andrews twin photometric telescope: Page 272 of 335
<table>
<thead>
<tr>
<th></th>
<th>Totem power</th>
<th>Field</th>
<th>Armature</th>
<th>Totem power</th>
<th>Field</th>
<th>Armature</th>
<th>Totem power</th>
<th>Field</th>
<th>Armature</th>
<th>Totem power</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>0VDC</td>
<td>Reference DEC</td>
<td>[86501]</td>
<td>B</td>
<td>GREY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>-12VDC</td>
<td>Reference DEC</td>
<td>[86501]</td>
<td>1A</td>
<td>B</td>
<td>PINK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>12VDC (1 phase)</td>
<td>Offset R.A.</td>
<td>[87501]</td>
<td>1A</td>
<td>A</td>
<td>WHITE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>0VDC RETURN</td>
<td>Offset R.A.</td>
<td>[87501]</td>
<td>A</td>
<td>ORANGE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>50VDC</td>
<td>Offset R.A.</td>
<td>[87501]</td>
<td>1A</td>
<td>A</td>
<td>DARK GREEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>0VDC</td>
<td>Offset R.A.</td>
<td>[87501]</td>
<td>A</td>
<td>LIGHT GREEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>+12VDC</td>
<td>Offset R.A.</td>
<td>[87501]</td>
<td>1A</td>
<td>B</td>
<td>ORANGE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>0VDC</td>
<td>Offset R.A.</td>
<td>[87501]</td>
<td>B</td>
<td>PURPLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>-12VDC</td>
<td>Offset R.A.</td>
<td>[87501]</td>
<td>1A</td>
<td>B</td>
<td>LIGHT GREEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>12VDC (1 phase)</td>
<td>Offset DEC</td>
<td>[88501]</td>
<td>1A</td>
<td>A</td>
<td>RED</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>0VDC RETURN</td>
<td>Offset DEC</td>
<td>[88501]</td>
<td>A</td>
<td>PURPLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>50VDC</td>
<td>Offset DEC</td>
<td>[88501]</td>
<td>1A</td>
<td>A</td>
<td>BLACK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>0VDC</td>
<td>Offset DEC</td>
<td>[88501]</td>
<td>A</td>
<td>GREY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>+12VDC</td>
<td>Offset DEC</td>
<td>[88501]</td>
<td>1A</td>
<td>B</td>
<td>RED</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>0VDC</td>
<td>Offset DEC</td>
<td>[88501]</td>
<td>B</td>
<td>BLACK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>-12VDC</td>
<td>Offset DEC</td>
<td>[88501]</td>
<td>1A</td>
<td>B</td>
<td>BLUE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>nc</td>
<td>Reference DEC unit</td>
<td>[86000]</td>
<td></td>
<td>Discrete</td>
<td>RED</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>+20VDC</td>
<td>Reference DEC unit</td>
<td>[86000]</td>
<td>Discrete</td>
<td>BLACK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>0VDC</td>
<td>Reference DEC unit</td>
<td>[87000]</td>
<td>Discrete</td>
<td>RED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>+20VDC</td>
<td>Offset R.A. unit</td>
<td>[87000]</td>
<td>Discrete</td>
<td>BLACK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>0VDC RETURN</td>
<td>Offset R.A. unit</td>
<td>[88000]</td>
<td>Discrete</td>
<td>RED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>+20VDC</td>
<td>Offset DEC unit</td>
<td>[88000]</td>
<td>Discrete</td>
<td>BLACK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>0VDC</td>
<td>Offset DEC unit</td>
<td>[88000]</td>
<td>Discrete</td>
<td>RED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>+20VDC</td>
<td>Counter unit</td>
<td></td>
<td>Discrete</td>
<td>BLACK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>0VDC</td>
<td>Counter unit</td>
<td></td>
<td>Discrete</td>
<td>RED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>+20VDC</td>
<td>Tertiary unit</td>
<td></td>
<td>Discrete</td>
<td>BLACK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>0VDC</td>
<td>Tertiary unit</td>
<td></td>
<td>Discrete</td>
<td>RED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>+20VDC</td>
<td>Tertiary unit</td>
<td></td>
<td>Discrete</td>
<td>BLACK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Max connected side: 52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>Reference window heater</td>
<td>6.2VAC</td>
<td>[82011]</td>
<td>Mains cable</td>
<td>BLUE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>Reference window heater</td>
<td>6.2VAC</td>
<td>[82011]</td>
<td>Mains cable</td>
<td>BROWN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>Offset window heater</td>
<td>6.2VAC</td>
<td>[82012]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>Offset window heater</td>
<td>6.2VAC</td>
<td>[82012]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>Earth</td>
<td></td>
<td>[81021]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>Line driver</td>
<td>0VDC</td>
<td>[81021]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>Line driver</td>
<td>+15VDC</td>
<td>[81021]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>Reference focus BLUE</td>
<td>50VAC</td>
<td>[52041]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>Reference focus RED</td>
<td>50VAC</td>
<td>[52041]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>Reference focus YELLOW</td>
<td>50VAC</td>
<td>[52041]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>Offset focus BLUE</td>
<td>50VAC</td>
<td>[52051]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>Offset focus RED</td>
<td>50VAC</td>
<td>[52051]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>Offset focus YELLOW</td>
<td>50VAC</td>
<td>[52051]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>Oil heater</td>
<td>240VAC</td>
<td>[82072]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>Oil heater</td>
<td>0VAC</td>
<td>[82072]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>Oil heater</td>
<td>0VAC (in)</td>
<td>[82072]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>Clutch POWER</td>
<td>+24VDC</td>
<td>Clutch</td>
<td>[52003]</td>
<td>Discrete</td>
<td>RED</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>Clutch RETURN</td>
<td>0VDC</td>
<td>Clutch</td>
<td>[52003]</td>
<td>Discrete</td>
<td>BLACK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>Microcontroller power</td>
<td>+20VDC</td>
<td>R.A. Reference unit</td>
<td>[85000]</td>
<td>Discrete</td>
<td>RED</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>Microcontroller power</td>
<td>0VDC</td>
<td>R.A. Reference unit</td>
<td>[85000]</td>
<td>Discrete</td>
<td>BLACK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>Reserved sidereal drive</td>
<td>+32VDC</td>
<td></td>
<td>[81041]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>Reserved sidereal drive</td>
<td>0VDC</td>
<td></td>
<td>[81041]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>Reserved sidereal drive</td>
<td>-32VDC</td>
<td></td>
<td>[81041]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>Reserved sidereal drive</td>
<td>2.5V (Triag)</td>
<td></td>
<td>[81041]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>Reserved sidereal drive</td>
<td>2.5V (Triag)</td>
<td></td>
<td>[81041]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A distributed control system for the St Andrews twin photometric telescope: Page 274 of 335
<table>
<thead>
<tr>
<th>86</th>
<th>Reserved sidereal drive</th>
</tr>
</thead>
<tbody>
<tr>
<td>117</td>
<td>Auxiliary RETURN</td>
</tr>
<tr>
<td>118</td>
<td>Auxiliary POWER (Switched)</td>
</tr>
<tr>
<td>119</td>
<td>Auxiliary RETURN</td>
</tr>
<tr>
<td>120</td>
<td>Auxiliary POWER (Unswitched)</td>
</tr>
</tbody>
</table>
### E.3 : CPU memory and I/O logic : GAL pinout

