MRO Delay Line

Production Metrology Software Functional Description

INT-406-VEN-1004

Bodie Seneta
bodie@mrao.cam.ac.uk

rev 1.0
22 January 2010

Cavendish Laboratory
Madingley Road
Cambridge CB3 0HE
UK

Change Record

Revision	Date	Authors	Changes
0.1	2010-01-15	EBS	First draft.
0.2	2010-01-18	EBS	Gave RTnet discussion its own subsection. Added RTnet references and bibliography. Corrections to control loop timing delays. Many minor diagram and wording changes.
0.3	2010-01-18	EBS	Added Reference and Applicable Documents sections, merged with bibliography. Added datum and slew algorithm discussion. Minor diagram and wording changes.
1.0	2010-01-22	EBS	Added information about jitter timing requirements. Edited slew servo for consistency with INT-406-CON-0101. Minor wording changes.

Objective

To describe the design of the metrology production software and present the rationale upon which that design is based.

Reference Documents

RD1	MRO Delay Line Derived Requirements	INT-406-VEN-0107
RD2	Metrology System & VME Hardware Design Description	INT-406-VEN-0113
RD3	MRO Delay Line Timing Requirements for Control Loops	INT-406-VEN-XXXX
RD4	Kiszka, J. WAgner, B. Zhang, Y. Broenink, J., *RTnet—A Flexible Hard Real-Time Networking Framework*, 10th IEEE Conference on Emerging Technologies and Factory Automation, 2005.	
RD5	Barbalace, A. Luchetta, A. Manduchi, G. Moro, M. Soppelsa, A. Taliercio, C., *Performance Comparison of VxWorks, Linux, RTAI and Xenomai in a Hard Real-time Application*, IEEE Transactions on Nuclear Science, **55** pp.435–439	
RD6	Luchetta, A. Barbalace, A. Manduchi, G. Soppelsa, A. Taliercio, C., *Real-time communication for distributed plasma control systems*, Proceedings of the 6th IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research, 2008. pp. 520–524.	
RD7	Tian, Y. Ren, G. Wu, Q., *Implementation of Real-time Network Extension on Embedded Linux*, International Conference on Communication Software and Networks, 2009. pp.163–167	

Applicable Documents

AD1	Requirement specifications for the MROI "production" delay line software	INT-406-CON-0101

Scope

This document forms part of the documentation for the preliminary production software design review. It describes the design and implementation of the metrology software and the rationale behind that design, as influenced by the performance requirements and the hardware constraints.

It is assumed that the reader is familiar with the construction and principle of operation of the Magdalena Ridge Observatory Interferometer (MROI) and in particular with the design of the MROI delay lines. Also assumed is some background in computer programming (especially under linux), electronics, and computer networks.

Contents

1 Introduction **4**

2 Metrology System Hardware Context **4**

3 Performance Requirements **5**

4 Hardware and Timing Constraints **6**

4.1 Hardware 6

4.2 Timing Constraints 7

5 Software Development and Execution Environment **8**

6 Software Architecture **9**

6.1 Kernel Modules 9

6.2 The Control Loop 10

6.3 Fringe Tracker 12

6.4 User-Space Program 13

6.4.1 Workstation connect timer 13

6.4.2 Workstation connection failure 14

6.4.3 Trolley connect timer 14

6.4.4 Trolley connection failure 14

6.4.5 Command arrives from workstation 14

6.4.6 Trajectory information arrives from workstation 15

6.4.7 Telemetry data arrives from metrocontrol 15

7 Testing **16**

8 Conclusion **18**

1 Introduction

A functional metrology system is central to the performance of any modern interferometer delay line. It provides vital real-time feedback as to how much optical delay is being inserted into each science beam and, as part of a control loop, enables delays to be so well controlled that science beam optical coherence and the ability to collect science data are achieved.

The metrology system at the MROI will be equally important to that instrument's functionality, and it must be designed with care. An important aspect of that design is the software, which must calculate feedback for up to ten delay lines within strict deadlines while also working within the constraints imposed by the available hardware.

This document firstly provides some context: an overview of the metrology system, its peformance requirements, a description of the metrology hardware and the constraints that hardware imposes on the software design.

The software itself is then described, beginning with the overall software architecture and moving on to a detailed discussion of the control loop that is central to the metrology system operation.

Finally, some results are presented as indicators of the ability of the software to meet the performance requirements.

2 Metrology System Hardware Context

The purpose of the metrology system is to keep each delay line trolley on the trajectory needed to collect science data. It does this by measuring the current trolley position and providing a negative feedback error signal to the trolley. Figure 1 shows how the metrology system fits into a delay line as a whole.

