When you want it done right now: experience from programming hard real time systems in Xenomai for the Magdalena Ridge Observatory interferometer

Seneta, Eugene, Young, John, Buscher, David, Ligon, E. Robert, Etscorn, Dylan, et al.

Eugene B. Seneta, John S. Young, David F. Buscher, E. Robert Ligon, Dylan Etscorn, Allen Farris, "When you want it done right now: experience from programming hard real time systems in Xenomai for the Magdalena Ridge Observatory interferometer," Proc. SPIE 11452, Software and Cyberinfrastructure for Astronomy VI, 114521G (13 December 2020); doi: 10.1117/12.2562191

SPIE.

Event: SPIE Astronomical Telescopes + Instrumentation, 2020, Online Only

When You Want It Done Right Now: Experience from Programming Hard Real Time Systems in Xenomai for the Magdalena Ridge Observatory Interferometer

Eugene B. Seneta[a], John S. Young[a], David F. Buscher[a], E. Robert Ligon[b], Dylan Etscorn[b], and Allen Farris[b]

[a]Cavendish Laboratory, University of Cambridge, United Kingdom
[b]Magdalena Ridge Observatory/New Mexico Tech, Socorro, NM, USA

ABSTRACT

Xenomai[1] is a hard real-time operating system suitable for many low-latency tasks encountered in astronomical instruments. It is open source, has microsecond-level response time and coexists with the Linux kernel, thereby facilitating the execution of hard real time code on Linux systems. This presentation presents experience coding systems with Xenomai for the Magdalena Ridge Observatory Interferometer.

Firstly an overview of Xenomai is given, focusing on how it achieves hard real time performance and how it can be used to interact with hardware using Linux-like device drivers. Secondly, a generic outline of the development process is given, including the mindset needed, general pitfalls to be avoided, and strategies that can be employed depending on how open the hardware and any existing source code is. Two specific case studies from the Magdalena Ridge Observatory are then presented: Firstly, the fast tip-tilt system, which must read out a 32x32 subframe from an EMCCD camera, determine a stellar image centroid and send a correction voltage to a tip-tilt mirror at up to 1kHz. Secondly, the MROI delay line metrology system, which must read laser metrology position data for ten delay line trolleys and send correction voltages to their cat's eyes at 5kHz.

Finally, some future challenges to development with Xenomai and other hard real time operating systems are discussed: processors with functionality such as system management interrupts that are beyond operating system control, and the trend towards buffered or closed interfaces between computers and hardware.

Keywords: Xenomai, real time, instrumentation, interface, servo, Magdalena Ridge Observatory Interferometer, MROI

1. INTRODUCTION

Real time low-latency tasks are commonplace in modern observatories. These frequently take the form of control loops that compensate for rapid perturbations: telescopes must move smoothly to their target positions and perform precise sidereal tracking despite the mechanical imperfections of their motors and gearing; adaptive optics systems must compensate for atmospheric turbulence, instrumental misalignments and vibration; optical paths must be controlled precisely despite the imperfect trajectories of optomechanical mounts.

Motor control can often be implemented using industrial off-the-shelf controllers, but other tasks most likely require a more customised and flexible approach. One method is to use a computer to measure perturbations, compute an appropriate response and move actuators to achieve compensation. However, to maximise performance, care must then be taken to minimise the propagation delays. This can often be achieved through the appropriate use of a hard real-time operating system such as Xenomai,[1] which seeks to minimise execution time (latency) and variations in execution time (jitter) in the control loop code, prioritising this over all other computing tasks. When this is achieved, execution is said to be deterministic.

Further author information:
E.B.S.: E-mail: bodie@mrao.cam.ac.uk

Software and Cyberinfrastructure for Astronomy VI, edited by Juan C. Guzman, Jorge Ibsen, Proc. of SPIE
Vol. 11452, 114521G · © 2020 SPIE · CCC code: 0277-786X/20/$21 · doi: 10.1117/12.2562191

In this paper we discuss our experience implementing highly effective hard real time control loops using Xenomai at the Magdalena Ridge Observatory Interferometer (MROI) in New Mexico, USA.[2] The MROI is designed to interferometrically combine light from up to ten telescopes, so has additional control loop requirements beyond those of a single-telescope observatory — it is effectively an optomechanical machine hundreds of meters in diameter. The interferometric combination requirement means each telescope beam needs its optical path length to be matched with the others to nanometre precision, even though each optical delay demand varies with sidereal motion of the celestial source and atmospheric turbulence.

2. XENOMAI OVERVIEW

Xenomai has been chosen for a number of reasons. Firstly, it is free, open-source, and actively maintained. Secondly, it coexists with the popular Linux kernel, so that the richness of the Linux software ecosystem can be leveraged for non-time-critical supporting code. Finally, a wide range of interface devices are available for personal computers running Linux, and it is frequently possible to port their device drivers to Xenomai, or to write supporting Xenomai drivers for critical code sections.

Xenomai has been described in detail by its developers.[3–5] It is available in two architectures, "cobalt" and "mercury". Cobalt is a lightweight hard real-time kernel that intercepts events (such as hardware interrupts) and processes them before (optionally) passing them on to the Linux kernel. Mercury is a set of libraries for Linux (optionally with PREEMPT-RT patches). For MROI development, cobalt is used as it offers better performance.[6]

Cobalt follows the familiar Linux programming model in that user-space code communicates with device driver code via file operations like `read()`, `write()` and `ioctl()`. For user-space development, MROI projects use Xenomai's real-time implementation of the POSIX `pthreads` programming interface, falling back to various Linux libraries for non-critical code. For driver development, Xenomai's native RTDM interface is used. All development is in C.

