Session #2: The Target Board and Class Software

Making it real

Installation Steps

- Install ARM9 tool chain and sample code from class CD
- Install the latest Firefox to run Blackboard
- Configure minicom
- Configure Host Networking
- Connect the Target
- Run sample program

Installing Class Files

- Insert CD ROM and mount (should automount)
- cd to your home directory
- Become root user – su
  - Password:
- Execute /media/EmbeddedLinux/install_tools.sh or
- <your_mount_point>/install_tools.sh <your_mount_point>
- Exit root user shell
- Execute /media/EmbeddedLinux/install.sh or
- <your_mount_point>/install.sh <your_mount_point>
- Add /usr/local/arm/4.3.2/bin to your PATH (Edit .bash_profile)
  - PATH=$PATH:$HOME/bin:/usr/local/arm/4.3.2/bin

What got installed

- /usr/local/arm/4.3.2
  - Complete ARM9 cross tool chain
- /usr/local
  - eclipse – Eclipse IDE
- /usr/src/arm
  - linux – link to…
  - linux-3.6-FA – kernel source tree
- /home
  - target_fs – link to $HOME/target_fs-2451

In your home directory

- busybox-1.20.2-p1 – Busybox source tree
- images – original target board images
- target_fs-2451 – root file system for target board
  - home/include – header files for sample code
  - home/src – sample source code

Running Blackboard on Linux

- Install Firefox 3.x or later if you don't have it
  - www.mozilla.com
  - Untar in /usr/local
- Install Java Runtime Environment (JRE) if necessary
  - http://java.com
  - Untar in /usr/local
  - Create link in /usr/local/firefox/plugins to /usr/local/jre1.6.0_11/plugin/i386/ns7/libjavaplugin_oji.so
  - Change link java in /usr/bin to point to /usr/local/jre1.6.0_11/bin/java

Running Blackboard on Linux 2

- Start firefox
  - Navigate to UCSD Blackboard and log in
  - Camtasia Studio complains that Flash Player is not installed. Follow the link to download.
- Install Flash Player
  - Untar download and copy it to the firefox plugins directory
  - On mine it's /usr/mozilla/plugins
- Restart firefox

Configure minicom

- As root user, run minicom -s
- Select *Serial port setup*
  - Serial Device (/dev/ttyS0)
  - (/dev/ttyUSB0 for USB to serial converter)
  - Bps/Par/Bits (115200 8N1)
  - No flow control
- Select *Screen and keyboard*
  - Type “b” to change backspace behavior to DEL

Configure minicom II

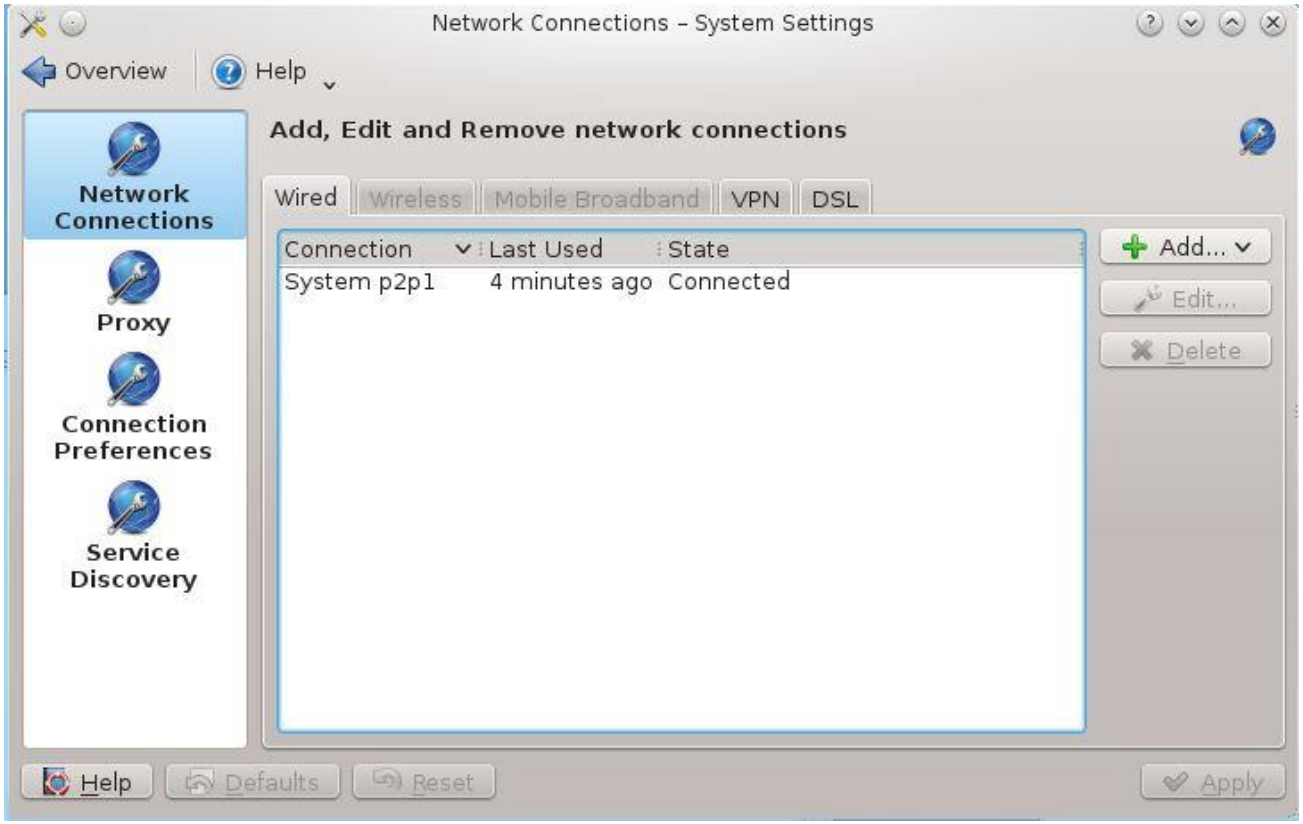
- Select *Modem and dialing*
  - Remove Init string
  - Remove Reset string
- Select *Save setup as dfl*
- Exit minicom