The system is based upon a commercial Agilent metrology laser, whose light is split ten ways to service up to ten delay lines. The light is sent via a beam expander to each trolley, where it traverses the catseye mirror pair within and returns to the metrology system via a beam compressor. There it mixes with the outgoing beam to produce optical interference that can be monitored with a detector. Movement of the trolley towards or away from the metrology system will cause the detector to generate beats which can be counted in an Agilent computer card to determine how far the trolley has moved, accurate to within fractions of the wavelength of light. Because the science beam traverses a geometrically identical path, the additional displacement in that path is thus also known.

Such a system can only measure relative displacement, so some other means of providing an absolute measurement (a "datum") is required. This is provided by an optical datum switch that is triggered by a knife edge on the trolley.

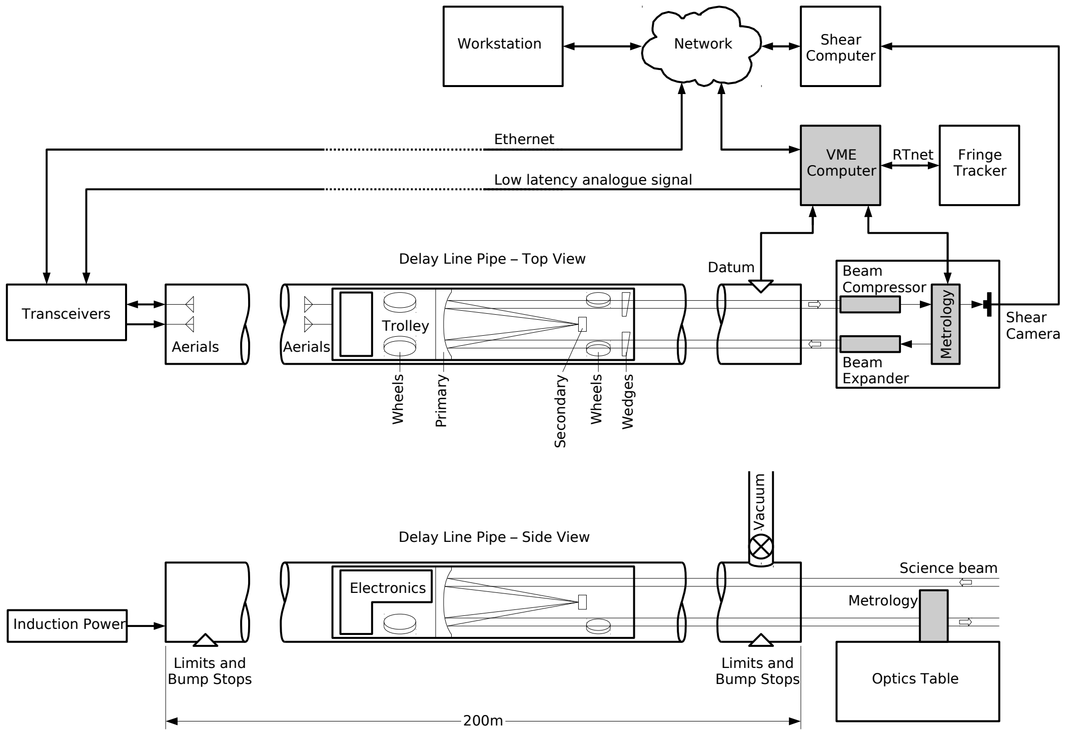

Figure 1: Overview of the hardware for a single delay line. The metrology system components are shown in grey and the metrology software under review executes on the VME computer.

The metrology system measurement count and the datum switch state are both inputs to the VME computer. Other inputs are the current time from an on-board GPS clock, commands and trajectory data that arrive via the network, and a dedicated ethernet input from the fringe tracker. The outputs are a low-latency analogue signal to the trolley catseye to compensate for small but rapidly varying trajectory errors, fringe tracker feedback, commands to the trolley carriage via ethernet for coarser corrections, and telemetry for the workstation.

3 Performance Requirements

The metrology software requirements are driven by the optical path delay control loop requirements (RD1 and RD3)[1]. The control loop attempts to minimise tracking

[1]RD3 is now out of date. However, the requirements on jitter in that document are still valid.

errors for each trolley by sending a feedback error signal to each trolley based on the difference between the trolley metrology-measured position and the demand position.

The requirements are:

- Control loop bandwidth: 3db frequency greater than 100Hz.
- Control loop latency less than 40μs.
- Maximum RMS waypoint-time to actualisation-time jitter:
  - For 10ms interval: 0.59μs.
  - For 35ms interval: 1.6μs.
  - For 50ms interval: 2.2μs.