It is possible to use Linux system calls in cobalt, so any Linux library can be linked to a Xenomai application and any Linux kernel call can be made from a Xenomai driver. However, when execution encounters such a call (for file access, for example), the program continues execution in "secondary mode" where there are no hard real-time guarantees. When a Xenomai system call is then encountered, hard real-time "primary mode" execution resumes. Therefore, for hard real-time performance, the developer must ensure that no Linux system calls are made in critical sections of their application or driver.

Some hardware is natively supported by Xenomai: a variety of analogue interface cards, some popular ethernet cards, and RS-232 serial ports. Other devices will require driver development under cobalt, as no device manufacturer that we are aware of supports it.

3. DEVELOPMENT OVERVIEW

Here we present an overview of control loop development for the MROI, starting with hardware decisions and moving on to the software framework and a generic overview of how applications are written.

3.1 Hardware Decisions

Xenomai is available for a variety of popular processors, including those with Arm- and Intel-style architectures. For the MROI we have been developing exclusively for Intel-style computers, and this has worked well, although it may not be the best choice for future projects (as discussed in Section 8).

The Peripheral Component Interconnect (PCI) and PCI Express (PCIe) interfaces have been chosen for all hardware running under Xenomai: firstly, these interfaces are well supported in both Linux and Xenomai; secondly, there is much flexibility in choice of cards; and finally, latency is low because the interface bandwidth is high, there is no significant buffering across the bus and Direct Memory Access (DMA) transfers and interrupts are part of the specification.

Specific cards have been chosen for functionality and ease of driver development. Preference is given to devices that are so well documented that drivers can be developed from scratch. We avoid devices that can only

be controlled through opaque libraries and drivers provided by their manufacturers, as it is difficult to adapt such devices to Xenomai.

3.2 Software Design

Figure 1 presents an overview of how a hard real-time Xenomai application is typically structured at the MROI. There is a physical process (for example, an active optical alignment) that forms part of a rapid feedback loop. A computer card measures it in some way and another produces a signal that has the desired response (these could in principle be different functions of the same card). The purpose of the Xenomai application is to minimise the time from measurement to output and to ensure that the time never exceeds some specified latency, regardless of whatever else the computer may be doing.

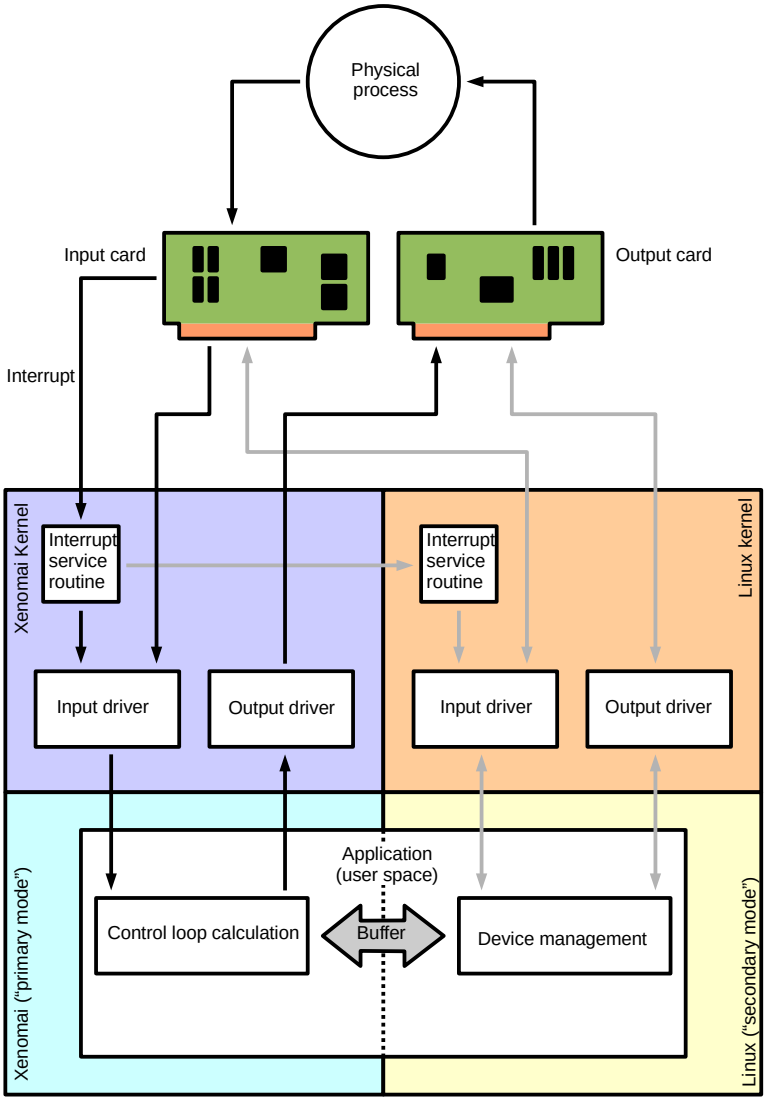

Figure 1. General overview of application implementation in Xenomai at the MROI. Arrows are coloured black for hard real time priority tasks, grey otherwise.

The application must firstly initialise the input and output hardware and later on may need to perform other device housekeeping tasks such as changing the configuration. Such tasks have no hard real-time requirements,

so conventional Linux code and a conventional device driver (such as one provided by the manufacturer) can be used for these purposes, just as they would in a pure Linux system. For these tasks, the application runs in secondary mode.

Once initialised, the application is ready to receive hardware interrupts from the input card. A Xenomai device driver is written that includes an interrupt service routine to handle the interrupt. It might read measurement data from the card and forward it on to user-space, or notify the user-space program that DMA data from the card has arrived. It might also forward the interrupt to the Linux kernel if the conventional driver needs it.

