Analysis of LinuxCNC Operating System Environment

Jeremy Tang

Washington Univesity in St. Louis  
1 Brookings Drive,

St. Louis MO 63130  
1 224 522 4485

jntang@wustl.edu

Stephen Decker

Washington Univesity in St. Louis  
1 Brookings Drive,

St. Louis MO 63130  
1 908 698 9147

sdecker747@gmail.com

Ethan Glassman

Washington Univesity in St. Louis  
1 Brookings Drive,

St. Louis MO 63130  
1 650 575 9193

ethan.glassman@gmail.com

ABSTRACT

In this paper we analyze the real-time performance of the LinuxCNC distribution.  In particular we study the performance of the kernel by writing a real-time application that writes a timestamped ‘hello’ message every millisecond for one minute, and analyze the results by comparing the actual time difference between ‘hello’ messages and the expected one millisecond difference.  We find that on a laptop under no load that the kernel performs with maximum difference of 1.1 millisecond, which occurred just one time and under full CPU load, the kernel holds to a difference of 1 millisecond 99.94% of the time.  We find therefore that the kernel in use on the LinuxCNC distribution is indeed real time.

CCS Concepts

* **Applied computing**
  + **Physical sciences and engineering**
    - **Engineering**
* **Software and its engineering**
  + **Software organization and properties**
    - **Software functional properties**
      * **Correctness**
        + **Real-time scheduleability**

**Keywords**

LinuxCNC, real-time, analysis, RTAI kernel, hardware abstraction layer, Arduino, CNC, manufacturing

INTRODUCTION

CNC machines are commonly used in industrial applications to accurately manufacture parts from plastics, woods, and metals such as aluminum or steel.  These machines can perform very precise operations down to the thousandth, or ten thousandth of an inch precision, and very fast operations at hundreds of inches per minute.  It is not uncommon for the mechanics and control hardware of the machines to last several decades, and outlive the control computer and software that was state of the art in the 1970s and 1980s, but are now hard to interface with modern CAD and CAM packages that engineers use to design new products.

LinuxCNC is a replacement machine control software that can replace the outdated control computer and control software that these older machines use, and allow them to be reintegrated with modern workflows at a fraction of the cost of new machines.  However, it is essential that the software controlling these machines runs in a real-time fashion in order to ensure safety of the machine operator and the machine itself.  Additionally, it is desireable to have robust, bidirectional, real-time control of the machines in order to take best advantage of the high speeds and accuracies of the machine.

We want to show, therefore, that the software does run in realtime during normal operation when commanding a machine’s operation.  Unfortunately due to complications with the code for the LinuxCNC software, and with our hardware limited to a laptop without a serial or parallel port, we chose to scale back our performance analysis.

A careful analysis of the differences between vanilla Linux and the distribution provided as a live CD for easy setup and installation (which will be a standard installation in use on real machines) suggested that there were two classes of changes made.  First, a hardware abstraction layer (HAL) was added to allow the kernel to communicate through virtual ports to real machines, avoiding the need to recompile the kernel for different hardware setups.  Second, a real time RTAI patched kernel is used between the HAL and the standard Linux completely fair scheduler kernel.  The normal kernel sees the hardware and hardware interrupts as software interrupts given the highest priority.

We chose to focus on an analysis of the kernel’s performance.  By verifying that the kernel was capable of real time operation, we could verify at least that is is possible for LinuxCNC to be real-time.

System[[1]](#footnote-1)

PREPARING THE SYSTEM

To analyze LinuxCNC we began by installing LinuxCNC on a Compaq Presario C762NRlaptop available for free usage.  This laptop was an older model equipped with an 1.73GHz Intel Pentium Dual-Core, three USB 2.0 ports, and a 100Mbps Ethernet port, originally running Windows Vista.  In order to best analyze the real world installation of LinuxCNC we downloaded and installed LinuxCNC 2.7 Wheezy following the directions as provided on for writing to and installing from a USB drive.[[2]](#footnote-2)

ARDUINO SETUP

In order to realistically, but safely, analyze LinuxCNC we wanted to setup a simple synthetic workload on an Arduino and were lucky to find that there was already an open source Arduino code base designed as a daughter board for LinuxCNC[[3]](#footnote-3) which we hoped would allow us to have a simple, safe synthetic workload that would interface with the LinuxCNC software. We were able to get the arduino and LinuxCNC to set up their initial handshake routine, but we were receiving no results.

Changes to arduino library:

* Modified my-mill.ini by commenting out PYVCP = custompanel.xml
* Modified custom\_portgui.hal by commenting out sets spindle-at-speed true
* Modified HAL2Arduino-0.6d.py to change port from

port = "/dev/ttyS" + str(i) to port = "/dev/ttyUSB0"

