LinuxCNC Real-Time Evaluation

Stephen Decker, Jeremy Tang, Ethan Glassman

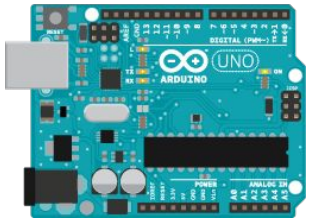
Project Recap

- We intend to evaluate real-time performance of the LinuxCNC operating system environment on a laptop.
- We intend to use an Arduino emulating a CNC mill as a workload.
- In this way we can test the real-time performance of the kernel while performing I/O operations without needing a large, complex, and dangerous real load.

Debian Linux Distribution With Real-Time Kernel

Sends directions to workload

Receives results from workload



Microcontroller

Timeline/Goals

- Pre Planning: Selecting a good data set and simulated load calculation. Complete by 9/25 ✔
- Hardware Selection and Setup: We have not yet selected between the Raspberry Pi and an Arduino. We need to finalize our hardware setup and purchase cabling, install LinuxCNC, etc. Complete by 10/2 ✔
- Initial Implementation: Code simulated load calculation on Raspberry Pi/Arduino. Setup desktop with data set and I/O operations. Complete by 10/9 ✔
- Real-time evaluation setup: Code desktop with software to time evaluation of I/O performance using high accuracy timing APIs (e.g. rdtsc), and begin to collect data on timing parameters. Complete by 10/16 ✔
- Prepare demo 1: Presentation of progress so far. Presentation on 10/21

Linux CNC vs Vanilla Linux

- Real-time kernel exists as a layer between hardware and normal kernel
- Hardware interrupts handled by Real-Time kernel
- Normal kernel sees hardware interrupts as software interrupts
- User services given highest priority, normal kernel services given lowest
- Other kernel config modifications (high resolution timers, no CPU frequency scaling)

Linux CNC vs Vanilla Linux (cont.)

- LinuxCNC also contains Hardware Abstraction Layer (HAL)
- Allows configuration with many host setups without recompiling kernel
- Achieved by virtualizing expected hardware
- Generally, LinuxCNC does not necessarily replace but rather supplements

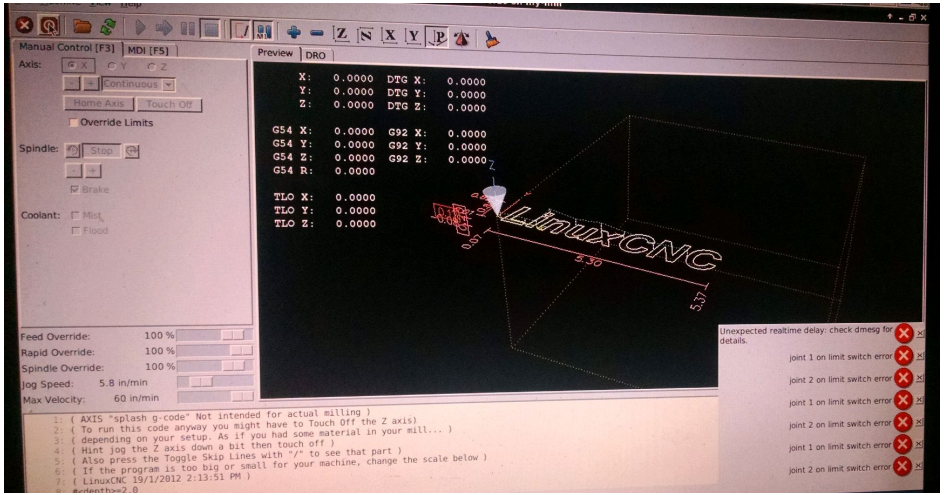
So...what is completed?

- Studied changes from vanilla Linux to narrow our focus on testing the modified portions of Linux CNC
- Picked Arduino over Raspberry Pi
- Located and installed code on Arduino to allow Linux CNC to recognize Arduino as a CNC mill
  - https://github.com/dewy721/EMC-2-Arduino/tree/master/Downloads/HAL2Arduino
- We can connect the Arduino to LinuxCNC as a workload and run GCode machining programs
- Hourglass and cyclictest are installed and initial timing tests have been performed

Problems faced: First Connection

Correctly configuring LinuxCNC to remove unneeded features

- Modified my-mill.ini by commenting out PYVCP = custompanel.xml
- Modified custom_postgui.hal by commenting out sets spindle-at-speed true



- Located and removed all references to limit switches in configuration files
- Located and removed references to homing cycles in configuration files