Once the input data arrives in user space, primary mode code performs the control loop calculation in hard real-time and sends the result to the output Xenomai driver, which in turn writes to the output card, thereby closing the loop. It would be quicker to process the control loop data entirely within the Xenomai drivers, thereby avoiding context switches to and from user-space. However, this is usually not possible, because the necessary functionality (floating point arithmetic, allocation of large memory blocks) is only available in user-space.

There are many inherently non-real-time operations that must be avoided within the critical code. These include memory allocation and initialisation of memory maps, many math library functions, hard-disk operations, network interactions, and display operations.

Memory allocation and mapping are best handled during initialisation, when timing is not critical. Actual access can occur without breaking hard real-time determinacy. Calculations requiring square roots or transcendental functions can be dealt with in a number of ways: depending on the accuracy required, an algebraic approximation (such as the first part of a Taylor series) can be used; a lookup table can be precalculated with values linearly interpolated in critical code; sometimes code can be rewritten to avoid the issue. For example,

```
if (z < sqrt(pow(x,2) + pow(y,2))...
```

could become

```
if (z*z < x*x + y*y)...
```

For system operations, typically the solution is to run critical and non-critical code in separate threads, with buffered communication between them. That way, the non-real-time threads only need to be able to keep up "on average".

4. CASE STUDY: MROI FAST TIP-TILT SYSTEM

4.1 Overview

An MROI fast tip-tilt system is to be installed at each telescope in the array. It compensates for rapidly varying wavefront tip and tilt introduced at each telescope pupil by the atmosphere.[7] A beamsplitter at the Nasmyth port of each telescope diverts light from the science beam which is then focused into an image on an electron multiplying CCD (EMCCD) camera with a frame rate of up to 1 kHz. A Xenomai system reads out each image, computes a centroid position error in hard real-time and sends compensating tip and tilt voltages to a piezo mirror in the light path.

The first tip-tilt system underwent site acceptance tests in November 2018. Another system is under construction.

4.2 Timing Requirements

The system is required to deliver a closed-loop bandwidth of 40 Hz on bright targets, with a goal of 50 Hz. For sources approaching the target limit (14th magnitude) this is relaxed to 15 Hz. Furthermore, the total time lag (half the exposure time, plus the readout time, the centroid computation time and the mirror actuation time) should not introduce a phase lag of more than 25°. For control loop bandwidths of 50 Hz, 40 Hz and 15 Hz the corresponding allowed time lags are 1.39 ms, 1.74 ms and 4.63 ms respectively.

4.3 Hardware

The camera is an Andor iXon X3 897, chosen for its low power consumption and open-source Linux device driver. It has a 512×512 pixel frame transfer EMCCD[8] which is used in full at the MROI as a narrowfield acquisition device, but it is also possible to read out a central 32×32 subframe at up to 414 Hz. This "conventional clocking" mode is very inefficient, as an entire 512 pixel line of the CCD must be clocked out to get the central 32 pixels of interest. Alternatively, a special "custom clocking mode" can be used to increase the frame rate up to 1140 Hz at the cost of increased noise and imaging artifacts.

The camera is connected by cable to an Andor CCI-23 PCI card in a host computer. Pixel clocking and digitisation occur concurrently with data transmission to the PCI card and writing to computer memory via DMA. No hardware documentation for the PCI card is available, but Andor provides the aforementioned open-source linux device driver and a closed-source user-space library.

The card that drives the tip-tilt mirror is an Advantech PCI-1716, chosen because its architecture is so well documented that it is possible to write a Xenomai device driver from scratch.

The host computer is a conventional rack-mounted industrial PC with several PCI slots. The processor is an Intel Core 2 Duo operating at 2.8 GHz. It is running Linux with a 3.14.17 kernel and Xenomai 2.6.4 patches.

4.4 Software Design

The software architecture is illustrated in Figure 2. The camera is configured to make regular exposures according to its internal timer. When an image is fully written into computer memory by the PCI card, it triggers an interrupt. The image centroid is computed in hard real time and compared with a requested position, taking care to avoid calling math library functions as described in Section 3. The resulting correction signal is sent to the analogue output card, which produces tip and tilt voltages that in-turn move a fast tip-tilt mirror to keep the image centroid at the requested position.

The design is complicated by two factors. Firstly, in Xenomai, floating point calculations must be performed in user-space. Hence image data must be read into user space before the centroid calculation is performed and the resulting analogue correction must be sent back to kernel space so that the analogue driver can deliver the result. Secondly, it is necessary to rely on the Andor library and non-real-time driver to configure and run the camera, because no information on the communications protocols for the PCI card is provided.

However, a workaround is possible to achieve hard real time performance. A Xenomai device driver is written containing a Xenomai port of the Andor driver's interrupt service routine[*]. This routine, instead of Andor's routine, runs when an interrupt from the PCI card arrives. It makes the DMA image data available to a Xenomai user-space program and also sends a message to the Andor driver to tell it that the data is available. This latter action is necessary because the Andor driver and library expect data to arrive, even though the Xenomai control loop has no use for it.

4.5 Performance

Latency can be gauged by measuring the time between the end of an exposure (when a "FIRE" line on the camera body goes low) and a change in the analogue output voltage. Additionally the PCI interrupt line can be monitored to determine when data is available for processing. When clocking out a central 32×32 central region, the total latency is 2.7 ms for conventional clocking and 1.1 ms for fast clocking. The majority of this is CCD readout and transfer time — just 58 µs is spent processing the image data and changing the output voltage. Jitter is typically 40 µs (one outlier of 90 µs has been seen).

To meet the requirement that less than 25° of phase change is introduced by the system, this translates into maximum achieveable control loop bandwidths (that is, with zero exposure time) of 26 Hz with conventional clocking and 63 Hz with the fast readout scheme. The 40 Hz bandwidth requirement is therefore met, at least for bright targets when fast clocking can be used.

