Camera Control Nodes

Camera Control Workshop
22 January, 2008
LSST

Kevan Hashemi
Brandeis University

PDF Version
Account of ELS Work

Contents

System Architecture

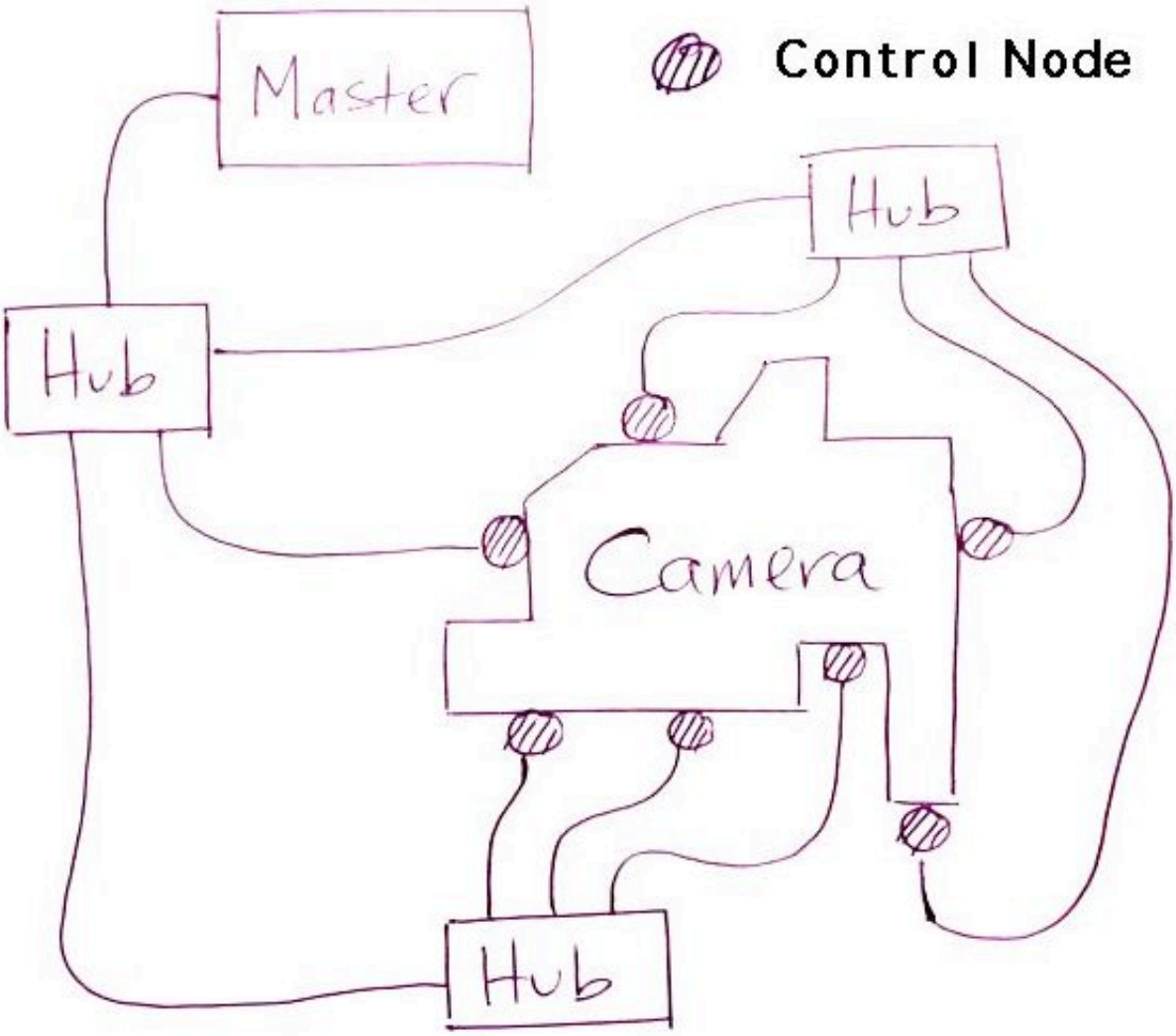

Figure: The Camera Control System.

Master controls nodes over CAT-5 Ethernet.