<table>
<thead>
<tr>
<th>Pin</th>
<th>I/O</th>
<th>Description</th>
<th>Pin</th>
<th>I/O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>I</td>
<td>10MHz CLK</td>
<td>20</td>
<td>-</td>
<td>5V</td>
</tr>
<tr>
<td>2</td>
<td>I</td>
<td>A11</td>
<td>19</td>
<td>O</td>
<td>BEN : Data bus enable</td>
</tr>
<tr>
<td>3</td>
<td>I</td>
<td>A12</td>
<td>18</td>
<td>O</td>
<td>IRD : I/O Port read</td>
</tr>
<tr>
<td>4</td>
<td>I</td>
<td>A13</td>
<td>17</td>
<td>O</td>
<td>IWR : I/O Port write</td>
</tr>
<tr>
<td>5</td>
<td>I</td>
<td>A14</td>
<td>16</td>
<td>O</td>
<td>EOU : EPROM /OE</td>
</tr>
<tr>
<td>6</td>
<td>I</td>
<td>A15</td>
<td>15</td>
<td>O</td>
<td>EEN : EPROM /CE</td>
</tr>
<tr>
<td>7</td>
<td>I</td>
<td>/MREQ</td>
<td>14</td>
<td>O</td>
<td>RWR : RAM /WR</td>
</tr>
<tr>
<td>8</td>
<td>I</td>
<td>/WR</td>
<td>13</td>
<td>O</td>
<td>ROU : RAM /OE</td>
</tr>
<tr>
<td>9</td>
<td>I</td>
<td>/RD</td>
<td>12</td>
<td>O</td>
<td>REN : RAM /CE</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>0V</td>
<td>11</td>
<td>I</td>
<td>GAL chip enable</td>
</tr>
</tbody>
</table>

### E.4 : Motor limit : GAL pinout

<table>
<thead>
<tr>
<th>Pin</th>
<th>I/O</th>
<th>Description</th>
<th>Pin</th>
<th>I/O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>I</td>
<td>NOT USED</td>
<td>20</td>
<td>-</td>
<td>5V</td>
</tr>
<tr>
<td>2</td>
<td>I</td>
<td>MEA : Motor A en.</td>
<td>19</td>
<td>O</td>
<td>NOT USED.</td>
</tr>
<tr>
<td>3</td>
<td>I</td>
<td>MEB : Motor B en.</td>
<td>18</td>
<td>O</td>
<td>NOT USED.</td>
</tr>
<tr>
<td>5</td>
<td>I</td>
<td>LAH : CCW limit A.</td>
<td>16</td>
<td>O</td>
<td>BEOUT : Motor B out.</td>
</tr>
<tr>
<td>6</td>
<td>I</td>
<td>MDA : Motor A dir.</td>
<td>15</td>
<td>O</td>
<td>NOT USED</td>
</tr>
<tr>
<td>7</td>
<td>I</td>
<td>MDB : Motor B dir.</td>
<td>14</td>
<td>O</td>
<td>NOT USED</td>
</tr>
<tr>
<td>8</td>
<td>I</td>
<td>LBL : CW limit B.</td>
<td>13</td>
<td>O</td>
<td>ADOUT : Motor A dir.</td>
</tr>
<tr>
<td>9</td>
<td>I</td>
<td>LBH : CCW limit B.</td>
<td>12</td>
<td>O</td>
<td>AEOUT : Motor A out</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>0V</td>
<td>11</td>
<td>I</td>
<td></td>
</tr>
</tbody>
</table>

### E.5 : Bus control : GAL pinout

<table>
<thead>
<tr>
<th>Pin</th>
<th>I/O</th>
<th>Description</th>
<th>Pin</th>
<th>I/O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>I</td>
<td>Sensor Ok</td>
<td>20</td>
<td>-</td>
<td>5V</td>
</tr>
<tr>
<td>2</td>
<td>I</td>
<td>CPU power</td>
<td>19</td>
<td>O</td>
<td>System OK</td>
</tr>
<tr>
<td>3</td>
<td>I</td>
<td>Standby power</td>
<td>18</td>
<td>O</td>
<td>415V Relay Drive</td>
</tr>
<tr>
<td>4</td>
<td>I</td>
<td>415V in</td>
<td>17</td>
<td>O</td>
<td>CPU Relay Drive</td>
</tr>
<tr>
<td>5</td>
<td>I</td>
<td>Reset /Resume</td>
<td>16</td>
<td>O</td>
<td>CPU Error</td>
</tr>
<tr>
<td>6</td>
<td>I</td>
<td>Stop /Resume</td>
<td>15</td>
<td>O</td>
<td>Bus Error</td>
</tr>
<tr>
<td>7</td>
<td>I</td>
<td>Line return</td>
<td>14</td>
<td>O</td>
<td>System Error</td>
</tr>
<tr>
<td>8</td>
<td>I</td>
<td>TX / RX</td>
<td>13</td>
<td>O</td>
<td>System Reset</td>
</tr>
<tr>
<td>9</td>
<td>I</td>
<td>TX request</td>
<td>12</td>
<td>O</td>
<td>Line Error</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>0V</td>
<td>11</td>
<td>I</td>
<td>CPU Error in</td>
</tr>
</tbody>
</table>

CAUTION : Programming this gal will cause unpredictable results in system wide control.

---

A distributed control system for the St Andrews twin photometric telescope : Page 276 of 335
### E.6: Bus control: Terminal block connectors

<table>
<thead>
<tr>
<th>Pin</th>
<th>Mains Power [11041]</th>
<th>Power out [11042]</th>
<th>Sensor inputs [11043]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>240VAC LIVE</td>
<td>CPU power out (15V)</td>
<td>0V</td>
</tr>
<tr>
<td>2</td>
<td>240VAC NEUTRAL</td>
<td>Front Panel Fuse</td>
<td>Relay Disable</td>
</tr>
<tr>
<td>3</td>
<td>GROUND</td>
<td>CPU power Out (0V)</td>
<td>Line return</td>
</tr>
<tr>
<td>4</td>
<td>nc</td>
<td>Front Panel Fuse</td>
<td>Stop / Resume</td>
</tr>
<tr>
<td>5</td>
<td>nc</td>
<td>GROUND case</td>
<td>Reset</td>
</tr>
<tr>
<td>6</td>
<td>nc</td>
<td>nc</td>
<td>415V on</td>
</tr>
<tr>
<td>7</td>
<td>nc</td>
<td>nc</td>
<td>Standby Power</td>
</tr>
<tr>
<td>8</td>
<td>nc</td>
<td>nc</td>
<td>CPU Power</td>
</tr>
<tr>
<td>9</td>
<td>nc</td>
<td>nc</td>
<td>/ Sensor OK</td>
</tr>
<tr>
<td>10</td>
<td>nc</td>
<td>TXRX</td>
<td>nc</td>
</tr>
<tr>
<td>11</td>
<td>nc</td>
<td>TXREQ</td>
<td>nc</td>
</tr>
<tr>
<td>12</td>
<td>nc</td>
<td>Signal ground</td>
<td>5V</td>
</tr>
</tbody>
</table>