forcing the usage of USB

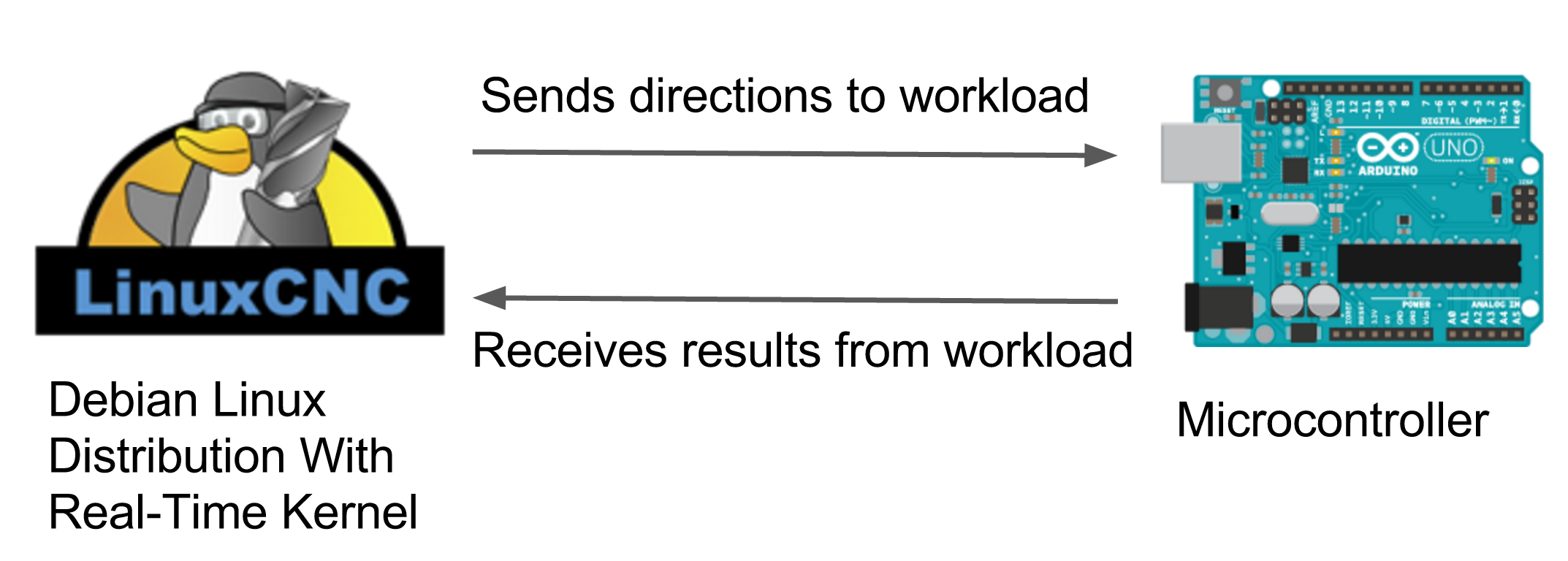


Figure 1: Our original system

SYSTEM ARCHITECTURE

Once we prepared the laptop, we began to study the system architecture once the laptop is prepared and we can focus our investigation.  In particular we considered <http://linuxcnc.org/docs/2.7/html/code/code-notes.html> where a diagram of the system architecture, as described by the LinuxCNC programming team, provided an ideal starting point.