Add your user to dialout group

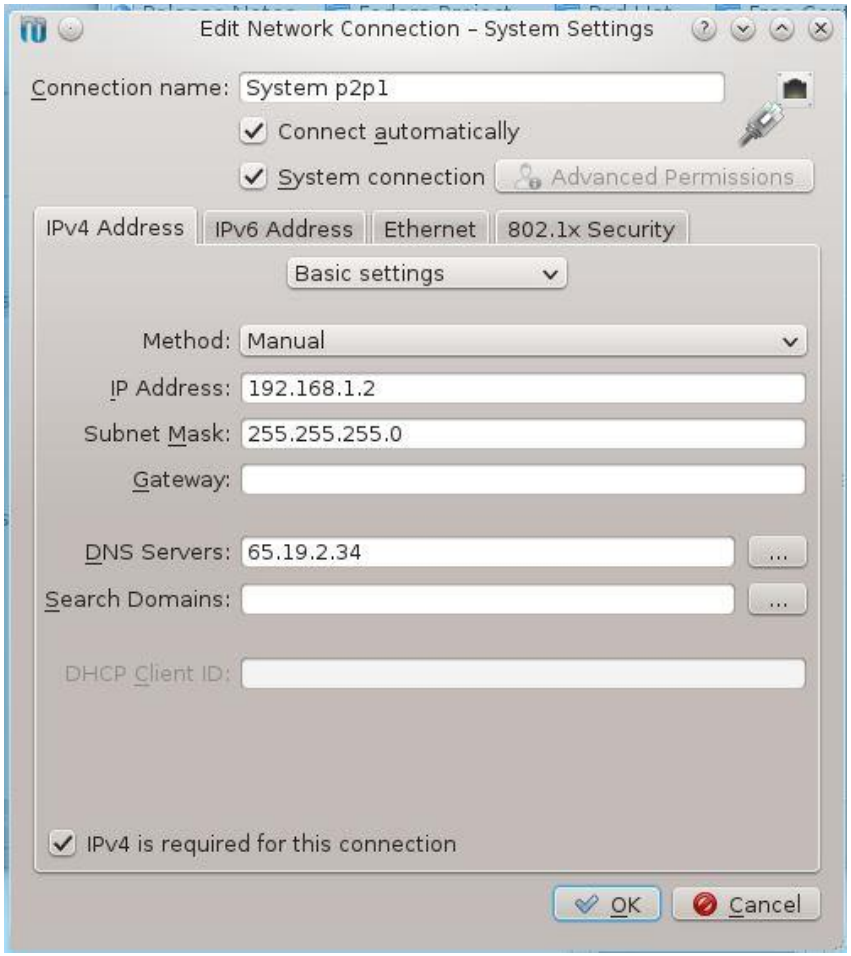
- As root user edit /etc/group
  - Add your user name to the line that begins "dialout"
  - Save file
- Use graphical dialog
  - Administration > Users and Groups
  - Select your user and click Properties
  - Click the Groups tab and check dialout
  - Exit dialog