The first requirement, plus a safety margin, translates into a control loop sample period of 200μs. The second requirement states that each trolley must have its error feedback applied less than 40μs (including transmission latency) after its position is measured by the metrology hardware. The last three requirements impose limits on how much the difference can vary between the time that a position is calculated *for* and the time at which it is *applied*.

Note that it is not necessary to include the transmission latency when considering the requirements, because that is negligible.

Additionally there are performance requirements concerning the fringe tracker (AD1):

- Fringe tracker offset delivery rate: 1Hz–1kHz.
- Fringe tracker offset delivery latency less than 200μs.
- Fringe tracker feedback latency less than 500μs.

4 Hardware and Timing Constraints

4.1 Hardware

The software executes on a single board computer, connected to peripheral devices via a VME bus within a fan-cooled VME crate. Specifically, the VME crate components are as follows:

- Computer: Concurrent Technologies VP 325 022-23U. An Intel x86-style computer with a Pentium-M 1.6GHz processor and a Tundra Universe II bridge. The function of the bridge is to make the VME bus appear as a native PCI bus peripheral to the computer.

- Storage: Concurrent Technologies DS MSS 002. Contains a hard drive and printer output for the computer.
- Timer: Symmetricom TTM635VME. This board keeps time using an input GPS signal and generates a periodic interrupt for use in processing metrology data.
- Digital Input/Output: Acromag IP470A. Interfaces to datum switches via a custom interface board.
- Analogue output: Acromag IP220. Interfaces with low latency links to trolleys via a custom interface board.
- VME bus carrier: Tews TechnologiesTVME200. Carrier board for both of the Acromag boards.
- Interface board: A custom interface board with front panel sockets for connection to the datum switches and low latency links.
- Metrology board: Agilent 10898A. Interfaces with the metrology laser and delay line metrology optics. One 10898A has two channels and each can measure the position of one delay line trolley. One 10898A board is currently used, but there could be up to five present, depending on how many delay lines are commissioned.

There are additionally several peripherals external to the VME crate:

- Metrology laser: Light from the metrology laser is split ten ways (one measurement beam per delay line). A reference beam is mixed with each of the ten measurement beams, and the resulting signals are fed to the Agilent 10898A board via optical fibres in order to determine displacement of the associated trolleys. Tests have shown that there is ample metrology signal per channel even though the original laser beam is split ten ways.
- Datum switches: Contrinex LTS-1180L-103-516. These optical switches trigger when a trolley goes past them, and are interfaced to the VME crate via the Acromag IP470A.
- Low latency links: A low latency link transmits an analogue voltage from the VME crate to a given delay line trolley via a balanced wire and FM radio link. The Acromag IP220 drives one low latency link per trolley.

4.2 Timing Constraints

The main software timing constraint is that reading from the VME data bus is slow. Using Xenomai real-time linux (Section 5) and an oscilloscope monitoring the VME bus, the following delays have been found:

- Interrupt routine entry latency: 5μs.
- VME bus data read: 1μs.
- VME bus data write: 0.3μs.
- Calculation time for one delay line: 0.5μs.

The slow read time appears to be an aspect of the PCI-VME bridge, and has also been observed when using QNX instead of Xenomai.

These results can be used to rule out some software architectures. For example, triggering the processing of each delay line off its own interrupt would not work well for ten delay lines because there would be 50μs of overhead per cycle spent on interrupt entry. Similar delays would be incurred in reading the time from the VME timer board prior to each trolley calculation.

It seems much better to have a single interrupt routine in which the time is read only once and the deterministic nature of Xenomai is exploited to calculate the time of each delay line update by dead reckoning. In this scenario it would be quite feasible to perform the necessary I/O and calculations for one trolley within 10μs such that an interrupt routine for all ten would take 100μs. This would leave 100μs out of each 200μs period free for the computer to perform housekeeping tasks.

This is the architecture that has been adopted and more detail can be found in Subsection 6.2

5 Software Development and Execution Environment

Development and execution of metrology system code occur on the same computer, the one described in Subsection 4.1 above. The operating system undergoing tests is Linux (the Debian "testing" distribution) using a 2.6.29.4 kernel and patched with Xenomai 2.4.8-2 to run as a hard real-time system. The Xenomai *RTDM* application programming interface is used. The compiler is *gcc*. This arrangement was chosen for its open-source nature, hard real-time performance (including the use of floating point arithmetic) and the flexibility to run real-time and non-real-time code in the same program.