Node Architecture

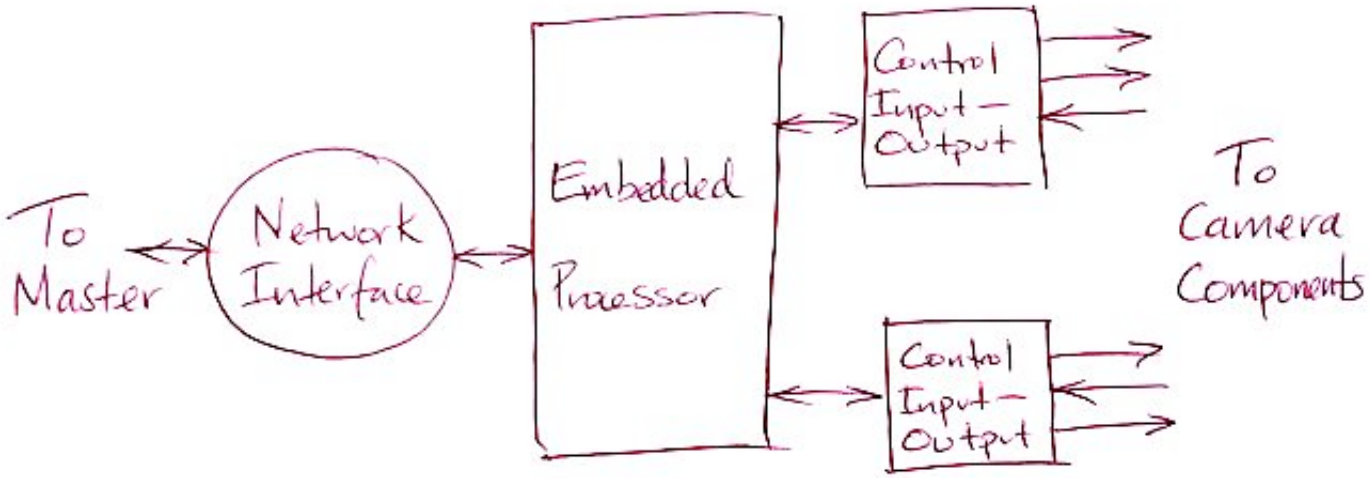

Figure: A Control Node.

Control Input-Output Types:

- Servo Motor Drive
- Stepper Motor Drive
- TTL Switching Output
- Analog Temperature Input
- LVDS Communication

For Timing and Control Module (TCM):

- 50 MHz Data Clock Distribution
- 50 MBPS Serial Transmission
- 50 MBPS Serial Reception
- Transmit 32-MByte images

Control Node Questions

Will one form of embedded processor suit all control nodes?

Requires Answers To:

- What is the maximum data rate from a node?
- What is the maximum data rate to a node?
- What is the quickest response required of a node?
- How important is dynamic upgrade of node software?
- How much engineering effort can we dedicate to development?
- When do we need the first control nodes for tests?

Proposed Solution

The CCS group proposes:

- Off-the-shelf ix86 processor in PC104 format
- Off-the-shelf input-output boards in PC104 format
- Custom input-output boards where necessary
- Linux operating system

We tried:

- From WinSystems
- PCM-SC520 embedded computer, i486 133 MHz.
- PCM-UIO48A 48-line digital I/O card
- Embedded Linux System based upon Simply MEPIS
- Total price $1500 with flash card and power supply.