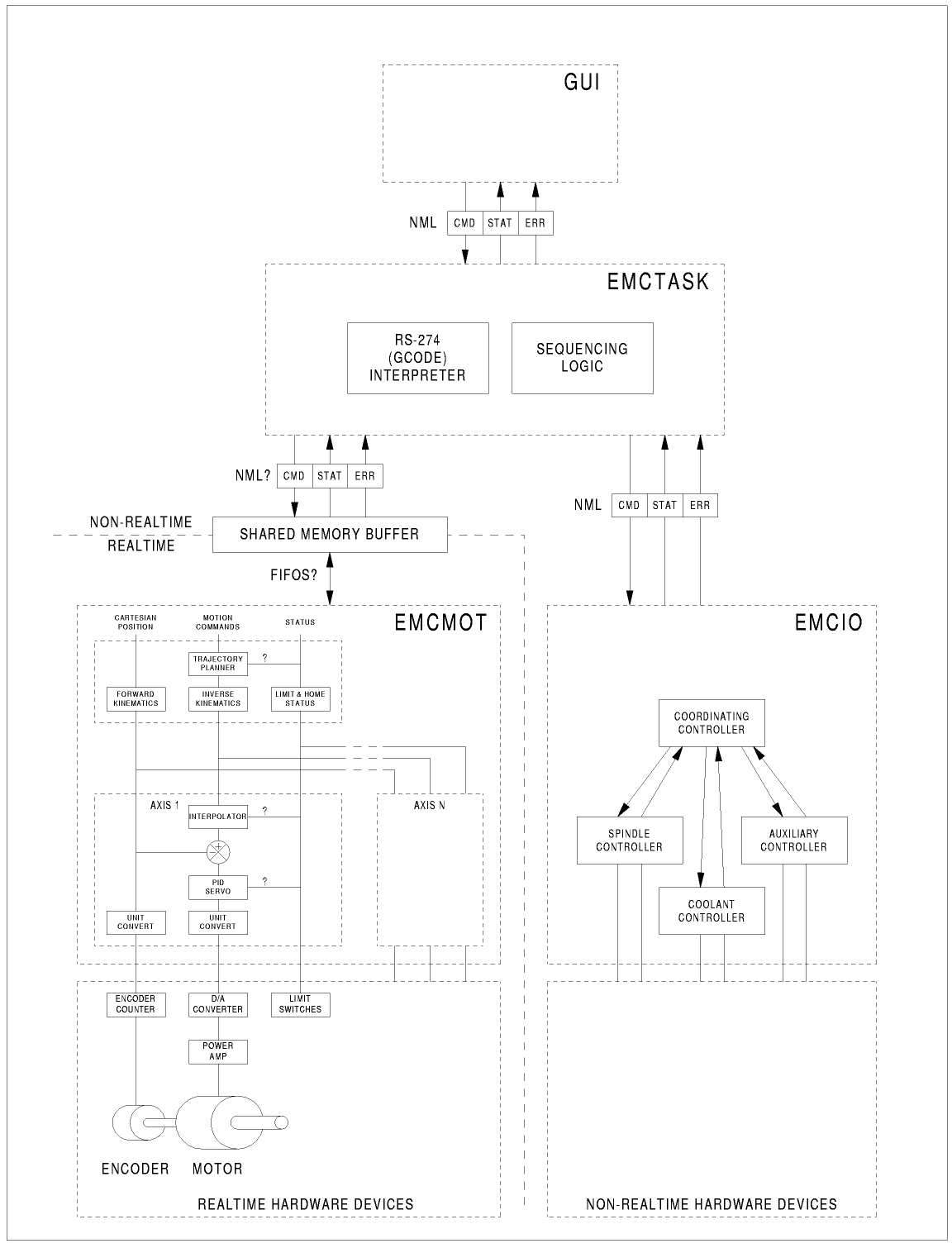


Figure 2: Full LinuxCNC architecture

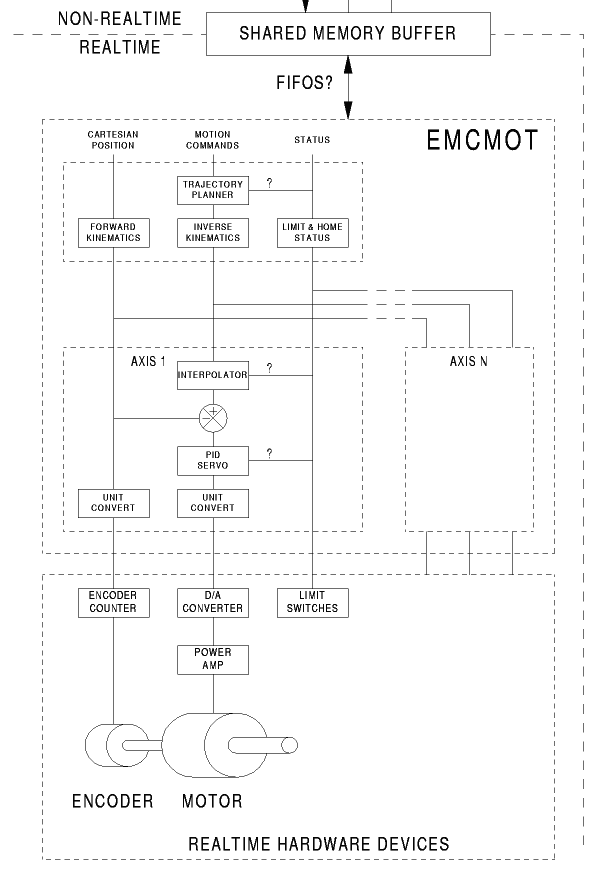


Figure 3: LinuxCNC realtime architecture only

There is a clear distinction between the realtime and non-realtime portions of the system, and we must clearly focus on the portions in the real-time section of the diagram in our investigation.  However, as we recall from the INTRODUCTION, the HAL is only one of two major types of changes made from vanilla Linux.  We intended to investigate the real-time portions of the LinuxCNC architecture but had difficulties getting using a USB connection between LinuxCNC and the arduino.  (See USB ISSUES for details)

We pivoted the focus of our investigations to primarily considering the kernel of the provided operating system environment, assuming that without building a good foundation (a real-time kernel) it is impossible to build a good building (a real-time program).

Kernel

Analyzing Kernel

To analyze the real-time kernel, we needed to write a simple workload application capable of running in real-time. We started with studying how the LinuxCNC application establishes real-time behavior, and learned that it uses a custom real-time application programming interface (RTAPI) that acts as a compatibility wrapper between LinuxCNC and a variety of real-time and non-real-time kernels. Given that the source code of LinuxCNC included a number of examples, we decided to design our real-time application using RTAPI.

We had a lot of trouble getting the examples to compile. It turned out that the examples were not compiled in the current versions of LinuxCNC, so we had to revert to a LinuxCNC build from February 8, 2006, and even then we had to hack the makefiles. Once we had the examples compiled and loaded, we could then begin writing our real-time application.

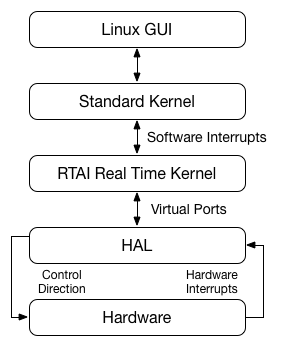


Figure 4: Kernel architecture

Sample Code

Real-time kernel space code

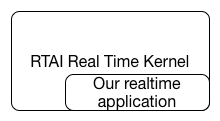


Figure 5: Real time kernel application location

User space code

In contrast to the real-time kernel module, the application we wrote to collect data in user-space is based on just the the standard GNU C libraries. The application, titled “simpletimer,” has the same role as the kernel module: run for 60 seconds, record a timestamp every millisecond, and write the output to a file. Unlike the real-time kernel module, the timing performance of the user-space application is dependent on the scheduler of the standard Linux kernel.

These applications were each run when the laptop was idle and when the laptop was at full CPU utilization. We used a program called “stress” to load the CPU during the tests.