The first tip-tilt system has been installed and tested at the MROI. Its on-sky performance is described in Ref.10 (at this meeting).

[*]This is possible (and legal) because Andor's driver is open-source and released under the GPLv2 licence.[9]

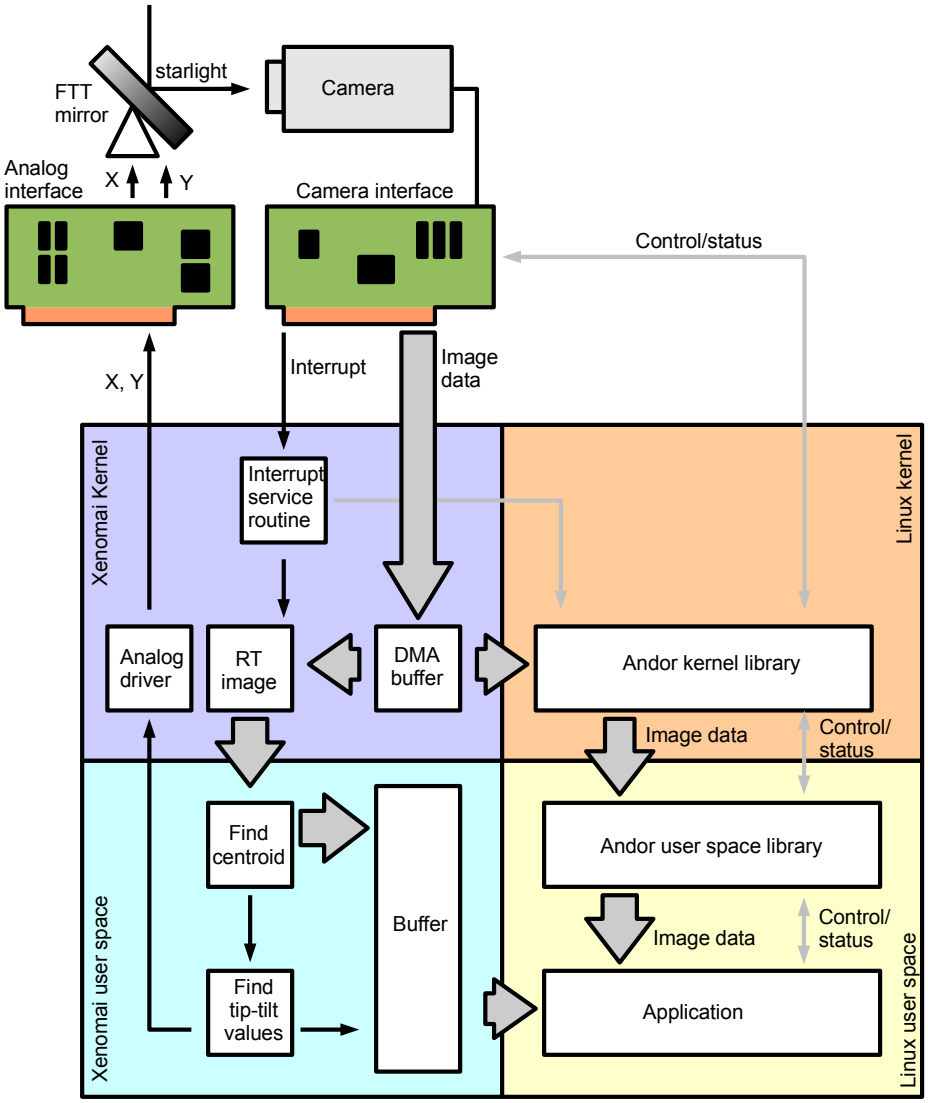

Figure 2. Overview of the fast tip-tilt software architecture.

5. CASE STUDY: MROI DELAY LINE METROLOGY SYSTEM

5.1 Overview

At the MROI, the science beams from each telescope must be combined interferometrically, requiring variable optical delay with nanometre precision for each beam. Optical delays will be implemented by ten delay lines, each consisting of a cat's eye retroreflector flexure-mounted on a cylindrical trolley that moves along a horizontal 200m long evacuated pipe, introducing the necessary and ever-changing optical delay as the sidereal source is tracked across the sky.[11] One delay line is already operational in 100m of pipe, another is nearing completion.

A computer running Xenomai is responsible for closing the control loop that maintains each optical delay correctly (Figure 3). A laser metrology system measures the round-trip optical distance to the cat's eye and back, relative to a datum that is checked during initialisation. The application checks the current reading and compares it with the required position (it will eventually also make adjustments according to fringe tracker measurements[12]). It then sends an error-correction signal to the cat's eye via a low-latency analogue path that

includes a differential line driver and an FM radio transmitter and receiver. The receiver drives a voice coil attached to the cat's eye to drive it backward or forward in order to minimise the error. The trolley has an onboard computer that drives the trolley along the pipe to keep it underneath the cat's eye, thereby extending the range to the full pipe length.

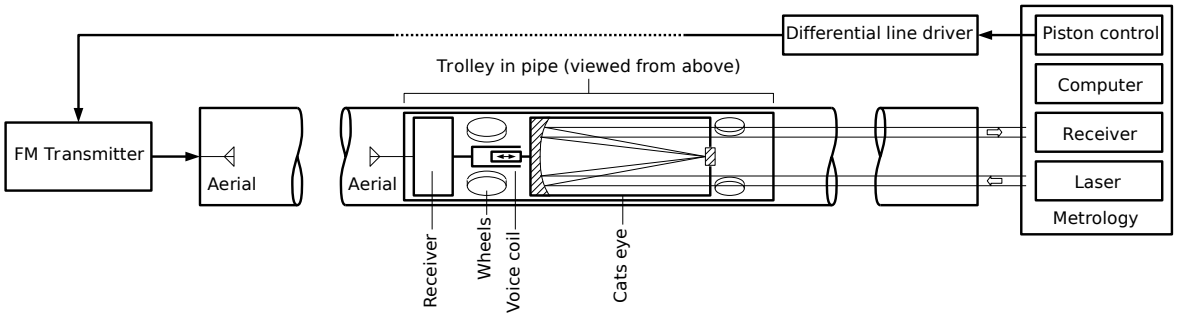

Figure 3. Overview of the metrology system control loop.