### E.7: Bus control: RS232 pinout

<table>
<thead>
<tr>
<th>Pin</th>
<th>RS 232 Port [11044]</th>
<th>IBM display [11045]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>nc</td>
<td>nc</td>
</tr>
<tr>
<td>2</td>
<td>RX in</td>
<td>'Bus error'</td>
</tr>
<tr>
<td>3</td>
<td>TX out</td>
<td>'Line error.'</td>
</tr>
<tr>
<td>4</td>
<td>RTS</td>
<td>'System error'.</td>
</tr>
<tr>
<td>5</td>
<td>CTS</td>
<td>'Stop / resume'.</td>
</tr>
<tr>
<td>6</td>
<td>Signal ground</td>
<td>'CPU error'</td>
</tr>
<tr>
<td>7</td>
<td>DTR</td>
<td>'System reset'</td>
</tr>
<tr>
<td>8</td>
<td>nc</td>
<td>nc</td>
</tr>
<tr>
<td>9</td>
<td>nc</td>
<td>nc</td>
</tr>
<tr>
<td>10</td>
<td>nc</td>
<td>LED return</td>
</tr>
<tr>
<td>11</td>
<td>nc</td>
<td>nc</td>
</tr>
<tr>
<td>12</td>
<td>nc</td>
<td>nc</td>
</tr>
<tr>
<td>13</td>
<td>nc</td>
<td>nc</td>
</tr>
<tr>
<td>14</td>
<td>nc</td>
<td>nc.</td>
</tr>
<tr>
<td>15</td>
<td>nc</td>
<td>'TX request'</td>
</tr>
<tr>
<td>16</td>
<td>nc</td>
<td>'TXRX'</td>
</tr>
<tr>
<td>17</td>
<td>nc</td>
<td>'System ok'</td>
</tr>
<tr>
<td>18</td>
<td>nc</td>
<td>'415V power'</td>
</tr>
<tr>
<td>19</td>
<td>nc</td>
<td>'CPU power'</td>
</tr>
<tr>
<td>20</td>
<td>nc</td>
<td>'Standby power'</td>
</tr>
<tr>
<td>21</td>
<td>nc</td>
<td>nc</td>
</tr>
<tr>
<td>22</td>
<td>nc</td>
<td>nc</td>
</tr>
<tr>
<td>23</td>
<td>nc</td>
<td>LED return</td>
</tr>
<tr>
<td>24</td>
<td>nc</td>
<td>nc</td>
</tr>
<tr>
<td>25</td>
<td>nc</td>
<td>nc.</td>
</tr>
</tbody>
</table>

Note that it is here that the TX / RX signals are crossed over.
E.8 : Bus control : Display front end

<table>
<thead>
<tr>
<th>STANDBY POWER</th>
<th>CPU POWER</th>
<th>415V POWER</th>
<th>SYSTEM OK</th>
<th>TX RX</th>
<th>TX REQUEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM RESET</td>
<td>CPU ERROR</td>
<td>STOP RESUME</td>
<td>SYSTEM ERROR</td>
<td>LINE ERROR</td>
<td>BUS ERROR</td>
</tr>
</tbody>
</table>

E.9 : DC axial motors

<table>
<thead>
<tr>
<th>DC axial motors.</th>
<th>[52013] [52021] [52031]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application:</td>
<td>DEC reference R.A. &amp; DEC offset</td>
</tr>
<tr>
<td>manufacturer:</td>
<td>Evershed and Vignoles Ltd. London W4</td>
</tr>
<tr>
<td>Type:</td>
<td>FAG 101/N4/BD</td>
</tr>
<tr>
<td>Field Voltage:</td>
<td>12V / winding</td>
</tr>
<tr>
<td>armature voltage:</td>
<td>12V</td>
</tr>
</tbody>
</table>

E.10 : Offset motor wiring

<table>
<thead>
<tr>
<th></th>
<th>OFFSET DEC (GRAY Multicore)</th>
<th>OFFSET R.A. (BLACK Multicore)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCW LIMIT</td>
<td>DARK GREEN</td>
<td>DARK GREEN</td>
</tr>
<tr>
<td>CCW LIMIT</td>
<td>LIGHT BLUE</td>
<td>LIGHT GREEN</td>
</tr>
<tr>
<td>CW LIMIT</td>
<td>PINK</td>
<td>PINK</td>
</tr>
<tr>
<td>CW LIMIT</td>
<td>ORANGE</td>
<td>ORANGE</td>
</tr>
<tr>
<td>FIELD 1</td>
<td>BROWN</td>
<td>BROWN</td>
</tr>
<tr>
<td>FIELD 2</td>
<td>BLACK</td>
<td>BLACK</td>
</tr>
<tr>
<td>FIELD COMMON</td>
<td>BLUE</td>
<td>BLUE</td>
</tr>
<tr>
<td>ARMATURE</td>
<td>RED</td>
<td>RED</td>
</tr>
<tr>
<td>ARMATURE</td>
<td>PURPLE</td>
<td>PURPLE</td>
</tr>
<tr>
<td>ZERO POT V+</td>
<td>YELLOW</td>
<td>YELLOW</td>
</tr>
<tr>
<td>ZERO POT V-</td>
<td>GRAY</td>
<td>GRAY</td>
</tr>
<tr>
<td>ZERO POT WIPER</td>
<td>WHITE</td>
<td>WHITE</td>
</tr>
</tbody>
</table>

E.11 : Reference R.A. : Slew motor

| Slew motor [52001].                     | ANSLADO |
| 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 |

A distributed control system for the St Andrews twin photometric telescope : Page 278 of 335
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))

<table>
<thead>
<tr>
<th>Multibloc [12001]</th>
<th>upper sensor</th>
<th>Lower sensor</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>blue</td>
<td></td>
<td>red</td>
</tr>
<tr>
<td>2</td>
<td>red</td>
<td>red</td>
<td>green</td>
</tr>
<tr>
<td>3</td>
<td>white</td>
<td>blue</td>
<td>black</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>white</td>
<td></td>
</tr>
<tr>
<td>5 -nc-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 -nc-</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Clamp motor [52002].

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
clamp gearbox.
manufacturer ROSSI MOTORDIDOTTORI
type MRV50PI1P
power 0.18 KW
rate 29.8 RPM

Clamp microswitch [62001].

Magnet RS 338-759
Sensors RS 339-213
### E.13 : Reference R.A. : Clutch unit

<table>
<thead>
<tr>
<th><strong>Clutch [52003]</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturer</td>
<td>MATRIX</td>
</tr>
<tr>
<td>Operating voltage</td>
<td>24VDC</td>
</tr>
<tr>
<td>RS stock no.</td>
<td>338-743</td>
</tr>
</tbody>
</table>

**Clutch gearbox.**

<table>
<thead>
<tr>
<th><strong>Manufacturer</strong></th>
<th>ROSSI MOTTORIDUTTORI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model</strong></td>
<td>MR 2V 85 P11A</td>
</tr>
<tr>
<td><strong>Built</strong></td>
<td>82</td>
</tr>
<tr>
<td>P1</td>
<td>0.12kW</td>
</tr>
<tr>
<td>n2</td>
<td>0.92 /min</td>
</tr>
</tbody>
</table>