The real-time software components are contained in linux kernel-space modules ("drivers") that use Xenomai libraries and the linux *kbuild* system. They are written in C, the de-facto standard for linux kernel code. The non-real-time components mainly occur in linux user-space like any other linux program and are event-driven through use of the *Glib* library framework. They are also written in C, but in an object oriented fashion to enable straightforward porting to C++ should that be preferred.

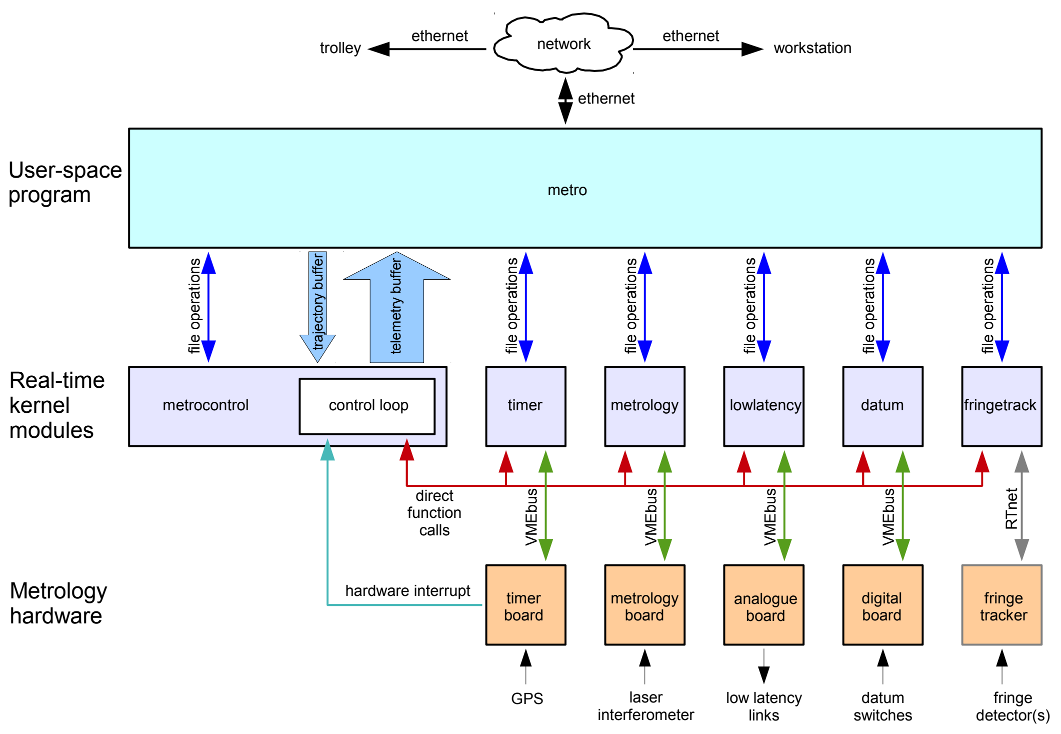

Figure 2: Overview of the metrology program architecture and its external connections.

6 Software Architecture

The metrology code consists of a set of real-time kernel modules which interface with the hardware and process data in real time, and a user-space program which processes non-real-time tasks. An overview of the metrology program architecture is illustrated in Figure 2.

6.1 Kernel Modules

There are six modules. Four of these, called "timer"," metrology", "lowlatency" and "datum", communicate with hardware in the VME crate via the VME bus, while "fringetrack" communicates with the remote fringe tracker via RTnet, a real-time low-latency dedicated ethernet link (RD4). These modules do very little to process data going to or coming from their associated hardware, they simply facilitate the connection. The majority of the metrology processing occurs in the "metrocontrol" module.

This modularity has many advantages over a monolithic program. For example, hardware changes can be accommodated simply by modifying the associated module (rather than modifying the entire program). The overall program structure also becomes easier to understand and maintain.

The metrocontrol module is a container for the metrology control loop (Subsection 6.2), which runs whenever a periodic hardware interrupt pulse arrives from the timer board. The control loop interacts with the metrology hardware by directly calling appropriately exported functions from the other modules (this is very fast as it has no more overhead than a function call within metrocontrol itself). metrocontrol also maintains two buffers that are accessable from user space: one gets loaded with the telemetry data generated during control loop execution, the other reads trajectory demand information that arrives from user space. These two buffers effectively decouple the real-time and non-real-time operations of the metrology software.

In addition to the interfaces already discussed, every module also has a standard file stream interface to the user-space program, which allows initial configuration and some state changes to be controlled from user-space.