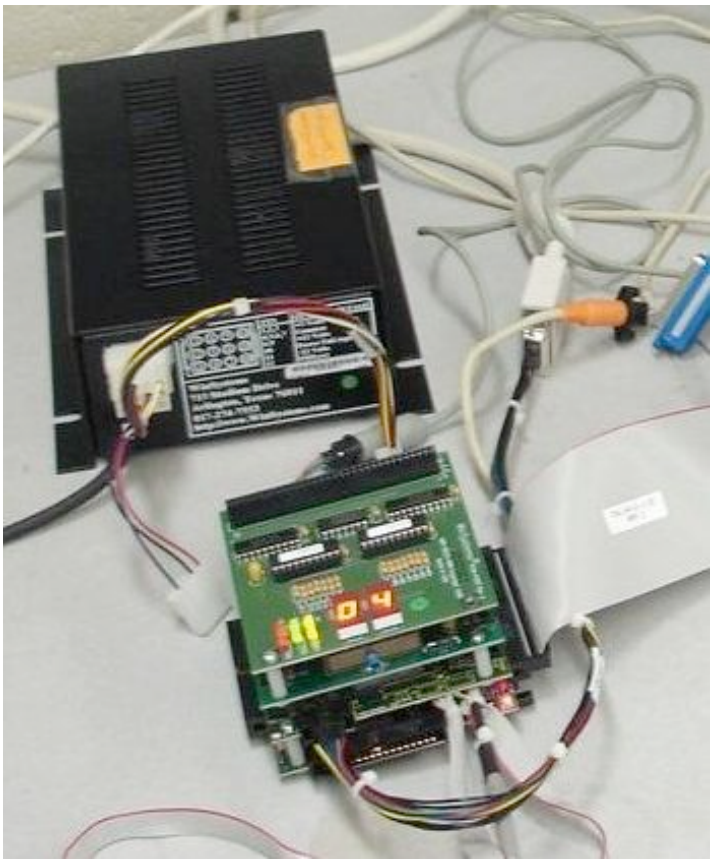

Figure: Our PC104 Embedded Linux System.

ELS with Toolchain

Standard way to build an ELS disk image is with a toolchain on a non-ELS computer.

Toolchain does the following:

- operates on a non-ELS machine
- compiles ELS
- compiles device drivers
- compiles control code
- creates bootable ELS disk image

Toolchain can work by:

- using chroot to change apparant root directory
- using Linux source code modified and curtailed to decrease size
- using GCC with abbreviated libraries
- using GCC compiler options to cross-compile
- altering uname implementation to change apparant processor type

Program ELS by:

- create ELS disk image with toolchain
- copy disk image to flash card
- put flash card in ELS computer

In Theory: Toolchain can reside on any machine that can host the GCC cross-compiler.

In Practice: We cannot compile device drivers on Scientific Linux for our Simply-MEPIS ELS.

ELS without Toolchain

Without a toolchain, we build the ELS on the ELS machine itself.
Use stripped-down C-compiler: GCC takes gigabytes.
Compile kernel, drivers, and control programs on machine that will run them.
Compiler must use RAM-disk or magnetid disk to avoid over-use of flash memory.
Create new disk image, re-write flash card, re-boot.

But: Can't find any examples of this being done.

ELS Distribution Proposal

We can guarantee effective distribution of an ELS in the following way:

- Create new disk partition on desktop Linux machine.
- Load ELS on partition.
- The toolchain is an instance of the ELS on the desktop.
- Compile ELS, device drivers, and example control programs in toolchain.
- Compress ELS and distribute.
- User de-compresses and places in directory on non-ELS Linux computer.
- User compiles control programs with non-ELS version of GCC.
- Copy files to flash card.
- Make flash card bootable with utility like grub.
- Plug into computer and run.
- Modify user programs, compile on non-ELS computer, upload by ftp to ELS computer.

Note that device drivers must be created by the toolchain, not on user's machine.
User could duplicate toolchain to create new device drivers.

Our Control Node

Use device driver for UIO48A compiled by WinSystems.
Use ELS disk image compiled by WinSystems.
Compile control programs on desktop computer, copy to flash.
512 MByte flash, 256 MByte RAM, i486 133 MHz processor.
Runs HTTP, FTP, Telnet.
48-line digital IO, 4 RS-232, 10/100 Ethernet, VGA driver.