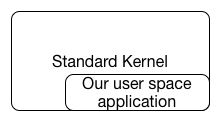


Figure 6: User space application location

Results

Plots

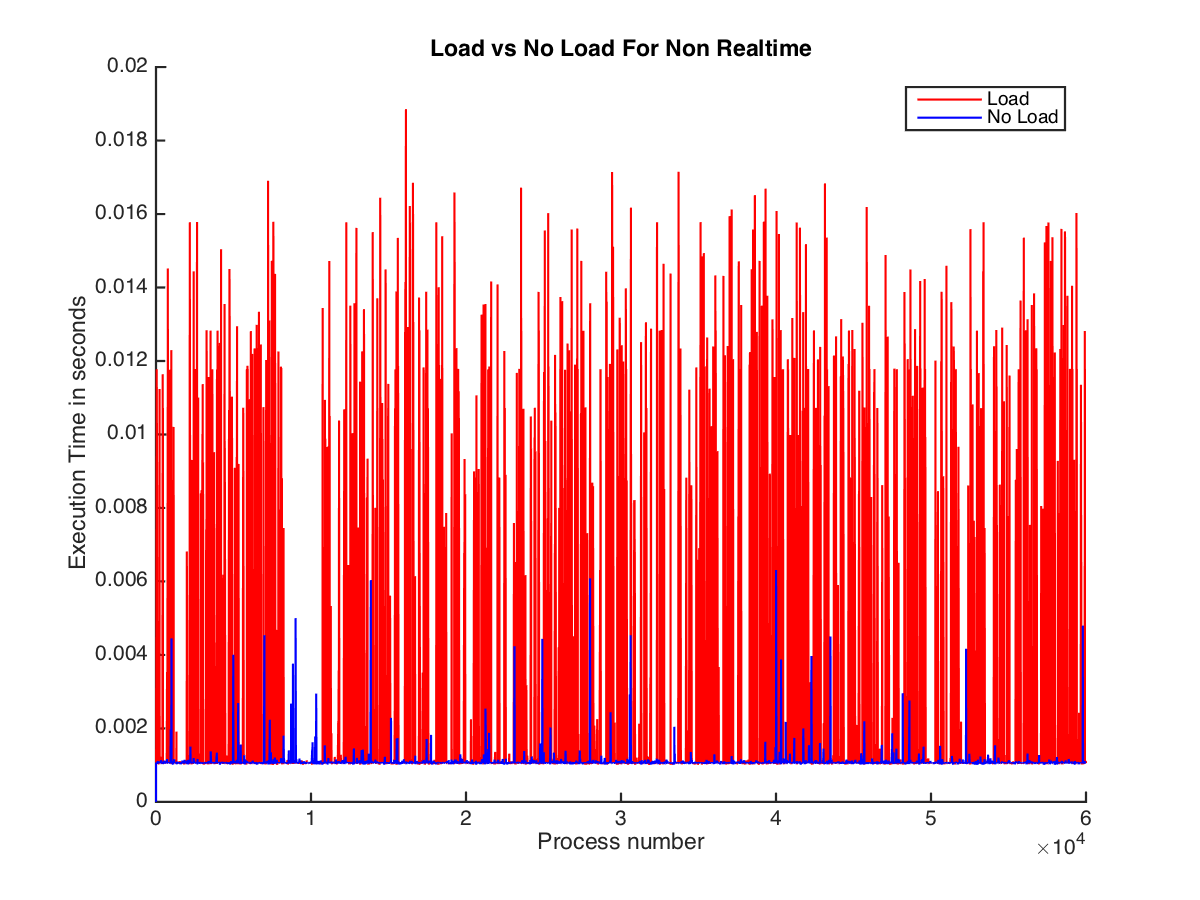


Figure 7: Non realtime

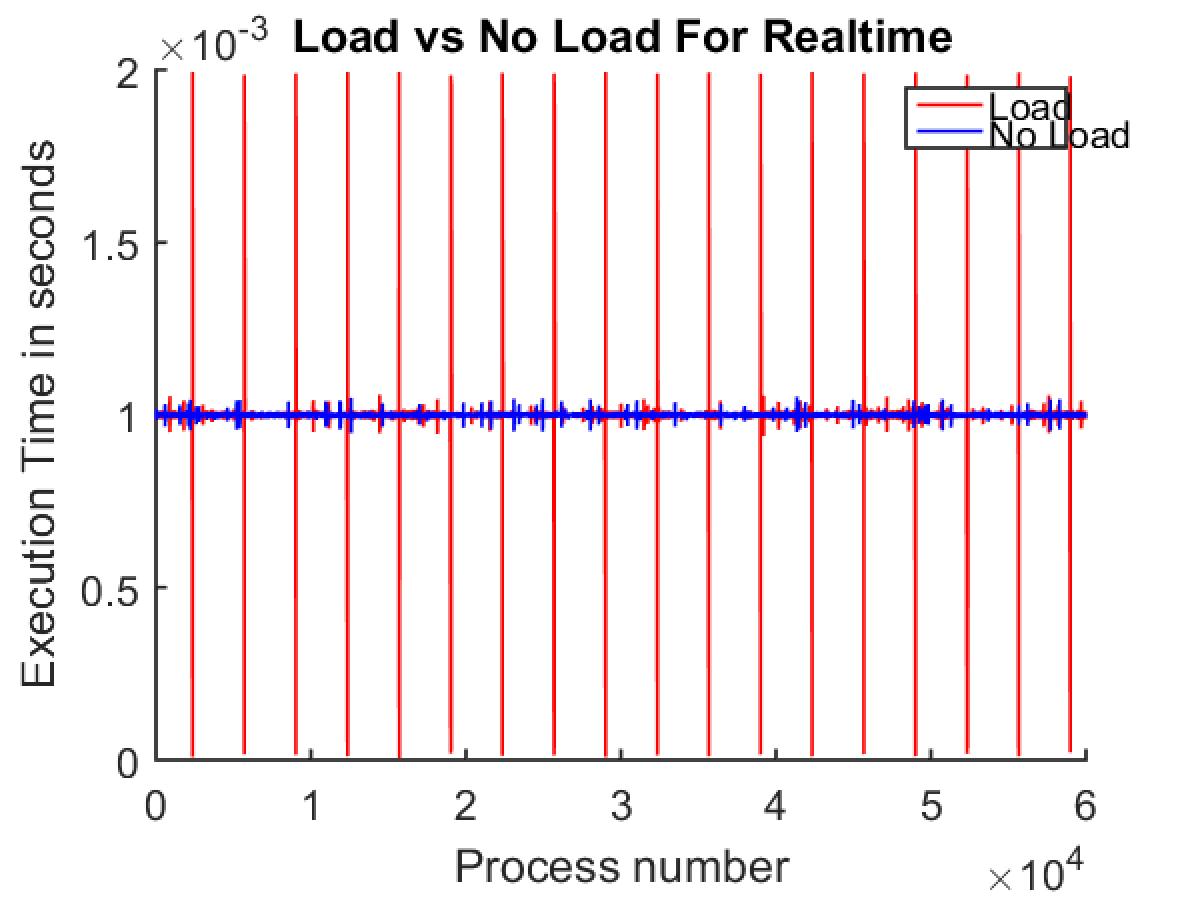


Figure 8: Realtime