5.2 Timing Requirements

It is a system requirement that the control loop 3 dB frequency is greater than 100 Hz. A sample period of 200 µs meets the bandwidth requirement and provides an additional safety margin. Also, the control loop latency for each delay line must be less than 40 µs.

5.3 Hardware

The hardware occupies a VME crate. The control computer is a Concurrent Technologies VP 325 022-23U with a Pentium-M 1.6GHz processor. It runs a Linux 2.6.29.4 kernel patched with Xenomai 2.4.8-2. The computer contains a Tundra Universe II bridge to the VME bus, which makes the VME bus appear as a native PCI bus peripheral. There is also an Intel ethernet port supported by Xenomai.

The other peripherals are all VME bus cards. The ones of interest are:

- Metrology board: an Agilent 10898A, capable of measuring two delay line trolley positions. Eventually there will be five boards installed for the ten delay lines.
- Timer board: a Symmetricom TTM635VME. This board is configured to generate pulses at 200 µs intervals, synchronised to one-second epochs by a GPS input. The pulses are tied to the VME bus SYSFAIL interrupt line.
- Analogue output board: An Acromag IP220 on a Tews Technologies TVME200 carrier board.

5.4 Software Design

The computer interfaces for all of these components, including the Universe II Bridge, are very well documented and hence it has been possible to write complete in-house Xenomai drivers for all of them.

The Universe II is configured to memory map the VME register space into 64 KiB of PCI memory space. The SYSFAIL interrupt is mapped to a PCI interrupt. Using an oscilloscope to measure timings, 16-bit data reads and writes of VME addresses take 1 µs and 0.3 µs respectively, while interrupt entry latency is 5 µs. The slow read and write times across the VME bus appear to be a property of the Universe II bridge (they have also been observed in QNX). As a measurement cycle is just 200 µs long, it is clear that VME bus transactions should be kept to a minimum.

The software uses the following execution strategy: when an interrupt is received, the interrupt service routine firstly reads the system clock, then latches and reads the current time from the timer card. The offset is then

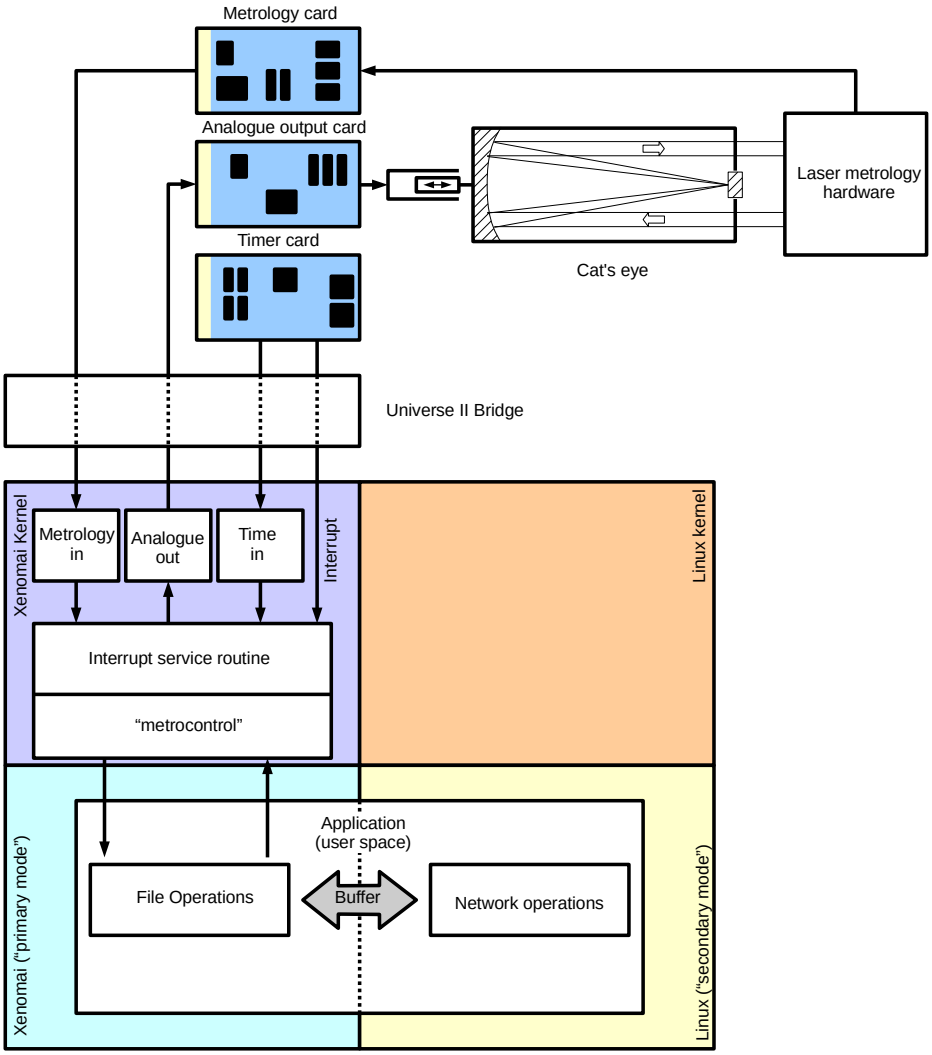

Figure 4. Overview of the metrology system software architecture.

calculated, so that further time measurements within the routine can be made solely with the system clock (it should not drift significantly within a 200 µs period). This avoids further timer card reads across the slow Universe II bridge. Hence accurate timestamps can be produced within the routine that compensate for any interrupt entry latency jitter.