### E.14 : Sidereal drive

<table>
<thead>
<tr>
<th><strong>Fine / Sidereal Drive [52004]</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturer</td>
<td>UNIMATIC ???</td>
</tr>
<tr>
<td>Dealer</td>
<td>UNIMATIC ENGINEERS Ltd</td>
</tr>
<tr>
<td></td>
<td>GRANVILLE ROAD WORKS</td>
</tr>
<tr>
<td></td>
<td>LONDON, ENGLAND</td>
</tr>
<tr>
<td></td>
<td>NW2 2LN</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>STEPPER</td>
</tr>
<tr>
<td><strong>Voltage</strong></td>
<td>32V</td>
</tr>
</tbody>
</table>
### E.15: Mains power back connectors

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>240VAC 1 phase switched out</td>
<td>240VAC 1 phase unswitched out</td>
<td>Main Dec armature in</td>
<td>240VAC 1 phase switched out</td>
<td>415VAC 3 phase delta in: BLUE</td>
</tr>
<tr>
<td>2</td>
<td>240VAC 1 phase unswitched out</td>
<td>1 phase NEUTRAL</td>
<td>Main Dec armature out</td>
<td>1 phase NEUTRAL</td>
<td>415VAC 3 phase delta in: RED</td>
</tr>
<tr>
<td>3</td>
<td>1 phase NEUTRAL out</td>
<td>240VAC 1 phase unswitched out</td>
<td>Offset Ra armature in</td>
<td>240VAC 1 phase switched out</td>
<td>415VAC 3 phase delta in: YELLOW</td>
</tr>
<tr>
<td>4</td>
<td>240VAC 3 phase BLUE switched out</td>
<td>1 phase NEUTRAL</td>
<td>Offset Ra armature out</td>
<td>1 phase NEUTRAL</td>
<td>415VAC 3 phase delta out: BLUE Oil pump 1A</td>
</tr>
<tr>
<td>5</td>
<td>240VAC 3 phase RED switched out</td>
<td>240VAC 1 phase unswitched out</td>
<td>Offset Dec armature in</td>
<td>240VAC 1 phase switched out</td>
<td>415VAC 3 phase delta out: RED Oil pump 1A</td>
</tr>
<tr>
<td>6</td>
<td>240VAC 3 phase YELLOW switched out</td>
<td>1 phase NEUTRAL</td>
<td>Offset Dec armature out</td>
<td>1 phase NEUTRAL</td>
<td>415VAC 3 phase delta out: YELLOW Oil pump 1A</td>
</tr>
<tr>
<td>7</td>
<td>3 phase NEUTRAL out</td>
<td>240VAC 1 phase unswitched out</td>
<td>240VAC 1 phase switched out</td>
<td>240VAC 1 phase switched out</td>
<td>415VAC 3 phase delta out: YELLOW Oil pump 1A</td>
</tr>
<tr>
<td>8</td>
<td>240VAC 3 phase BLUE in</td>
<td>1 phase NEUTRAL</td>
<td>24VDC out</td>
<td>1 phase NEUTRAL</td>
<td>nc</td>
</tr>
<tr>
<td>9</td>
<td>240VAC 3 phase RED in</td>
<td>240VAC 1 phase unswitched out</td>
<td>24VDC RETURN</td>
<td>240VAC 1 phase switched out</td>
<td>415VAC 3 phase delta out: BLUE Shutter 1A</td>
</tr>
<tr>
<td>10</td>
<td>240VAC 3 phase YELLOW in</td>
<td>1 phase NEUTRAL</td>
<td>+12VDC</td>
<td>1 phase NEUTRAL</td>
<td>415VAC 3 phase delta out: RED Shutter 1A</td>
</tr>
<tr>
<td>11</td>
<td>3 phase NEUTRAL in</td>
<td>240VAC 1 phase unswitched out</td>
<td>0VDC</td>
<td>240VAC 1 phase switched out</td>
<td>415VAC 3 phase delta out: YELLOW Shutter 1A</td>
</tr>
<tr>
<td>12</td>
<td>Earth</td>
<td>1 phase NEUTRAL</td>
<td>-12VDC</td>
<td>1 phase NEUTRAL</td>
<td>nc</td>
</tr>
</tbody>
</table>

A distributed control system for the St Andrews twin photometric telescope: Page 281 of 335
### E.16: Fuse panel back connectors

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>415VAC 3 phase delta in BLUE</td>
<td>415VAC 3 phase delta out 1A BLUE : Ra slew</td>
<td>nc</td>
<td>12VDC in BLUE</td>
<td>240VAC 1 phase in PINK : Switched</td>
</tr>
<tr>
<td>2</td>
<td>415VAC 3 phase delta in RED</td>
<td>415VAC 3 phase delta out 1A RED : Ra slew</td>
<td>nc</td>
<td>0VDC in BLACK</td>
<td>1 phase return in BLACK</td>
</tr>
<tr>
<td>3</td>
<td>415VAC 3 phase delta in YELLOW</td>
<td>415VAC 3 phase delta out 1A YELLOW : Ra slew</td>
<td>nc</td>
<td>12VDC out 1A BLUE : Offset ra</td>
<td>240VAC 1 phase in PINK : Unswitched</td>
</tr>
<tr>
<td>4</td>
<td>nc</td>
<td>415VAC 3 phase delta out 1A BLUE : Ra clamp</td>
<td>nc</td>
<td>0VDC out 1A BLACK : Offset ra</td>
<td>1 phase return in BLACK</td>
</tr>
<tr>
<td>5</td>
<td>nc</td>
<td>415VAC 3 phase delta out 1A RED : Ra clamp</td>
<td>nc</td>
<td>12VDC out 1A BLUE : Offset dec</td>
<td>240VAC 1 phase out 1A PINK : Desk switched</td>
</tr>
<tr>
<td>6</td>
<td>nc</td>
<td>415VAC 3 phase delta out 1A YELLOW : Ra clamp</td>
<td>nc</td>
<td>0VDC out 1A BLACK : Offset dec</td>
<td>1 phase return out BLACK</td>
</tr>
<tr>
<td>7</td>
<td>nc</td>
<td>415VAC 3 phase delta out 1A BLUE : Dec slew</td>
<td>nc</td>
<td>12VDC out 1A BLUE : Main dec</td>
<td>240VAC 1 phase out 1A PINK : Scope switched</td>
</tr>
<tr>
<td>8</td>
<td>nc</td>
<td>415VAC 3 phase delta out 1A RED : Dec slew</td>
<td>nc</td>
<td>0VDC out 1A BLACK : Main dec</td>
<td>1 phase return out BLACK</td>
</tr>
<tr>
<td>9</td>
<td>nc</td>
<td>415VAC 3 phase delta out 1A YELLOW : Dec slew</td>
<td>nc</td>
<td>50VAC 1 phase in PINK</td>
<td>240VAC 1 phase out 1A PINK : Desk unswitched</td>
</tr>
<tr>
<td>10</td>
<td>nc</td>
<td>415VAC 3 phase delta out 1A BLUE : Dec clamp</td>
<td>nc</td>
<td>50VAC 1 phase in PINK</td>
<td>1 phase return out BLACK</td>
</tr>
<tr>
<td>11</td>
<td>nc</td>
<td>415VAC 3 phase delta out 1A RED : Dec clamp</td>
<td>nc</td>
<td>50VAC 1 phase out 1A PINK : focus motors</td>
<td>240VAC 1 phase out 1A PINK : Scope unswitched</td>
</tr>
<tr>
<td>12</td>
<td>nc</td>
<td>415VAC 3 phase delta out 1A YELLOW : Dec clamp</td>
<td>nc</td>
<td>50VAC 1 phase out 1A PINK : focus motors</td>
<td>1 phase return out BLACK</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Low voltage connector</th>
<th>[11030]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>240VAC 1 phase LIVE unswitched in</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>240VAC NEUTRAL</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>240VAC 1 phase LIVE switched in</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>240VAC NEUTRAL</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>EARTH</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>EARTH</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>+12V out</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>return</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>-12V out</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>return</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>-12V out</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>return</td>
<td></td>
</tr>
</tbody>
</table>
### E.18: R.A. Reference Output