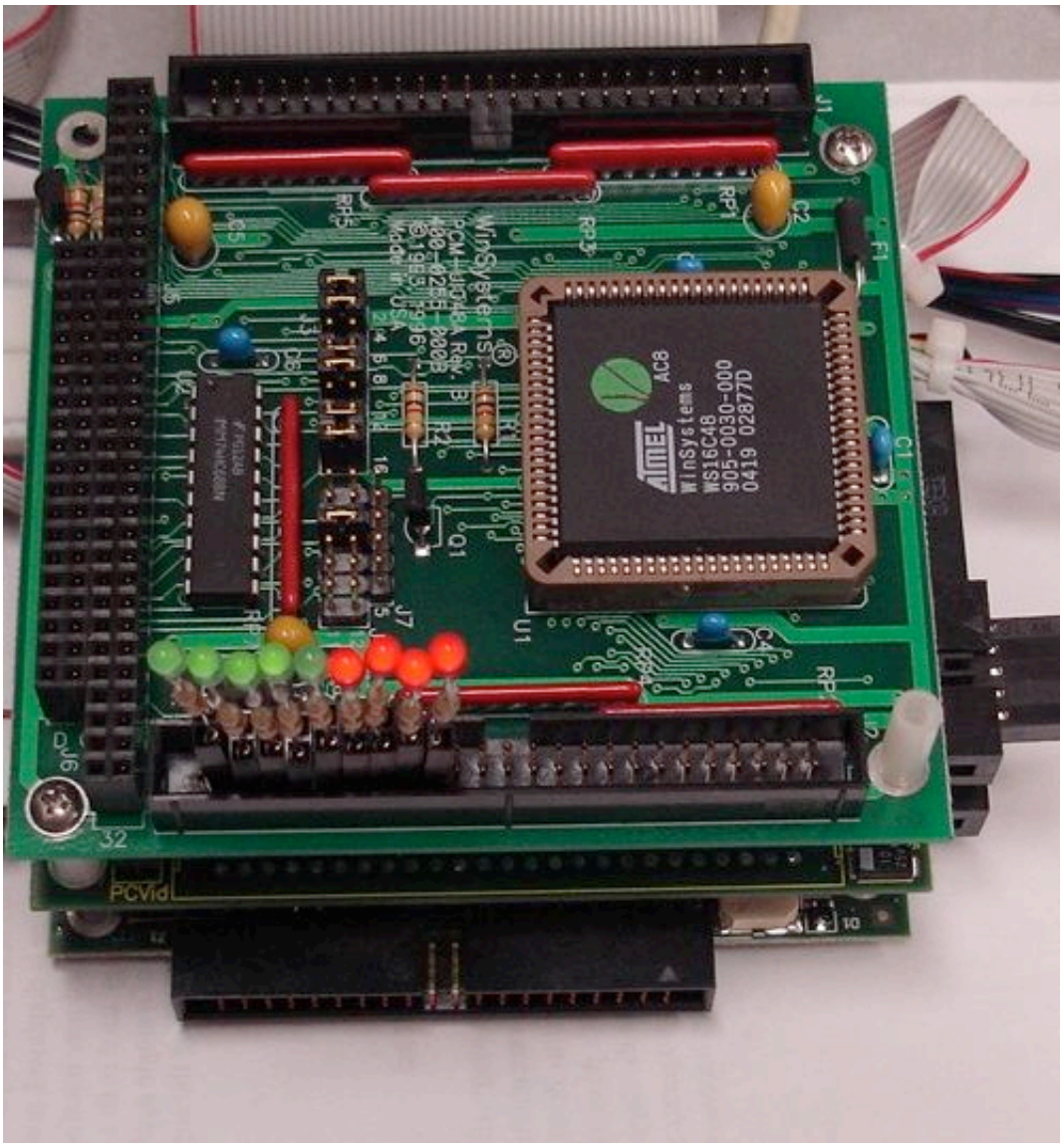

Figure: Our Input-Output Board.

We compile example code and flash lights.
Unpack to flash lights labor: 5 engineering days.

Performance of Example Node

Fastest bit rate through Ethernet: 4 MBytes/s.

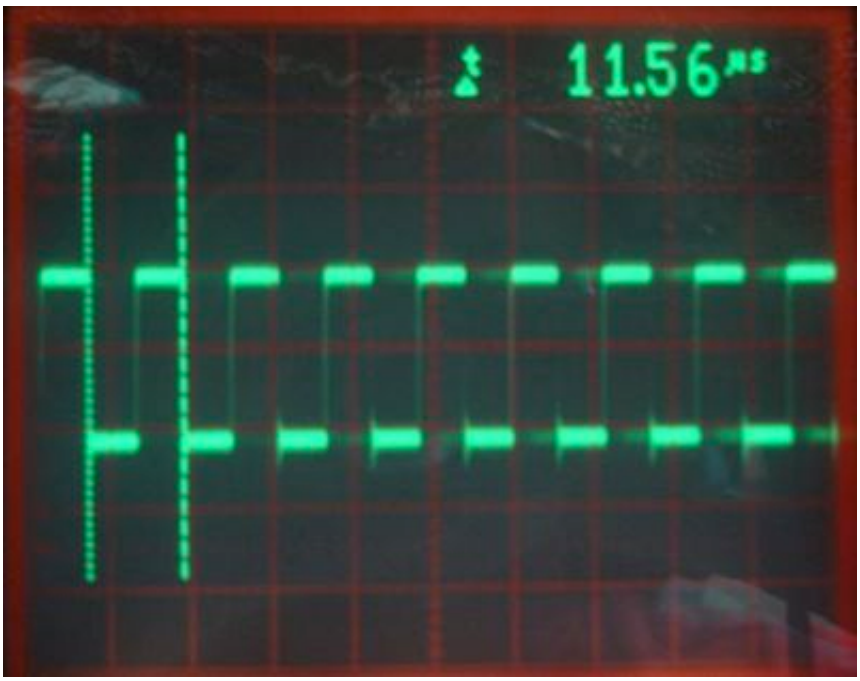

Figure: Minimum Output Update Time.