Each delay line is then considered in turn. A trolley position is firstly read from the metrology card. A demand position is then interpolated from a preloaded set of trajectory waypoints and the current time (optionally a fringe tracker offset is added, see Section 7). The two values are compared to generate an error signal. This signal is written to the analogue output for the specific delay line. Because of the tight timing constraints, there is no context switching to user space — the entire calculation takes place using integer arithmetic within a Xenomai kernel module.

The architecture is shown in Figure 4. Within the Xenomai kernel, individual drivers handle communication with the metrology card, analogue output card and the timer card via the Universe II VME bridge. A fourth driver ("metrocontrol") contains an interrupt service routine, triggered by a timer card interrupt, that orchestrates the

hardware communication and performs the calculations needed to close the control loop.

Xenomai user-space code sends buffered trajectory demands to metrocontrol and receives telemetry data from it. In turn, this data is buffered to and from non-real-time threads that communicate with other computers on the network that send and receive this data.

5.5 Performance

Timing performance is illustrated in Figure 5. Here, the system is set up to manage two delay lines (a single metrology card can measure two channels and only one card is currently installed). A dual-channel oscilloscope measures the voltages of the VME bus SYSFAIL interrupt line (upper trace) and the least significant bit of the VME data bus (lower trace). The oscilloscope is triggered off the falling edge of the SYSFAIL line. The image is a 40 ms exposure, capturing 100 sweeps of each trace.

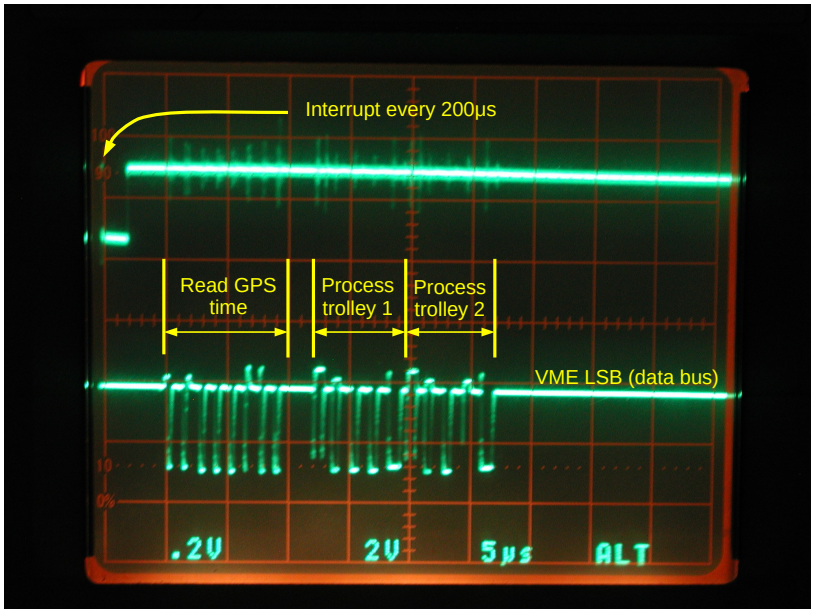

Figure 5. Oscilloscope trace of timing when the system is managing two delay lines. The VME bus SYSFAIL voltage is graphed against time in the upper trace and the voltage of the least significant bit of the VME data bus below it.

Transactions first occur about 5 µs after the SYSFAIL line goes low. This is an upper bound on the interrupt routine entry latency. The first group of pulses on the data line indicates the reading of the timer card registers to establish the offset between the timer card time and the system clock time. There is then a second burst, about 15 µs long, which indicates that during two consecutive 7.5 µs periods each of the two trolley positions is being read and an analogue compensating voltage applied. For each trolley, this meets the control loop latency requirement with 32.5 µs to spare (Section 5.2). For ten delay lines, an additional 60 µs would be needed, completing all bus transactions 92 µs after the SYSFAIL interrupt. Hence, 108 µs or 54% of CPU time is still available to run the application and other processes.

Finally, the traces are sharp despite the long exposure. This indicates that there is very little jitter present. No formal measurement of jitter has been made, however 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 ±15 ns, and the latter should be infrequent enough that the RMS jitter is still within specification. In any case, the technique of reading the time from the timer card within the interrupt allows interrupt entry jitter to be determined and compensated for.

The performance of the complete control loop is illustrated in Figure 6. Here, the trolley is tracking a demand position that changes at a constant speed of $15\,\mathrm{mm\,s^{-1}}$ within a delay line pipe that has not been evacuated

(the worst-case scenario). The control loop must compensate for mechanical imperfections in the trolley drive, pipe bends, seeing and vibration. It can be seen that the metrology error has a fixed offset of about 2.9 µm. The peak-to-peak variation is only 200 nm (this is physical trolley distance, double these values for the optical delay). Furthermore, the frequency response does not begin to roll off until beyond 100Hz. Both parameters are within specification. The fixed offset and slow variation are due to the tilt of the trolley within the pipe, they will be removed when the fringe tracker comes online.