<table>
<thead>
<tr>
<th>Bit</th>
<th>FFF0h</th>
<th>FFF1h</th>
<th>FFF2h</th>
<th>FFF3h</th>
<th>FFF4h</th>
<th>FFF5h</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATE</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>SID GUIDE-</td>
<td>CANCEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SID SET+</td>
<td>CANCEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SID SET-</td>
<td>CANCEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>SID GUIDE+</td>
<td>CANCEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>CLUTCH OUT</td>
<td>CLUTCH IN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>CLAMP ON</td>
<td>CLAMP OFF</td>
<td>CLAMP OUT</td>
<td>CLAMP IN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>SLEW ON</td>
<td>SLEW OFF</td>
<td>SLEW CCW</td>
<td>SLEW CW</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### E.19: R.A. Reference Inputs

<table>
<thead>
<tr>
<th>Bit</th>
<th>FFF0h [15302]</th>
<th>FFF1h [15303]</th>
<th>FFF2h [15304]</th>
<th>FFF3h [15305]</th>
<th>FFF4h [15306]</th>
<th>FFF5h [15307]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>low encoder</td>
<td>low encoder</td>
<td>1</td>
<td>high encoder</td>
<td>ccw limit [1]</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>low encoder</td>
<td>low encoder</td>
<td>1</td>
<td>high encoder</td>
<td>cw limit [1]</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>low encoder</td>
<td>low encoder</td>
<td>1</td>
<td>high encoder</td>
<td>clamp out [1]</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>low encoder</td>
<td>1</td>
<td>1</td>
<td>high encoder</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>low encoder</td>
<td>1</td>
<td>1</td>
<td>high encoder</td>
<td>ccw limit [1]</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>low encoder</td>
<td>clutch in [0]</td>
<td>1</td>
<td>1</td>
<td>cw limit [1]</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>low encoder</td>
<td>clamp overload [0]</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>low encoder</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
### E.20: DEC. reference output

<table>
<thead>
<tr>
<th>STATE</th>
<th>FFF0h</th>
<th>FFF1h</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>ENABLE</td>
<td>DISABLE</td>
</tr>
<tr>
<td>2</td>
<td>FINE CCW</td>
<td>FINE CW</td>
</tr>
<tr>
<td>3</td>
<td>BRAKE ON</td>
<td>BRAKE OFF</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>CLAMP ON</td>
<td>CLAMP OFF</td>
</tr>
<tr>
<td>7</td>
<td>SLEW ON</td>
<td>SLEW OFF</td>
</tr>
</tbody>
</table>

### E.21: DEC. reference inputs

<table>
<thead>
<tr>
<th>Bit</th>
<th>FFF0h [16302]</th>
<th>FFF1h [16303]</th>
<th>FFF2h [16304]</th>
<th>FFF3h [16305]</th>
<th>FFF4h [16306]</th>
<th>FFF5h [16307]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>high encoder</td>
<td>1</td>
<td>low encoder</td>
<td>low encoder</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>high encoder</td>
<td>1</td>
<td>low encoder</td>
<td>low encoder</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>high encoder</td>
<td>1</td>
<td>low encoder</td>
<td>low encoder</td>
<td>fine low 1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>high encoder</td>
<td>1</td>
<td>low encoder</td>
<td>1</td>
<td>fine high 1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>high encoder</td>
<td>1</td>
<td>low encoder</td>
<td>1</td>
<td>clamp out 1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>high encoder</td>
<td>1</td>
<td>low encoder</td>
<td>1</td>
<td>clamp in 1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>low encoder</td>
<td>1</td>
<td>CCW limit 1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>fine centre</td>
<td>low encoder</td>
<td>1</td>
<td>CW limit 1</td>
<td>1</td>
</tr>
</tbody>
</table>
### E.22: R.A. offset outputs

<table>
<thead>
<tr>
<th>Bit</th>
<th>Bit 0 FFF0h</th>
<th>Bit 1 FFF1h</th>
<th>Bit 2 FFF2h</th>
<th>Bit 3 FFF3h</th>
<th>Bit 4 FFF4h</th>
<th>Bit 5 FFF5h</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATE</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- DIR RA+: Directional Right Ascension +
- DIR RA-: Directional Right Ascension -
- BRAKE OFF: Brake Off
- BRAKE ON: Brake On
- UNUSED: Unused
- DISABLE: Disable
- ENABLE: Enable

### E.23: R.A. offset inputs

<table>
<thead>
<tr>
<th>Bit</th>
<th>Bit 0 FFF0h [17302]</th>
<th>Bit 1 FFF1h [17303]</th>
<th>Bit 2 FFF2h [17304]</th>
<th>Bit 3 FFF3h [17305]</th>
<th>Bit 4 FFF4h [17306]</th>
<th>Bit 5 FFF5h [17307]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Centre</td>
<td>1</td>
<td>Limit CW [0]</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Phase 0</td>
<td>1</td>
<td>Limit CCW [0]</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Phase 90</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>SYNC [1]</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### E.24: DEC. offset outputs

<table>
<thead>
<tr>
<th>Bit</th>
<th>Bit 0 FFF0h</th>
<th>Bit 1 FFF1h</th>
<th>Bit 2 FFF2h</th>
<th>Bit 3 FFF3h</th>
<th>Bit 4 FFF4h</th>
<th>Bit 5 FFF5h</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATE</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- DIR DEC+: Directional Declination +
- DIR DEC-: Directional Declination -
- BRAKE OFF: Brake Off
- BRAKE ON: Brake On
- UNUSED: Unused
- DISABLE: Disable
- ENABLE: Enable

---

A distributed control system for the St Andrews twin photometric telescope: Page 286 of 335
### E.25: DEC. offset inputs

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Centre</td>
<td>1</td>
<td>Limit CW [0]</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Phase 0</td>
<td>1</td>
<td>Limit CCW [0]</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Phase 90</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>SYNC [1]</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### E.26: Dome outputs

<table>
<thead>
<tr>
<th>STATE</th>
<th>FFF0h</th>
<th>FFF1h</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SLEW CCW</td>
<td>INHIBIT</td>
</tr>
<tr>
<td>1</td>
<td>F. SET CCW</td>
<td>INHIBIT</td>
</tr>
<tr>
<td>2</td>
<td>S. SET CCW</td>
<td>INHIBIT</td>
</tr>
<tr>
<td>3</td>
<td>GUIDE CCW</td>
<td>INHIBIT</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### E.27: Dome inputs

<table>
<thead>
<tr>
<th>Bit</th>
<th>FFF0h [19302]</th>
<th>FFF1h [19303]</th>
<th>FFF2h [19304]</th>
<th>FFF3h [19305]</th>
<th>FFF4h [19306]</th>
<th>FFF5h [19307]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>phase 0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>phase 90</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Sync</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