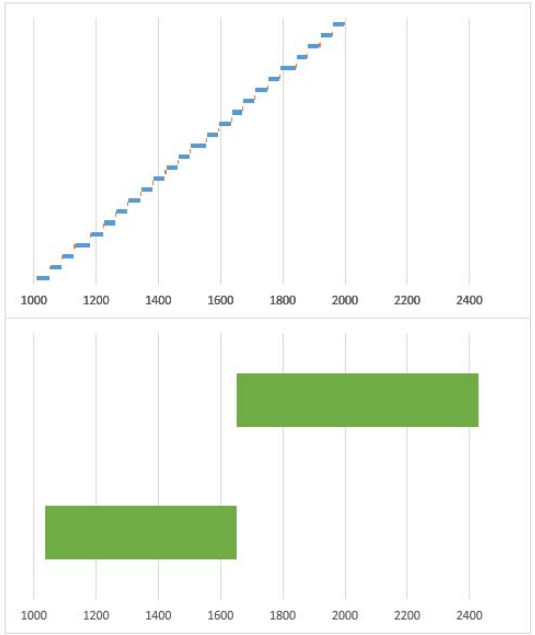
Demo Time!

Hourglass and Cyclictest

- x86 Linux distribution comes with rdtsc(), no modifications needed
- Initial measurements taken of hourglass & cyclictest performance
  - when idle
  - when operating the arduino “mill”
- Strange quirks
  - Hourglass’s “high resolution timer” API is out of date, incompatible with LinuxCNC.
    - Pretty sure hourglass’s standard timing calls invoke LinuxCNC’s modern high resolution timer (nanosecond resolution)

Hourglass data

- `hourglass -n 3 -d 5s -t 0 -p LOW -t 1 -p NORMAL -t 2 -p HIGH`
  - 3 threads running for 5 seconds, one LOW priority, one NORMAL priority, one HIGH priority



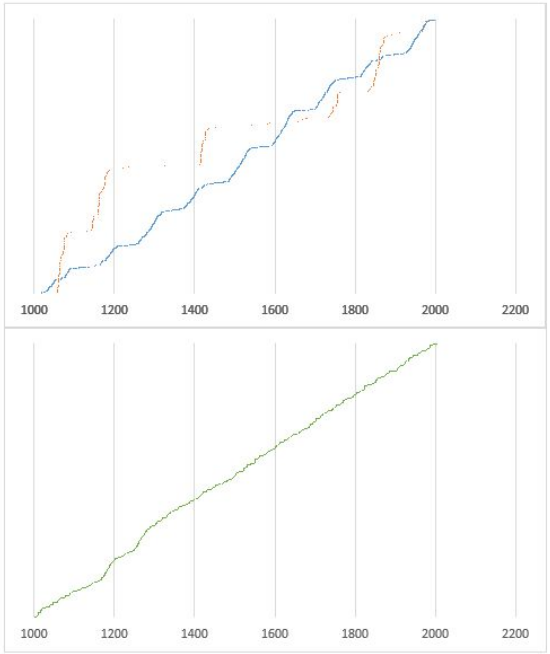
Left: Running when PC is idle

Right: When also running LinuxCNC

Low Priority

Normal Priority

High Priority



Cyclictest data

- Priority 80, 1 thread, 10000us intervals, 10000 loops
- PC is idle
  - ```
    root@cse520linux:/home/test# cyclictest -t1 -p 80 -n -i 10000 -l 10000
    # /dev/cpu_dma_latency set to 0us
    policy: fifo: loadavg: 0.14 0.05 0.06 1/207 5432

    T: 0 ( 5432) P:80 I:10000 C:  10000 Min:  4 Act:   11 Avg:   9 Max:   48
    ```
- PC is running LinuxCNC
  - ```
    root@cse520linux:/home/test# cyclictest -t1 -p 80 -n -i 10000 -l 10000
    # /dev/cpu_dma_latency set to 0us
    policy: fifo: loadavg: 0.63 0.31 0.16 1/220 5549

    T: 0 ( 5548) P:80 I:10000 C:  10000 Min:  6 Act:   20 Avg:   60 Max:  138
    ```

Hourglass and Cyclictest: Next Steps

- Confirm Hourglass's use of high resolution timers
- Additional and useful analyses and testing workloads
- Dealing with the dual-core nature of the laptop (accommodate? disable a core?)
- Better data visualizations
- Additional synthetic benchmarks

Timeline/Goals (cont.)

- Real-time evaluation: Continue to collect data and refine desktop code. Perform ~1 million tests to determine if schedule is ever missed. If schedule is missed, determine a best guess of what is causing the scheduling to be missed and during what scenarios this might occur. Complete by 11/6
- Prepare demo 2: Presentation of progress so far. Presentation on 11/9 or 11/11
- Finalize real-time evaluation: Complete any lingering tests, write up results. Complete by 11/27
- Prepare final demo: Presentation of progress throughout class. Presentation on 11/30

Stretch Objectives

- Real-time performance comparison of different interfaces (Ethernet, USB)
- Measure scheduling latency of the Linux environment that LinuxCNC uses (Hourglass, Cyclictest, hooking into LinuxCNC/arduino, other benchmarks)
- Determining specific circumstances when LinuxCNC does not maintain real time scheduling
- Create recommended steps for how LinuxCNC can be modified to maintain real time performance

Questions?