6.2 The Control Loop

The metrology software design depends on the structure of its innermost control loop, which in turn depends on the performance requirements (Section 3) and the hardware timing constraints (Subsection 4.2).

To guarantee that the control loop has sufficiently accurate timing, it is written as an interrupt-driven routine within the metrocontrol Xenomai real-time linux kernel module. The interrupt is triggered at 5kHz by a pulse on the VME bus SYSFAIL line, which is generated by the timer board and phase-locked to the timer clock's one-second tick.

The execution of the control loop is summarised in Figure 3.

Once within the interrupt routine, the code firstly latches and then reads the time from the timer board. This serves two purposes: firstly it provides an accurate timestamp for data acquired during the routine, and secondly it tells the routine what the interrupt latency was. All other times within the routine can be approximated by dead reckoning, since a Xenomai kernel-level interrupt routine is highly deterministic.

Next, the datum switch array is read. All the switches can be polled at once by a single VME bus read so it makes sense to do this before considering individual trolleys.

The current time is then calculated, based on the time read from the timer board and the time taken to read that time and the datum switches. A delay is added to pad the

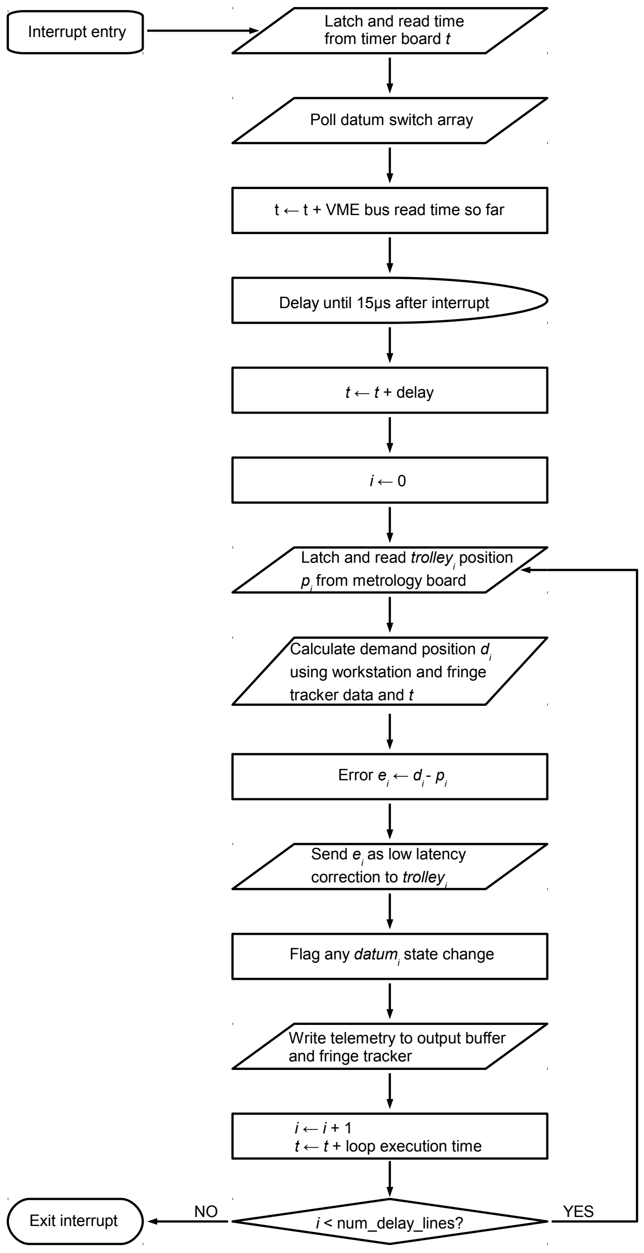
Interrupt entry
Latch and read time from timer board t
Poll datum switch array
t ← t + VME bus read time so far
Delay until 15µs after interrupt
$t \leftarrow t$ + delay
$i \leftarrow 0$
Latch and read $trolley_i$ position p_i from metrology board
Calculate demand position d_i using workstation and fringe tracker data and t
Error $e_i \leftarrow d_i - p_i$
Send e_i as low latency correction to $trolley_i$
Flag any $datum_i$ state change
Write telemetry to output buffer and fringe tracker
$i \leftarrow i + 1$
$t \leftarrow t$ + loop execution time
i < num_delay_lines?
YES
NO
Exit interrupt

Figure 3: Overview of inner control loop execution. Here num_delay_lines is the number of delay lines currently in use.

total elapsed time since the SYSFAIL interrupt to 15μs. This corrects for any interrupt entry latency jitter that may have occurred so that trolley processing always begins 15μs after the SYSFAIL line fired.