---

*A distributed control system for the St Andrews twin photometric telescope: Page 287 of 335*
E.28: Encoders: Reference R.A. / DEC. coarse / fine

Ra reference encoders: Coarse/fine encoders linked via 25way D plug. Wire colors are constant.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Colour</th>
<th>Coarse Encoders</th>
<th>Fine Encoders</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BLUE/BLACK</td>
<td>D0</td>
<td>D10/MSB</td>
</tr>
<tr>
<td>2</td>
<td>WHITE/BLUE</td>
<td>D1</td>
<td>D9</td>
</tr>
<tr>
<td>3</td>
<td>YELLOW/GREEN</td>
<td>D2</td>
<td>D8</td>
</tr>
<tr>
<td>4</td>
<td>PURPLE</td>
<td>D3</td>
<td>D7</td>
</tr>
<tr>
<td>5</td>
<td>RED/BROWN</td>
<td>D4</td>
<td>D6</td>
</tr>
<tr>
<td>6</td>
<td>RED/BLACK</td>
<td>D5</td>
<td>D5</td>
</tr>
<tr>
<td>7</td>
<td>WHITE/GREEN</td>
<td>nc</td>
<td>D4</td>
</tr>
<tr>
<td>8</td>
<td>WHITE/RED</td>
<td>nc</td>
<td>D3</td>
</tr>
<tr>
<td>9</td>
<td>YELLOW</td>
<td>nc</td>
<td>D2</td>
</tr>
<tr>
<td>10</td>
<td>BLACK</td>
<td>nc</td>
<td>D1</td>
</tr>
<tr>
<td>11</td>
<td>WHITE</td>
<td>nc</td>
<td>D0/LSB</td>
</tr>
<tr>
<td>12</td>
<td>BROWN</td>
<td>nc</td>
<td>spare</td>
</tr>
<tr>
<td>13</td>
<td>ORANGE/D.GREEN</td>
<td>nc</td>
<td>spare</td>
</tr>
<tr>
<td>14</td>
<td>PINK</td>
<td>nc</td>
<td>spare</td>
</tr>
<tr>
<td>15</td>
<td>RED</td>
<td>nc</td>
<td>output ground</td>
</tr>
<tr>
<td>16</td>
<td>L.GREEN</td>
<td>nc</td>
<td>circuit ground</td>
</tr>
<tr>
<td>17</td>
<td>ORANGE</td>
<td>nc</td>
<td>spare</td>
</tr>
<tr>
<td>18</td>
<td>D.BLUE</td>
<td>nc</td>
<td>+5VDC</td>
</tr>
<tr>
<td>19</td>
<td>D.GREEN</td>
<td>nc</td>
<td>spare</td>
</tr>
<tr>
<td>20</td>
<td>YELLOW/RED</td>
<td>nc</td>
<td>spare</td>
</tr>
<tr>
<td>21</td>
<td>YELLOW/L.GREEN</td>
<td>nc</td>
<td>spare</td>
</tr>
<tr>
<td>22</td>
<td>ORANGE/L.GREEN</td>
<td>+5VDC</td>
<td>Case ground</td>
</tr>
<tr>
<td>23</td>
<td>GREY</td>
<td>+5VDC</td>
<td>Case input</td>
</tr>
<tr>
<td>24</td>
<td>RED/BLUE</td>
<td>0VDC</td>
<td>Reverse count</td>
</tr>
<tr>
<td>25</td>
<td>RED/L.GREEN</td>
<td>0VDC</td>
<td></td>
</tr>
</tbody>
</table>

E.29: Dome hydraulic pump

Hydraulic Pump:[53061]

| Manufacturer: | Vickers, England |
| Model:        | V 210-5-1A-12   |
| ref:          | H4 2396/1/2     |

Pump motor [52061].

| 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                   |

A distributed control system for the St Andrews twin photometric telescope: Page 288 of 335
## E.30: Dome valves

### Dump valve [52062]

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Flui - trol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating voltage</td>
<td>24VDC</td>
</tr>
<tr>
<td>Model</td>
<td>A37A</td>
</tr>
</tbody>
</table>

### Direction valves [52063] [52064]

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Sperry gyroscope co. LTD, Brentford, Essex.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating voltage</td>
<td>24VDC</td>
</tr>
<tr>
<td>Coil resistance</td>
<td>40ohms</td>
</tr>
<tr>
<td>Max current</td>
<td>600mA</td>
</tr>
<tr>
<td>Max flow rate</td>
<td>2.5 G.P.M.</td>
</tr>
<tr>
<td>Max pressure</td>
<td>3000 P.S.I.</td>
</tr>
<tr>
<td>Part number</td>
<td>21172-0</td>
</tr>
<tr>
<td>Model</td>
<td>3025-L</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Color Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RED + WHITE</td>
<td>Dump valve</td>
</tr>
<tr>
<td>BACK + WHITE</td>
<td>Dump valve</td>
</tr>
<tr>
<td>GREEN + WHITE</td>
<td>Direction valves return</td>
</tr>
<tr>
<td>YELLOW + WHITE</td>
<td>CW direction valve</td>
</tr>
<tr>
<td>BROWN + WHITE</td>
<td>nc</td>
</tr>
<tr>
<td>BLUE + WHITE</td>
<td>CCW direction valve</td>
</tr>
</tbody>
</table>
### E.31 : Interface units (general) : Mutibloc connectors