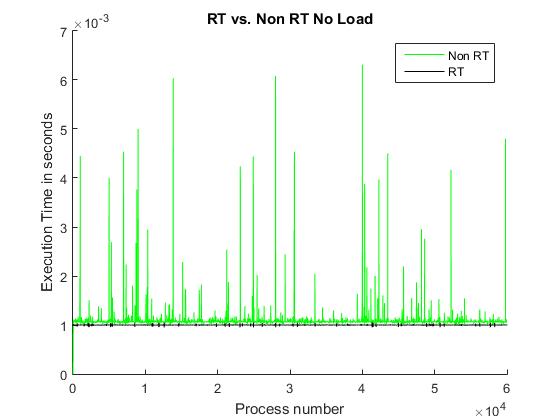


Figure 9: Compare no load

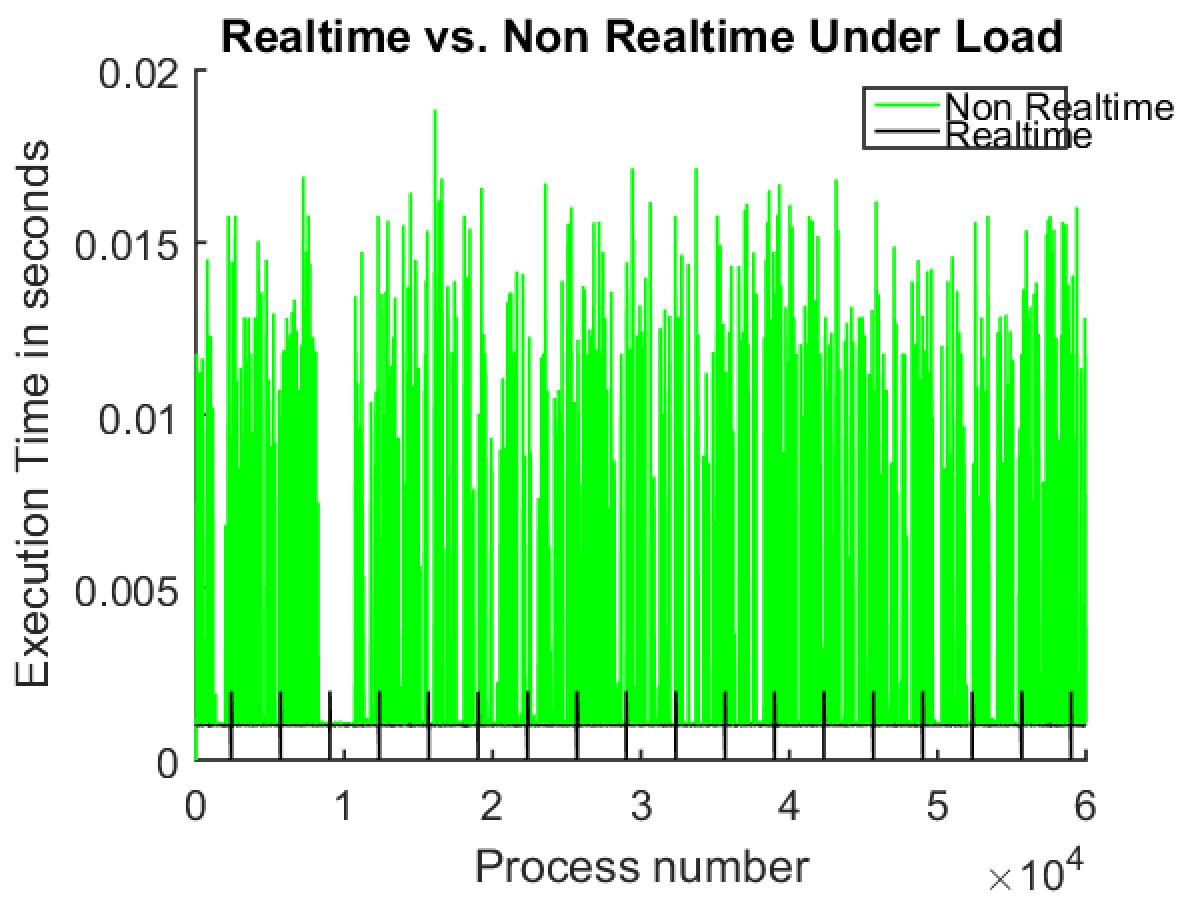


Figure 10: Compare under load

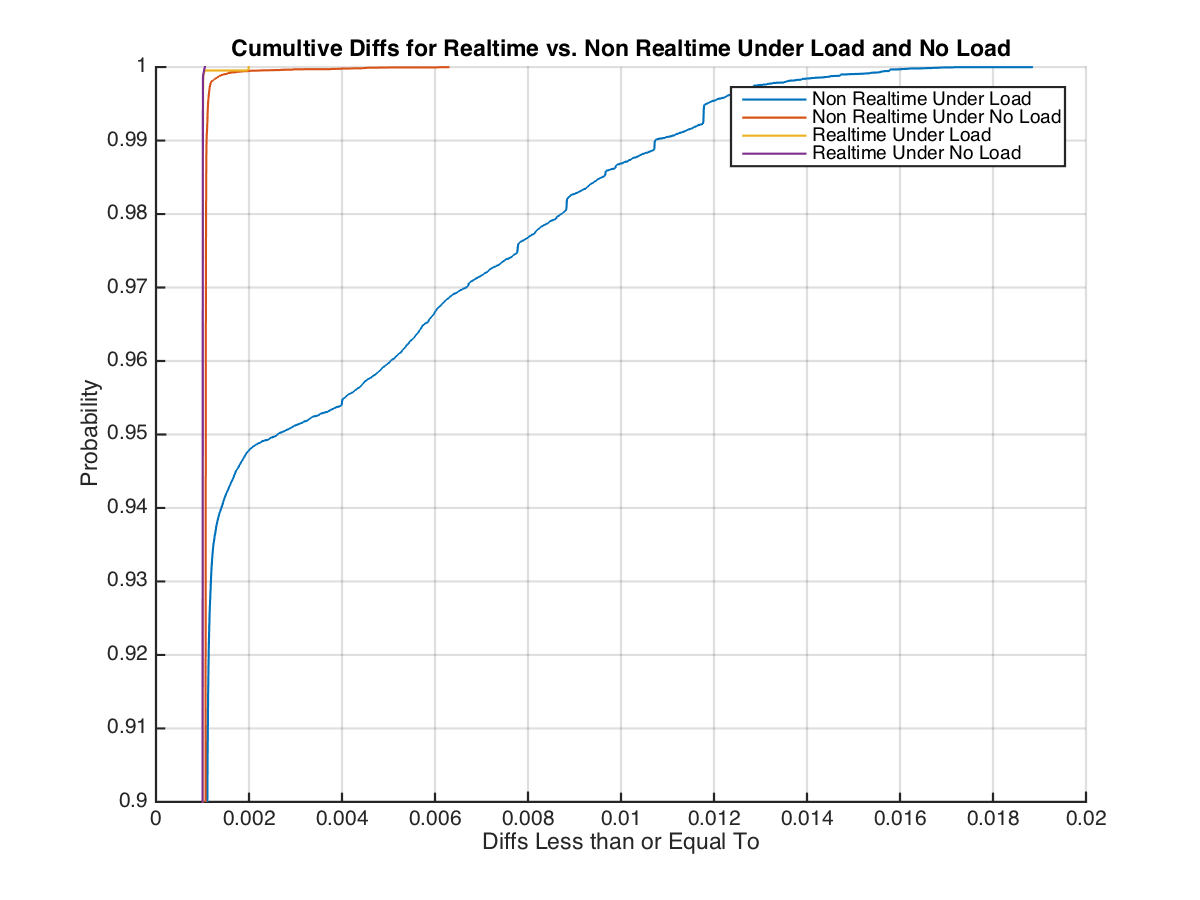


Figure 11: CDF no zoom

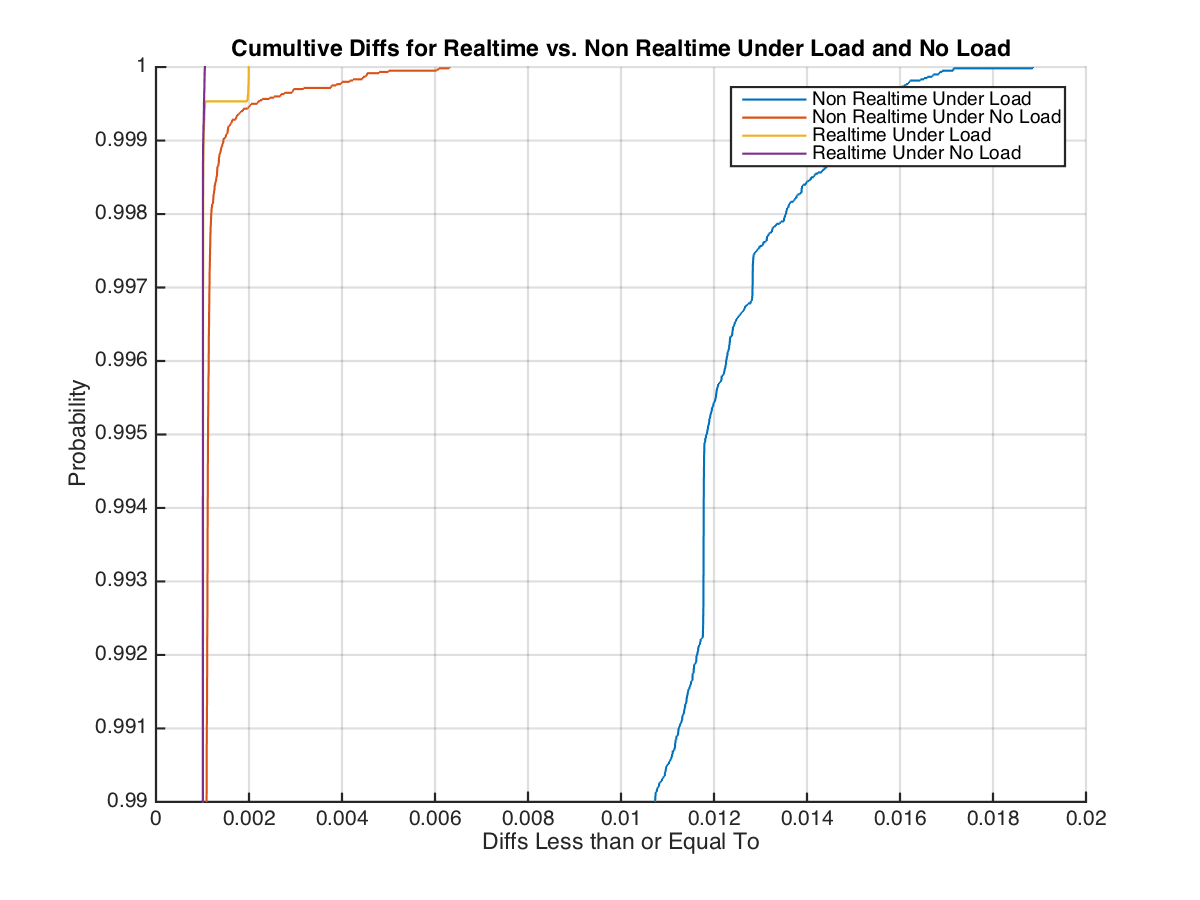