Each operational delay line is now considered in turn[2].

Firstly, the position of that trolley is latched and read from a metrology board. Then a demand position is calculated, interpolating from a set of trajectory waypoints sent by the workstation to the time at which the metrology position was latched. Optionally, an offset from the fringe tracker is then added. Finally, the demand and actual trolley positions are compared to determine an error signal. That signal is written to the analogue ouput register for the low latency link to that trolley. The gain is as high as the link will tolerate, to minimise noise encountered during transmission.

The time between latching the metrology and sending the low latency correction takes about 8μs, well within the 40μs tolerance specified above for the calculation. Some housekeeping then follows. Any datum state changes get flagged, data accumulated during this iteration is added to an output buffer for later processing, and that 8μs is added to the elapsed time to be used when considering the next trolley. Because the time is calculated using dead reckoning, any errors in it are cumulative and build up until the last trolley is considered. However, the tolerance on the total error is 40μs and errors on that scale would have easily been noticed in lab tests already performed.

The jitter requirements (Section 3) are met by exploiting the determinism of the routine. The only potential disturbances to the execution time are caused by PCI bus access variability, due to the state of the PCI bus clock when requests arrive or to bus traffic caused by semi-autonomous PCI devices. The former will cause a jitter of only $\pm$15ns, and the latter should be infrequent enough that the RMS jitter is still within specification. However, if this becomes an issue, it should be possible to explicitly lock out external PCI bus requests for the duration of the interrupt routine.

Finally, after all trolleys have been considered the interrupt exits and hands control back to the kernel.

6.3 Fringe Tracker

It is a requirement (Section 3) that the metrology system receive real-time updates from the fringe tracker, at rates of 1Hz–1kHz and latency less than 200μs. It is also a requirement that the metrology system sends feedback to the fringe tracker with a latency of less than 500μs. In order to meet these requirements, data is transmitted between the two systems using RTnet.

[2]The control loop knows which delay lines are in use from the application's configuration file, which in turn is generated by the Interferometer Supervisory System before the application is started.

No in-house tests of RTnet's performance have yet been done. However, other research groups have made such investigations. Barbalace et. al. (RD5) found RTnet had a latency of just over 100μs when transmitting packets of approximately 256 bytes between two VME systems. The same group (RD6) concluded that "the performance obtained by using open source code is suitable for sub-millisecond real-time communication in plasma control." Additionally, Tian et. al. (RD7) measured a round trip time of 183μs for 400 bytes of data. Given these results it seems unlikely that RTnet would be unable to satisfy the communications requirements.

As the data arrival rate from the fringe tracker is variable, the fringetrack module checks for incoming data at the end of each control loop run. If data has arrived, then it is written into a fringe tracking offset variable that can be read during future control loop iterations until it is updated once more. The metrology feeback, on the other hand, occurs at a fixed rate and is simply sent to the fringe tracker each time the control loop runs, or once every two cycles if RTnet cannot sustain a 5kHz packet rate.

6.4 User-Space Program

The user-space program, "metro", is relatively ordinary. It has the task of interfacing the control loop with the delay line workstation and trolleys. Code written for a metrology emulator by John Young will be extensively re-used, greatly accelerating development.

On startup, the program loads a local configuration file that sets various parameters. These include an identifier string, internet protocol addresses and port numbers for communication with the workstation and trolley. There is also a mapping between delay line number and metrology channel number, since this will vary as more delay lines are added to the MROI and delay lines may also occasionally be out of commission during repair or maintenance.

During initialisation, the program checks for the presence of all the modules and initialises them. Once initialisation is complete, the program enters the main event loop, which is managed by the *Glib* framework. It waits there for various events to happen. The possible events are described in more detail below.

6.4.1 Workstation connect timer

During initialisation, a timer is set to cause an event once every two seconds (this value can be changed in the configuration file). On this signal the program attempts to initiate a connection to the workstation via the network. If the connection is successful, the program is able to receive commands from the workstation and the timer is cancelled. If no connection is made, further connection attempts will be triggered by the timer until one is successful.

6.4.2 Workstation connection failure

If the connection to the workstation is broken, perhaps by a network or workstation problem, this event causes the timer described in Subsection 6.4.1 to be reinitialised, so that the metrology system will reconnect to the workstation automatically once the problem is fixed.

6.4.3 Trolley connect timer

The trolley connect timer behaves like the workstation connect timer, except that it attempts to connect to all the trolleys listed in the initialisation file, thereby allowing direct control of coarse carriage velocity from the metrology system.