Compile code that toggles one output as fast as it can.

- Fastest bit rate through IO board: 180 kbits/s.
- Fastest data clock rate through IO board: 90 kHz.
- With byte-wide IO, max transfer: 90 kBytes/s.
- Takes 6 min to transfer 32 MByte image through TCM.
- Update time is ≈6 µs.

Update Time

6 μs × 133 MHz = 800 clocks ≈ 200 instructions

- Program calls a library routine
- Library routine calls the device driver
- Device driver writes to device node
- Hardware receives update

Example block move in assembler, 21 clocks per cycle:

```
inner_loop_sr:
  ioe ld a,(hl)
  ld (de),a
  inc de
  dec b
jr nz,inner_loop_sr
```

Our ELS code is 40 × slower than it would be in assembler.
Our 133-MHz i486 acts like a 3 MHz Z80.

Reduce update time by:

- Use –O3 optimization to compile all code: ×2 faster
- Direct access to IO devices: ×2 faster
- Block moves in assembler: ×10 faster

Alternative Processor

Brandeis Unviersity uses RabbitCore for TCPIP Data Acquisition.

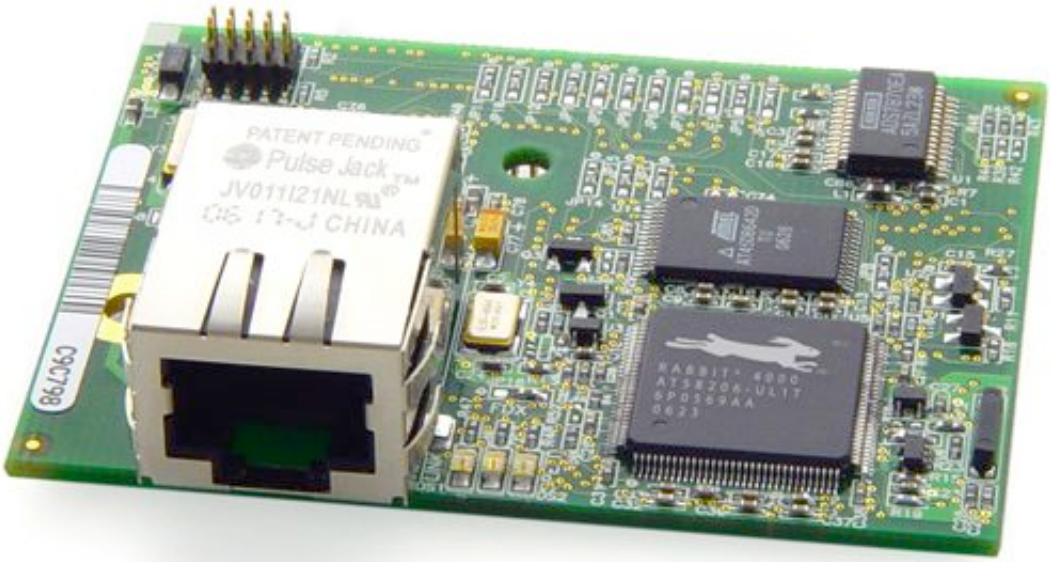

Figure: RCM4200 RabbitCore Embedded Processor.

The new RCM4200 features:

- 60 MHz 8-bit processor
- 10/100 Ethernet
- 8-channel, 12-bit ADC
- Byte-Wide IO Ports
- Synchronized PWM Channels
- Real-Time Clock
- $100 in single quantity
- TCPIP stack, HTTP, FTP, and Flash file system
- Unpack to flash lights labor: 1 engineering hr

Ethernet: 500 kBytes/s, Byte-wise IO: 3 MBytes/s
Dynamic re-programming possibe but not yet tested at Brandeis.

Conclusion

Embedded Linux System offers:

- dynamic re-programming
- multiple-user access that won't crash
- fast Ethernet transfer rate for shorter network monopoly

But it has the following problems:

- Toolchain and distribution method require development.
- Expensive, complex, and large compared to smaller TCPIP modules.
- Without optimised device drivers, digital IO is slower thann smaller TCPIP modules.

We cannot guarantee Embedded Linux System development by end Summer 2008.

We can guarantee delivery of RabbitCore control nodes by end Summer 2008.