Figure 12: CDF partial zoom

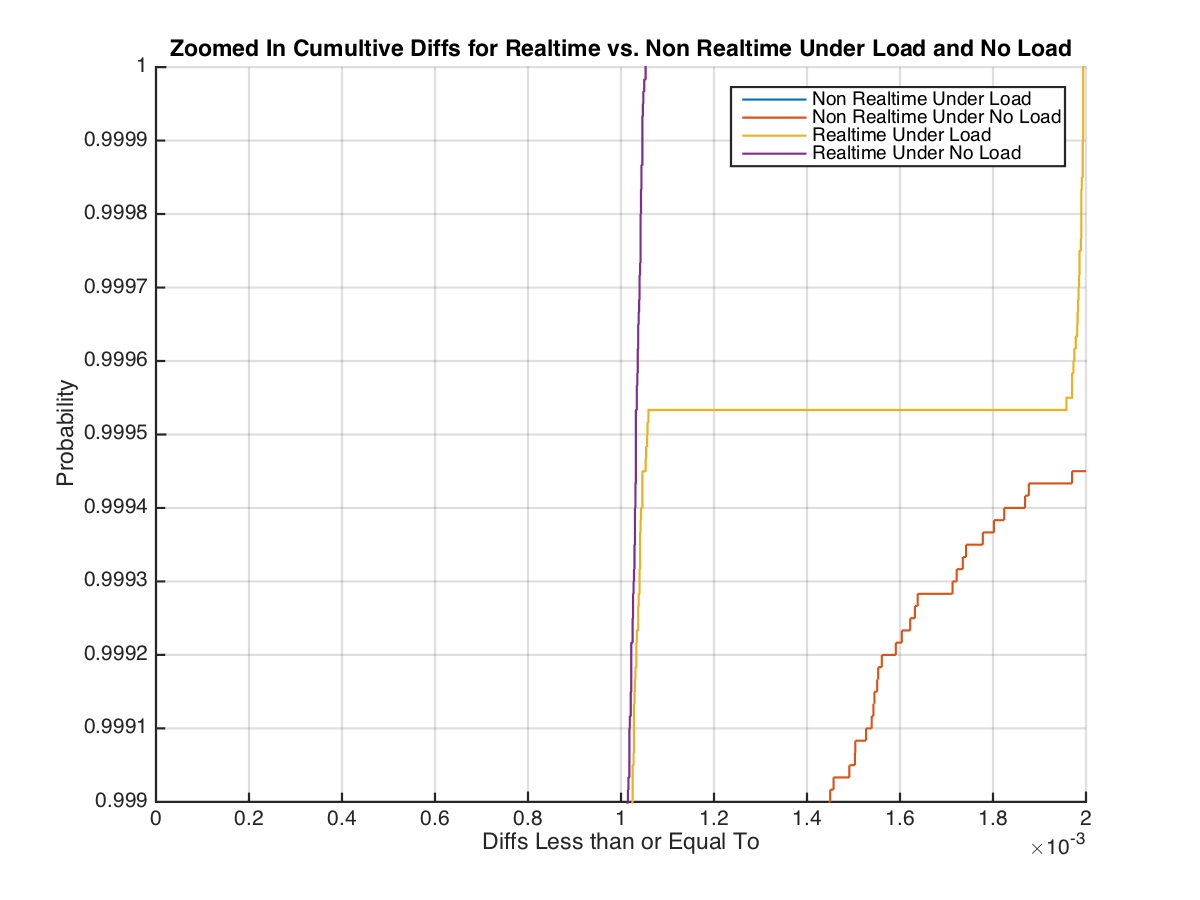


Figure 13: CDF full zoom

Table 1: Diffs less than, equal to, and greater than 0.001s

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Value | Non\_RT\_No\_Load | Non\_RT\_No | RT\_No\_Load | RT\_Load |
| Less than 1 milisecond | 0.002 | 0.002 | 42.77 | 58.635 |
| Exactly 1 milisecond | 0 | 0 | 35.1 | 13.068 |
| Greater than 1 milisecond | 99.998 | 99.998 | 22.13 | 28.297 |
|  |  |  |  |  |