6.4.4 Trolley connection failure

If the network connection with a trolley fails the timer in 6.4.3 gets reinitialised so that a connection can be reestablished once the failure is rectified.

6.4.5 Command arrives from workstation

When connected, the workstation can send commands to the metrology system to change its behaviour. These are checked for consistency upon arrival.

Some commands make the metrology system drive trolleys: making them idle, trying to make them track trajectories input from the workstation, or getting them to search for the datum switch. Other commands are concerned with fringe tracker input: turning it on or off and resetting the offset.

Of these commands, the datum-seeking command initiates the most complex sequence of events, because no assumptions can be made as to the trolley position when this command arrives. The algorithm is as follows:

- Send the trolley away from the metrology system at full speed for several seconds. This is to ensure that it is further away than the datum switch, and the delay line limit switches will cause it to safely stop if it is at the far end of the delay line.
- Stop the trolley.
- Slew the trolley towards the metrology system at full speed until the datum switch triggers. Note the metrology reading when this happens. At full speed the trolley is moving at 0.7m/s, so with metrology sampling at 5kHz this technique introduces no more than 140μm of error to the datum finding procedure.

- Stop the trolley.
- Slew the trolley away from the metrology system until it is 5mm closer to the metrology system than the datum is, using the previously noted metrology reading.
- Stop the trolley.
- Slew the trolley away from the metrology system at 1mm/s until the datum switch triggers. Zero the metrology counter when this happens. This reading only introduces a maximum of 200nm of error, so errors inherent in the datum switch measurement (a few microns) now dominate.
- Stop the trolley.

The slew algorithm aims to minimise the time for a trolley to go from a tracking trajectory (or a standing start) to a new tracking trajectory:

- Implement a maximum-acceleration, constant-deceleration servo if the displacement is large.
- If the trolley is within 10mm of the new tracking position, implement a damped first-order servo.
- Once within tracking range (about 2mm), tell the trolley catseye to adjust its position using low latency link feedback and tell the trolley carriage to travel at the required velocity.

6.4.6 Trajectory information arrives from workstation

A second's worth of trajectory demands arrives from the workstation at least one second in advance, to give metro time to deliver it to metrocontrol. This event causes metro to put the data in metrocontrol's trajectory buffer so that it is available to the control loop when it is needed.

6.4.7 Telemetry data arrives from metrocontrol

This signal occurs when metrocontrol's telemetry buffer has acumulated at least a tenth of a second's worth of data. When that happens, the oldest tenth of a second's worth is read out and packaged up into telemetry that is then sent to the workstation over the network. The most recent of this data is also used to change the state of metro (for example, when a datum switch has changed state) and to modify the velocity sent to the trolley over the network.

Once this is done, the program returns to the main event loop and waits for another event. This particular event occurs at 10Hz (with some jitter) and importantly is synchronised to the rate at which telemetry data is generated. That in turn is locked to the phase-locked 5kHz interrupt signal coming from the timer board.

7 Testing

The production metrology software is a complete rewrite of the prototype metrology software, in a different operating system, and it is prudent to perform timing tests to ensure that it will work to specification. Here, such a test is presented.

In this test, test software is run to simulate a control loop. It performs all the calculations and VME bus reads and writes that will be required by the final product. In this particular test, it handles two metrology channels.

The performance is monitored by a dual channel oscilloscope. Channel 1 monitors the SYSFAIL interrupt line on the VME bus and this also serves as the oscilloscope trigger. Channel 2 monitors the least significant bit on the VME data bus and gives a good indication of bus activity. 40ms exposure photographs of the oscilloscope screen are then taken, thereby integrating the display over 200 control loop cycles.

Figure 4 shows an entire 200μs control loop cycle, indicated by the SYSFAIL interrupt appearing at the start and end of the top trace. Some VME bus activity is visible in the lower trace. Importantly bus activity has finished about 30μs into the cycle, indicating that only a small portion of the entire cycle is spent within the interrupt routine. This is an ideal result, as it means there is plenty of time for the computer to execute other tasks before SYSFAIL triggers again.

Figure 5 shows the earliest part of the signal from Figure 4 in more detail. In the top trace, the SYSFAIL interrupt is shown to have a finite width, while the bottom trace shows two groups of bus activity. The first of these groups is associated with reading the time and datum hardware, which is only done once per cycle. The second group is actually two groups of nearly identical activity and indicates the metrology reads and low latency writes that occur once per metrology channel.