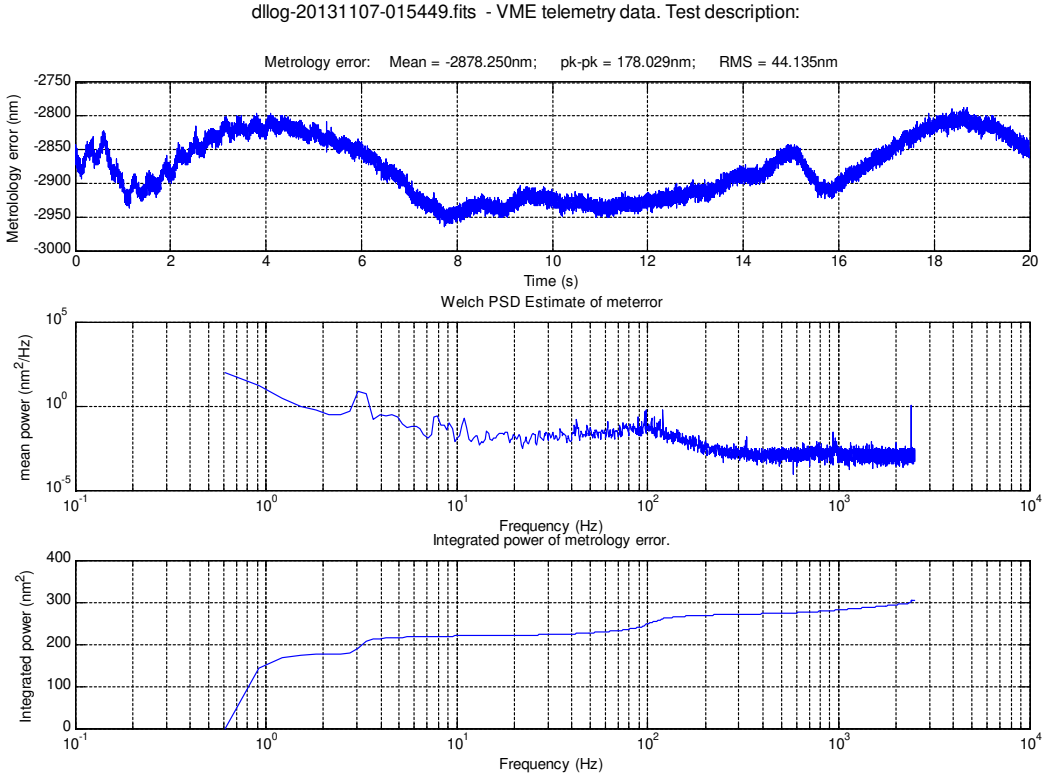

Figure 6. Worst-case delay line control loop performance. The delay line trolley is tracking a demand position that changes at a constant velocity of 15 mm s^{-1} in air.

6. RECENT DEVELOPMENTS

Xenomai 2 is no longer supported, and the computers that the fast tip-tilt system and the metrology system run on are no longer manufactured. Consequently, both systems have been ported to Linux 4.14.111 patched with Xenomai 3.0.8 running on contemporary computers. Performance has not been formally measured, but qualitatively is at least as good as the systems described in this paper.

7. FUTURE WORK

A third project currently underway is the MROI fringe tracker.[12] This system measures errors in the positions of the delay line trolleys (Section 5) by monitoring interference fringes from combined science beams. These fringe patterns are imaged by a SAPHIRA infrared array detector[13] at 1 Hz to 1 kHz using hardware provided by the European Southern Observatory.

The data will be sent to the delay line metrology system to implement the corrections. It is a requirement that the transmission latency is no more than 200 µs, and furthermore that the latency of feedback from the metrology system is no more than 500 µs. As Xenomai supports hard real time transmission over ethernet ("RTNet"), this is achieved via a dedicated ethernet link. No formal in-house measurements of RTNet latency have been made, but others have found the one-way latency to be just over 100 µs with jitter just over 3 µs using a similar point-to-point link.[14]

8. FUTURE CHALLENGES

These case studies show that implementing control loops with Xenomai is currently a practical proposition. However, recent technical developments make this more difficult — not just for Xenomai, but for all hard real-time operating systems.

Off-the-shelf hardware increasingly connects to computers via interfaces that are non-deterministic or closed-architecture (or both). If USB is used, for example, data transmission is only sent in response to a request from the host. Even in the most deterministic case (an "interrupt transfer"), transfer is governed by 125 µs time slots generated by a clock in the USB hardware. Not only is significant latency introduced, but the USB clock can alias with the control loop cycle period. Other interfaces can be problematic because their architecture documentation requires licensing. For example, industrial cameras are often supplied with Camera Link, GigE Vision, USB3 Vision or Firewire interfaces, but only Firewire is completely open (USB3Vision is partially open). In these cases, a Xenomai developer is completely dependent on the vendor making their device driver open enough that access to any DMA memory is possible, and that the format of that data is straightforward to decode.

Another issue is the increasing use of system management modes in both Intel- and ARM-style processors, where code executes at a higher priority than the operating system in response to a specialised interrupt. This makes it possible for a hard real-time process to lose control of the processor at any moment for an indeterminate amount of time, undermining the system's determinacy. This functionality was originally used for power management but in recent years the scope has broadened to include USB and Trusted Platform support, thereby increasing the risk of outages during control loop execution.

Xenomai offers workarounds for Intel-style processors,[15] however as computer motherboards are sometimes completely reliant on system management code this must be done with caution.

As the use of system management modes becomes more predominant it may become necessary to procure specialised processors and computers designed for hard real-time operation.

9. CONCLUSIONS

Xenomai has been used very effectively to implement the fast tip-tilt and delay line control loops at the MROI. Implementation of the MROI fringe tracker is also underway. These examples show that it is possible to construct hard real-time control loops of significant practical use in observatory settings using Xenomai, generic computers and commercial off-the-shelf hardware.

This approach requires that manufacturers' hardware and software is well documented and has a computer interface with minimal latency. Nevertheless, Xenomai allows the development of hard real-time device drivers and user-space code within a familiar Linux-like environment while allowing access to the rich Linux software ecosystem when hard real-time performance is not required.