Table 2: Diffs less than, equal to, and greater than 0.001005s

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Value | Non\_RT\_No\_Load | Non\_RT\_No | RT\_No\_Load | RT\_Load |
| Less than 1 milisecond 5 nanoseconds | 0.002 | 0.002 | 94.711 | 88.851 |
| Exactly 1 milisecond 5 nanoseconds | 0 | 0 | 1.97 | 5.624 |
| Greater than 1 milisecond 5 nanoseconds | 99.998 | 99.998 | 3.319 | 5.525 |

Table 3: Diffs less than, equal to, and greater than 0.00101s

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Value | Non\_RT\_No\_Load | Non\_RT\_No | RT\_No\_Load | RT\_Load |
| Less than 1 milisecond 10 nanoseconds | 0.002 | 0.002 | 0 | 0.047 |
| Exactly 1 milisecond 10 nanoseconds | 0 | 0 | 0 | 0 |
| Greater than 1 milisecond 10 nanoseconds | 99.998 | 99.998 | 100 | 99.953 |

Analysis

We make plots comparing load and no load performance for both realtime and userspace code, as well as a CDF plot to see the probabilities of the execution time missing the expected one millisecond frequencies.

We consider the comparison of the relative performance of the real-time kernel and user space.  From figures 11-13 and tables 1-3 we can consider the probability that the real-time kernel and user space codes are running with the expected 1-millisecond difference between execution timestamps.  The execution of the real-time kernel code executes in one millisecond or less ~72% of the time, within 5 nanoseconds of the expected one millisecond ~95% of the time, and within 10 nanoseconds 100% of the time.

When running the code in the real time kernel space, with each execution a timestamp is printed to terminal.  This print to terminal may be a bottleneck causing the execution time to extend past the expected 1 millisecond time.

Nonetheless, the real-time performance of the kernel depends on the deadlines.  We see there is a clear improvement between load and no load, user space and real-time kernel space performance.  Given a deadline of 1 millisecond, 10 nanoseconds, the no load, real-time kernel code would indeed be running in real-time.

With slight modifications to the code, or a faster processer, LinuxCNC indeed appears to have a kernel that makes real-time performance possible. Further work with a computer that has a parallel or serial port would permit exploration of the real-time performance of the HAL, as well as the standard LinuxCNC UI to further verify the performance of the system.

ACKNOWLEDGMENTS

Our thanks to the professor, Dr. Chenyang Lu, and TAs Chong Li, Jing Li for their guidance finding and completing a project.

REFERENCES

1. Bowman, M., Debray, S. K., and Peterson, L. L. 1993. Reasoning about naming systems. *ACM Trans. Program. Lang. Syst.* 15, 5 (Nov. 1993), 795-825. DOI= <http://doi.acm.org/10.1145/161468.16147>.
2. Ding, W. and Marchionini, G. 1997. *A Study on Video Browsing Strategies*. Technical Report. University of Maryland at College Park.
3. Fröhlich, B. and Plate, J. 2000. The cubic mouse: a new device for three-dimensional input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (The Hague, The Netherlands, April 01 - 06, 2000). CHI '00. ACM, New York, NY, 526-531. DOI= <http://doi.acm.org/10.1145/332040.332491>.
4. Tavel, P. 2007. *Modeling and Simulation Design*. AK Peters Ltd., Natick, MA.
5. Sannella, M. J. 1994. *Constraint Satisfaction and Debugging for Interactive User Interfaces*. Doctoral Thesis. UMI Order Number: UMI Order No. GAX95-09398., University of Washington.
6. Forman, G. 2003. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.* 3 (Mar. 2003), 1289-1305.
7. Brown, L. D., Hua, H., and Gao, C. 2003. A widget framework for augmented interaction in SCAPE. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology* (Vancouver, Canada, November 02 - 05, 2003). UIST '03. ACM, New York, NY, 1-10. DOI= <http://doi.acm.org/10.1145/964696.964697>.
8. Yu, Y. T. and Lau, M. F. 2006. A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions. *J. Syst. Softw.* 79, 5 (May. 2006), 577-590. DOI= <http://dx.doi.org/10.1016/j.jss.2005.05.030>.
9. Spector, A. Z. 1989. Achieving application requirements. In *Distributed Systems*, S. Mullender, Ed. ACM Press Frontier Series. ACM, New York, NY, 19-33. DOI= <http://doi.acm.org/10.1145/90417.90738>.

1. Final code setup available at <https://github.com/efinkg/washu520_rts> [↑](#footnote-ref-1)
2. [http://linuxcnc.org/docs/2.7/html/getting-started/getting-linuxcnc.html# download the image](http://linuxcnc.org/docs/2.7/html/getting-started/getting-linuxcnc.html#_download_the_image) [↑](#footnote-ref-2)
3. <https://github.com/dewy721/EMC-2-Arduino> [↑](#footnote-ref-3)