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0V</td>
<td>0V</td>
<td>0V</td>
<td>0V</td>
<td>15V CPU power input</td>
<td>0V</td>
<td>Motor 1 : 230VAC 3 phase in BLUE</td>
<td>0V</td>
</tr>
<tr>
<td>2</td>
<td>bit 1</td>
<td>bit 11</td>
<td>bit 1</td>
<td>bit 11</td>
<td>rx /tx input</td>
<td>output 1</td>
<td>Motor 1 : 230VAC 3 phase in BLUE</td>
<td>Motor 1 CCW limit</td>
</tr>
<tr>
<td>3</td>
<td>bit 2</td>
<td>bit 12</td>
<td>bit 2</td>
<td>bit 12</td>
<td>txrequest</td>
<td>output 2</td>
<td>Motor 1 CCW limit</td>
<td>Motor 1 CCW limit</td>
</tr>
<tr>
<td>4</td>
<td>bit 3</td>
<td>bit 13</td>
<td>bit 3</td>
<td>bit 13</td>
<td>0V</td>
<td>output 3</td>
<td>Motor 2 CW limit</td>
<td>Motor 2 CW limit</td>
</tr>
<tr>
<td>5</td>
<td>bit 4</td>
<td>bit 14</td>
<td>bit 4</td>
<td>bit 14</td>
<td>nc</td>
<td>output 4</td>
<td>Motor 2 CCW limit</td>
<td>Motor 2 CCW limit</td>
</tr>
<tr>
<td>6</td>
<td>bit 5</td>
<td>bit 15</td>
<td>bit 5</td>
<td>bit 15</td>
<td>nc</td>
<td>output 5</td>
<td>Motor 3 CW limit</td>
<td>Motor 3 CW limit</td>
</tr>
<tr>
<td>7</td>
<td>bit 6</td>
<td>bit 16</td>
<td>bit 6</td>
<td>bit 16</td>
<td>nc</td>
<td>output 6</td>
<td>Motor 3 CCW limit</td>
<td>Motor 3 CCW limit</td>
</tr>
<tr>
<td>8</td>
<td>bit 7</td>
<td>Buffer Test ** (Connect to 0V)</td>
<td>bit 7</td>
<td>Buffer Test ** (Connect to 0V)</td>
<td>output 7</td>
<td>Motor 4 CW limit</td>
<td>Motor 4 CW limit</td>
<td>Motor 4 CCW limit</td>
</tr>
<tr>
<td>9</td>
<td>bit 8</td>
<td>Buffer Test ** (Connect to 0V)</td>
<td>bit 8</td>
<td>Buffer Test ** (Connect to 0V)</td>
<td>nc</td>
<td>output 8</td>
<td>Motor 4 CCW limit</td>
<td>Motor 4 CCW limit</td>
</tr>
<tr>
<td>10</td>
<td>bit 9</td>
<td>nc</td>
<td>bit 9</td>
<td>nc</td>
<td>Sidereal Clk</td>
<td>nc</td>
<td>nc</td>
<td>nc</td>
</tr>
<tr>
<td>11</td>
<td>bit 10</td>
<td>nc</td>
<td>bit 10</td>
<td>nc</td>
<td>50Hz Sync</td>
<td>nc</td>
<td>nc</td>
<td>nc</td>
</tr>
<tr>
<td>12</td>
<td>5V</td>
<td>5V</td>
<td>5V</td>
<td>5V</td>
<td>nc</td>
<td>5V</td>
<td>5V</td>
<td>5V</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Pin</th>
<th>[1x308]</th>
<th>Colour</th>
<th>[10030]</th>
<th>[1x313]</th>
<th>Colour</th>
<th>[10030]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Manual select</td>
<td>Black</td>
<td>10</td>
<td>5V Digital</td>
<td>nc</td>
<td>nc</td>
</tr>
<tr>
<td>2</td>
<td>Fine enable</td>
<td>Brown</td>
<td>9</td>
<td>led anode</td>
<td>Green</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>Clamp enable</td>
<td>Red</td>
<td>8</td>
<td>led anode</td>
<td>Yellow</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>Clutch enable</td>
<td>Orange</td>
<td>7</td>
<td>Sidereal Clock</td>
<td>Orange</td>
<td>nc</td>
</tr>
<tr>
<td>5</td>
<td>Slew enable</td>
<td>Yellow</td>
<td>6</td>
<td>50Hz Sync In</td>
<td>Red</td>
<td>nc</td>
</tr>
<tr>
<td>6</td>
<td>Fine direction</td>
<td>Green</td>
<td>5</td>
<td>Fine Guide Ramp</td>
<td>Brown</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>Clamp direction</td>
<td>Blue</td>
<td>4</td>
<td>nc</td>
<td>nc</td>
<td>nc</td>
</tr>
<tr>
<td>8</td>
<td>Clutch direction</td>
<td>Purple</td>
<td>3</td>
<td>nc</td>
<td>nc</td>
<td>nc</td>
</tr>
<tr>
<td>9</td>
<td>Slew direction</td>
<td>Grey</td>
<td>2</td>
<td>nc</td>
<td>nc</td>
<td>nc</td>
</tr>
<tr>
<td>10</td>
<td>0V</td>
<td>White</td>
<td>1</td>
<td>0V</td>
<td>Black</td>
<td>15</td>
</tr>
</tbody>
</table>

### E.33 : Board connectors

<table>
<thead>
<tr>
<th>Pin</th>
<th>[1x101]</th>
<th>[1x313]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Power in</td>
<td>5V Digital</td>
</tr>
<tr>
<td>2</td>
<td>Power in</td>
<td>Manual override LED 'computer' anode</td>
</tr>
<tr>
<td>3</td>
<td>nc</td>
<td>Manual override LED 'Manual' anode</td>
</tr>
<tr>
<td>4</td>
<td>nc</td>
<td>Sidereal Clock out</td>
</tr>
<tr>
<td>5</td>
<td>0v</td>
<td>50Hz sync in</td>
</tr>
<tr>
<td>6</td>
<td>TXRX</td>
<td>Fine guide ramp inhibit</td>
</tr>
<tr>
<td>7</td>
<td>0v</td>
<td>nc</td>
</tr>
<tr>
<td>8</td>
<td>TX REQ</td>
<td>nc</td>
</tr>
<tr>
<td>9</td>
<td>0v</td>
<td>nc</td>
</tr>
<tr>
<td>10</td>
<td>0v</td>
<td>0V</td>
</tr>
</tbody>
</table>

---

A distributed control system for the St Andrews twin photometric telescope : Page 291 of 335
<table>
<thead>
<tr>
<th>Pin</th>
<th>[1x102], [1x103], [1x104], [1x201], [1x301], [1x401]</th>
<th>Pin</th>
<th>[1x102], [1x103], [1x104], [1x201], [1x301], [1x401]</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>5V</td>
<td>31</td>
<td>0V</td>
</tr>
<tr>
<td>30</td>
<td>12V</td>
<td>29</td>
<td>0V</td>
</tr>
<tr>
<td>28</td>
<td>/NMI</td>
<td>27</td>
<td>/RST</td>
</tr>
<tr>
<td>26</td>
<td>/IO READ</td>
<td>25</td>
<td>/IO WRITE</td>
</tr>
<tr>
<td>24</td>
<td>/INT</td>
<td>23</td>
<td>/IO BOARD</td>
</tr>
<tr>
<td>22</td>
<td>/CLOCK BOARD</td>
<td>21</td>
<td>/POWER BOARD</td>
</tr>
<tr>
<td>20</td>
<td>nc</td>
<td>19</td>
<td>Data 7</td>
</tr>
<tr>
<td>18</td>
<td>Data 6</td>
<td>17</td>
<td>Data 5</td>
</tr>
<tr>
<td>16</td>
<td>Data 4</td>
<td>15</td>
<td>Data 3</td>
</tr>
<tr>
<td>14</td>
<td>Data 2</td>
<td>13</td>
<td>Data 1</td>
</tr>
<tr>
<td>12</td>
<td>Data 0</td>
<td>11</td>
<td>I/O Address 0</td>
</tr>
<tr>
<td>10</td>
<td>I/O Address 1</td>
<td>9</td>
<td>I/O Address 2</td>
</tr>
<tr>
<td>8</td>
<td>I/O Address 3</td>
<td>7</td>
<td>nc</td>
</tr>
<tr>
<td>6</td>
<td>/WATCH DOG OUT</td>
<td>5</td>
<td>Fine guide out</td>
</tr>
<tr>
<td>4</td>
<td>50Hz Sync in</td>
<td>3</td>
<td>Sidereal Drive out</td>
</tr>
<tr>
<td>2</td>
<td>/RXTX</td>
<td>1</td>
<td>/TX REQUEST</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pin</th>
<th>[1x302], [1x303], [1x304], [1x305], [1x502], [1x503]</th>
<th>[1x310], [1x601], [1x602]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5V Digital</td>
<td>5V Digital</td>
</tr>
<tr>
<td>2</td>
<td>Data bit 7</td>
<td>FFF0h bit 4</td>
</tr>
<tr>
<td>3</td>
<td>Data bit 6</td>
<td>FFF0h bit 5</td>
</tr>
<tr>
<td>4</td>
<td>Data bit 5</td>
<td>FFF0h bit 6</td>
</tr>
<tr>
<td>5</td>
<td>Data bit 4</td>
<td>FFF0h bit 7</td>
</tr>
<tr>
<td>6</td>
<td>Data bit 3</td>
<td>FFF1h bit 4</td>
</tr>
<tr>
<td>7</td>
<td>Data bit 2</td>
<td>FFF1h bit 5</td>
</tr>
<tr>
<td>8</td>
<td>Data bit 1</td>
<td>FFF1h bit 6</td>
</tr>
<tr>
<td>9</td>
<td>Data bit 0</td>
<td>FFF1h bit 7</td>
</tr>
<tr>
<td>10</td>
<td>0V</td>
<td>0V</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pin</th>
<th>[1x311]</th>
<th>[1x312]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12V relay out</td>
<td>12V relay out</td>
</tr>
<tr>
<td>2</td>
<td>Motor enable A out</td>
<td>Motor enable C out</td>
</tr>
<tr>
<td>3</td>
<td>Clamp A+B enable in</td>
<td>Clamp C+D enable in</td>
</tr>
<tr>
<td>4</td>
<td>Motor enable B out</td>
<td>Motor enable D out</td>
</tr>
<tr>
<td>5</td>
<td>Buffer enable in (See right)</td>
<td>Buffer enable in (Connect to pin 6)</td>
</tr>
<tr>
<td>6</td>
<td>5V digital</td>
<td>5V digital</td>
</tr>
<tr>
<td>7</td>
<td>Motor direction A</td>
<td>Motor direction C</td>
</tr>
<tr>
<td>8</td>
<td>Clamp A+B direction in</td>
<td>Clamp C+D direction in</td>
</tr>
<tr>
<td>9</td>
<td>Motor direction B</td>
<td>Motor direction D</td>
</tr>
<tr>
<td>10</td>
<td>0V</td>
<td>0V</td>
</tr>
</tbody>
</table>