The event timing in the second trace can be used to make deductions about the control loop performance:

- Bus activity is first seen 5μs after the SYSFAIL line goes low. This shows that Xenomai interrupt entry latency is consistently about 5μs for the metrology computer.
- Successive reads occur about 1μs apart, indicating, as discussed above, that VME bus reads are the slowest parts of the control loop.

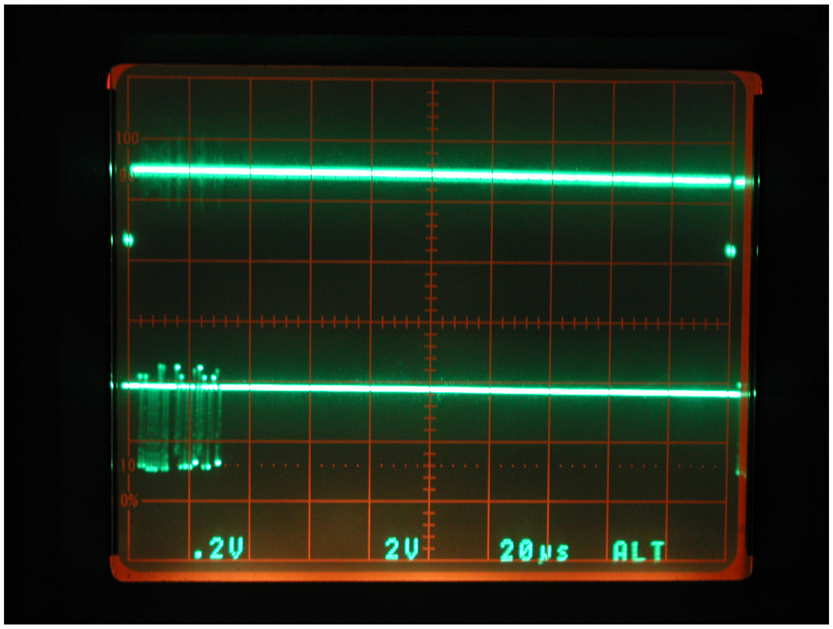

Figure 4: Oscilloscope screenshot of VME bus activity over an entire 200μs metrology sample cycle.

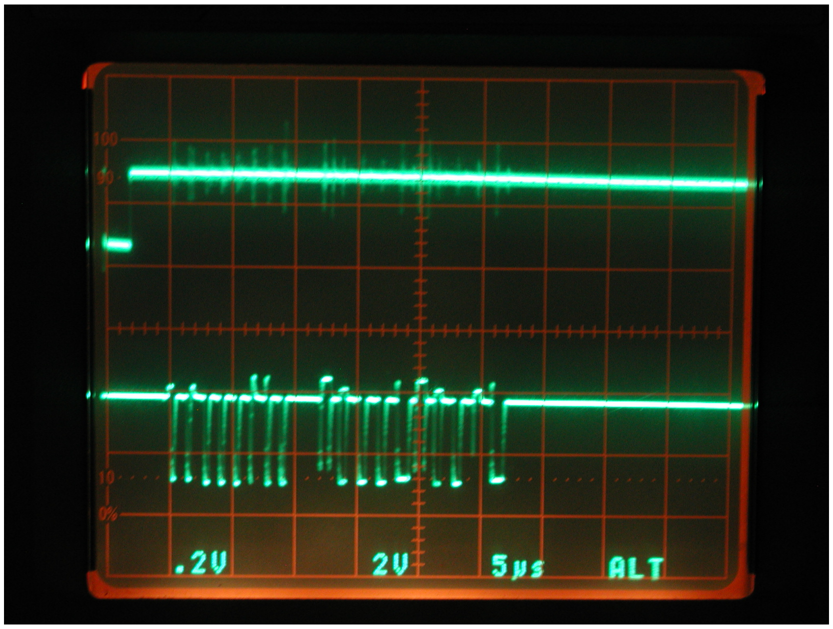

Figure 5: Oscilloscope screenshot of VME bus activity over the first 50μs after the SYSFAIL interrupt.

- Each of the two calculations for the two metrology channels takes about 8μs to execute. Hence ten channels would take about 80μs and the entire interrupt routine would exit 97μs after the SYSFAIL line went low. This still leaves 100μs per cycle, or half the available CPU time, for the computer to run the user-space program and do housekeeping.
- The trace is sharp even though it is an integrated view of 200 oscilloscope sweeps. This indicates that the interrupt jitter and any other causes of jitter within the interrupt routine are small.

8 Conclusion

In conclusion, it is clear that the hardware and software in the metrology system will be able to correctly manage the tracking of ten delay line trolleys with ease.