ACKNOWLEDGMENTS

This material is based on research sponsored by Air Force Research Laboratory (AFRL) under agreement number FA9453-15-2-0086. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory (AFRL) and or the U.S. Government.

The authors are additionally grateful to the following for their contributions: Ratna Halder, Omid Hosseini, Rob Kelly, Jennie Maes, Dipanjan Das Roy.

Portions of the metrology system code are ported from QNX code written by Roger Boysen.

REFERENCES

[1] Kiszka, J. and Gerum, P., "Xenomai." http://www.xenomai.org. (Accessed: 23 October 2020).

[2] Santoro, F. G., Olivares, A. M., Salcido, C. D., Jimenez, S. R., Sun, X., Haniff, C. A., Buscher, D. F., Creech-Eakman, M. J., Jurgenson, C. A., Shtromberg, A. V., Bakker, E. J., Selina, R. J., Fisher, M., Young, J. S., and Wilson, D. M. A., "Mechanical design of the Magdalena Ridge Observatory Interferometer," in [*Optical and Infrared Interferometry II*], Danchi, W. C., Delplancke, F., and Rajagopal, J. K., eds., **7734**, 1412 – 1428, International Society for Optics and Photonics, SPIE (2010).

[3] Gerum, P., "Start_Here." GitLab, 3 February 2019 https://gitlab.denx.de/Xenomai/xenomai/-/wikis/Start_Here. (Accessed: 23 October 2020).

[4] Kiszka, J., "Xenomai 3: An Overview of the Real-Time Framework for Linux." Embedded Linux Conference, San Diego, California, April 4–6, 2016 https://elinux.org/images/7/76/Kiszka.pdf https://youtu.be/h7GXyy5QVCA. (Accessed: 23 October 2020).

[5] Karim Yaghmour, Jon Masters, G. B.-Y. and Gerum, P., "The xenomai real-time system," in [*Building Embedded Linux Systems*], Oram, A., ed., 365–386, O'Reilly Media Inc., Sebastopol (2008).

[6] Brown, J. H. and Martin, B., "How fast is fast enough? Choosing between Xenomai and Linux for real-time applications." Twelfth Real-Time Linux Workshop on October 25 to 27, Nairobi, Kenya, October 25–27, 2010 https://web.archive.org/web/20151004142300/https://www.osadl.org/fileadmin/dam/rtlws/12/Brown.pdf. (Accessed: 26 October 2020).

[7] Young, J., Buscher, D., Fisher, M., Haniff, C., Rea, A., Seneta, E., Sun, X., Wilson, D., Farris, A., and Olivares, A., "The performance of the MROI fast tip-tilt correction system," in [*Optical and Infrared Interferometry IV*], Rajagopal, J. K., Creech-Eakman, M. J., and Malbet, F., eds., **9146**, 574 – 588, International Society for Optics and Photonics, SPIE (2014).

[8] Teledyne e2V, "CCD97-00 Datasheet." https://www.teledyne-e2v.com/shared/content/resources/File/documents/Imaging%202017/EM%20Sensors/CCD97/1.%20BI/1485.pdf. (Accessed: 9 November 2020).

[9] The Free Software Foundation, "GNU General Public License, version 2." https://www.gnu.org/licenses/old-licenses/gpl-2.0.html. (Accessed: 13 November 2020).

[10] Buscher, D. F., Young, J. S., Seneta, E. B., Sun, X., Fisher, M., Haniff, C. A., Ligon, E. R., Olivares, A. M., and Salcido, C. D., "On-sky performance of the fast tip/tilt system for the Magdalena Ridge Observatory Interferometer (this meeting)," in [*Optical and Infrared Interferometry and Imaging VII*], Tuthill, P. G., Merand, A., and Sallum, S., eds., **11452**, International Society for Optics and Photonics, SPIE (2020).

[11] Fisher, M., Boysen, R. C., Buscher, D. F., Haniff, C. A., Seneta, E. B., Sun, X., Wilson, D. M. A., and Young, J. S., "Design of the MROI delay line optical path compensator," in [*Optical and Infrared Interferometry II*], Danchi, W. C., Delplancke, F., and Rajagopal, J. K., eds., **7734**, 1470 – 1488, International Society for Optics and Photonics, SPIE (2010).

[12] McCracken, T. M., Jurgenson, C. A., Young, J. S., Seneta, E. B., Buscher, D. F., Haniff, C. A., Creech-Eakman, M. J., Santoro, F. G., Shtromberg, A. V., Schmidt, L. M., and Rochelle, S., "The MROI fringe tracker: laboratory tracking with ICONN," in [*Optical and Infrared Interferometry IV*], Rajagopal, J. K., Creech-Eakman, M. J., and Malbet, F., eds., **9146**, 422 – 431, International Society for Optics and Photonics, SPIE (2014).

[13] Atkinson, D. E., Hall, D. N. B., Jacobson, S. M., and Baker, I. M., "The SAPHIRA detector: a near-infrared photon counter for astronomy," in [*Advanced Photon Counting Techniques XIII*], Itzler, M. A., Bienfang, J. C., and McIntosh, K. A., eds., **10978**, 41 – 47, International Society for Optics and Photonics, SPIE (2019).

[14] Barbalace, A., Luchetta, A., Manduchi, G., Moro, M., Soppelsa, A., and Taliercio, C., "Performance comparison of vxworks, linux, rtai, and xenomai in a hard real-time application," *IEEE Transactions on Nuclear Science* **55**(1), 435–439 (2008).

[15] Gerum, P., "Dealing with x86 SMI troubles." https://gitlab.denx.de/Xenomai/xenomai/-/wikis/Dealing_With_X86_SMI_Troubles. (Accessed: 17 November 2020).