A distributed control system for the St Andrews twin photometric telescope: Page 292 of 335
Pin 5 was brought out to enable hot limit / safety cutouts to be integrated into the board. Connect to Pin 6 (5V digital).

<table>
<thead>
<tr>
<th>Pin</th>
<th>[1x701]</th>
<th>[1x704]</th>
<th>[1x705]</th>
<th>Int. color</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5V</td>
<td>5V</td>
<td>0V</td>
<td>yellow/purple</td>
</tr>
<tr>
<td>2</td>
<td>/Sensor OK</td>
<td>CPU disable RLY A</td>
<td>nc</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>CPU power</td>
<td>Clamp AB</td>
<td>Power in</td>
<td>grey</td>
</tr>
<tr>
<td>4</td>
<td>Standby power</td>
<td>415V disable RLY B</td>
<td>TX/RX</td>
<td>green</td>
</tr>
<tr>
<td>5</td>
<td>415V on</td>
<td>0V</td>
<td>TX request</td>
<td>blue</td>
</tr>
<tr>
<td>6</td>
<td>Reset</td>
<td>0V</td>
<td>CTS in</td>
<td>orange</td>
</tr>
<tr>
<td>7</td>
<td>Stop/Resume</td>
<td>CPU disable RLY C</td>
<td>RTS out</td>
<td>red</td>
</tr>
<tr>
<td>8</td>
<td>Line return</td>
<td>Clamp CD</td>
<td>RX in</td>
<td>brown</td>
</tr>
<tr>
<td>9</td>
<td>Relay disable</td>
<td>415V disable RLY D</td>
<td>TX out</td>
<td>black</td>
</tr>
<tr>
<td>10</td>
<td>0V</td>
<td>0V</td>
<td>nc</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pin</th>
<th>[1x702]</th>
<th>[1x703]</th>
<th>[11045]</th>
<th>Ribbon</th>
<th>[1x702]</th>
<th>[1x703]</th>
<th>[11045]</th>
<th>Ribbon</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5V</td>
<td>5V</td>
<td>5V</td>
<td>Black</td>
<td>Black</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CPU power relay</td>
<td>CPU error</td>
<td>Brown</td>
<td>Brown</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>System OK</td>
<td>nc</td>
<td>Red</td>
<td>Red</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Standby power</td>
<td>System error</td>
<td>Orange</td>
<td>Orange</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>415V motor relay</td>
<td>Stop/Resume</td>
<td>Yellow</td>
<td>Yellow</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>415V OK</td>
<td>System reset</td>
<td>Green</td>
<td>Green</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>CPU OK</td>
<td>Bus error</td>
<td>Blue</td>
<td>Blue</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>TX</td>
<td>Line error</td>
<td>Purple</td>
<td>Purple</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>TX request</td>
<td>System reset</td>
<td>Grey</td>
<td>Grey</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0V</td>
<td>0V</td>
<td>White</td>
<td>White</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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
F.2: Oil pump and dome shutter power supply
F.4 : R.A. slew and R.A. clamp switchgear
F.5 : DEC. slew and DEC. clamp switchgear
F.7: Low voltage transformers and distribution

A distributed control system for the St Andrews twin photometric telescope: Page 301 of 335
F.8: Low voltage spine distribution

A distributed control system for the St Andrews twin photometric telescope: Page 302 of 335
A distributed control system for the St Andrews twin photometric telescope: Page 303 of 335
F.10: Offset R.A. fine motor
F.11 : Offset DEC. fine motor
F.12: Reference R.A. limits
F.13: Reference DEC. limits
F.14: Offset R.A. limits and encoder
F.15: Offset DEC. limits and encoder
F.16: Dome rotation opto isolator and controller
F.17: Sidereal motor driver, R.A. sidereal opto isolator, driver and R.A. clutch

A distributed control system for the St Andrews twin photometric telescope: Page 311 of 335
F.18: Hydraulic pump 3 phase supply and switchgear
F.19: Reference R.A. coarse and fine encoders

A distributed control system for the St Andrews twin photometric telescope: Page 313 of 335
F.20: Reference DEC, coarse and fine encoders
F.21 : Reference and offset focus motors and switchgear
F.22: Microcontroller power and communications
A distributed control system for the St Andrews twin photometric telescope: Page 317 of 335
F.24 : 3 Phase solid state relay unit

A distributed control system for the St Andrews twin photometric telescope : Page 318 of 335
F.25: Pulse width modulated motor driver

A distributed control system for the St Andrews twin photometric telescope: Page 319 of 335
E.26: Digital and Opto Isolator
F.27: Dome encoder isolator
F.28 : CPU power board

A distributed control system for the St Andrews twin photometric telescope: Page 322 of 335
F.29 : 3 Phase clamp overload sensor

A distributed control system for the St Andrews twin photometric telescope : Page 323 of 335
F.30 : 3 Phase mechanical pump contactor

A distributed control system for the St Andrews twin photometric telescope : Page 324 of 335
F.31 : 3 Phase pump solid state relay contactor

A distributed control system for the St Andrews twin photometric telescope : Page 325 of 335
F.32 : Manual switch unit

A distributed control system for the St Andrews twin photometric telescope : Page 326 of 335
F.34: I/O board (input buffer section)
F.36: CPU board (memory section)

A distributed control system for the St Andrews twin photometric telescope: Page 330 of 335
F.37: Proposed fibre optic transceiver unit

A distributed control system for the St. Andrews twin photometric telescope: Page 331 of 335
F.38: Transceiver used

A distributed control system for the St Andrews twin photometric telescope: Page 332 of 335