Configure Host Networking

Settings > System Settings > Network and Connectivity > Network Settings



Configure Host Networking 2



Configure Host Networking 3

- Set Gateway and DNS Servers to match your network
- Click OK
- Ignore “KDE Wallet Service”

Configure NFS

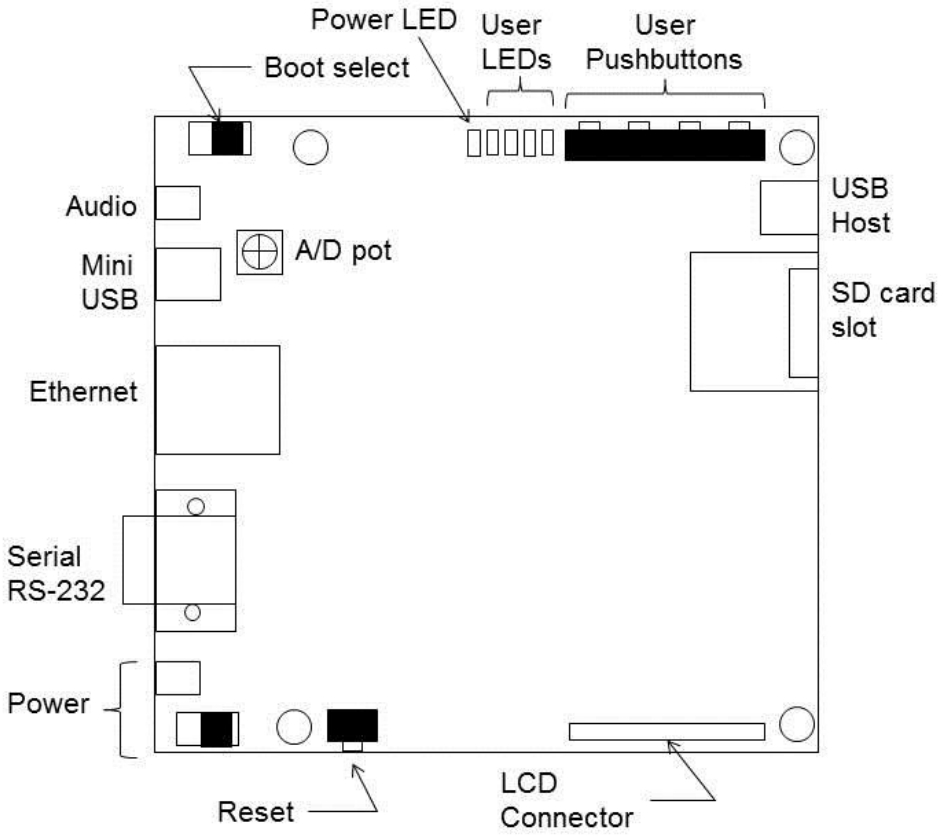
- Edit /etc/exports (must be root)
  - Change <your_user_name> to your user name
  - Save
- In shell window as root:
  - systemctl enable nfs-server.service
  - systemctl start nfs-server.service
- If NFS not there:
  - yum install nfs-utils

The Target Board

Specifications

- 400 MHz ARM9 processor (Samsung S3C2451)
- 128 MB of SDRAM
- 256 MB of NAND flash
- SD/MMC socket
- 3.5 inch color LCD
- 10/100 Ethernet port
- USB 2.0 host and device ports
- RS-232 serial port for debugging
- Multi-channel A/D converter
- User-programmable LEDs and switches

Layout



Connect and Power up the Target

- Connect the serial and network cables
- Run minicom –w
  - -w = wrap long lines
- Plug in the power supply
- Target boots into Linux
- Try some shell commands

Flash memory and filesystems

- NOR flash
  - Fast read, reliable
  - Random access, good for code execution
- NAND flash
  - Higher density, lower cost
  - Faster write and erase times
  - Longer re-write life expectancy
  - Not random access, treat as file system
  - At boot prompt execute mtdparts

YAFFS

- Yet Another Flash File System
  - Designed specifically for NAND flash
  - Wear leveling – distribute writes evenly across the device
  - Detect bad blocks – move data around them
  - Journaling and error correction for reliability

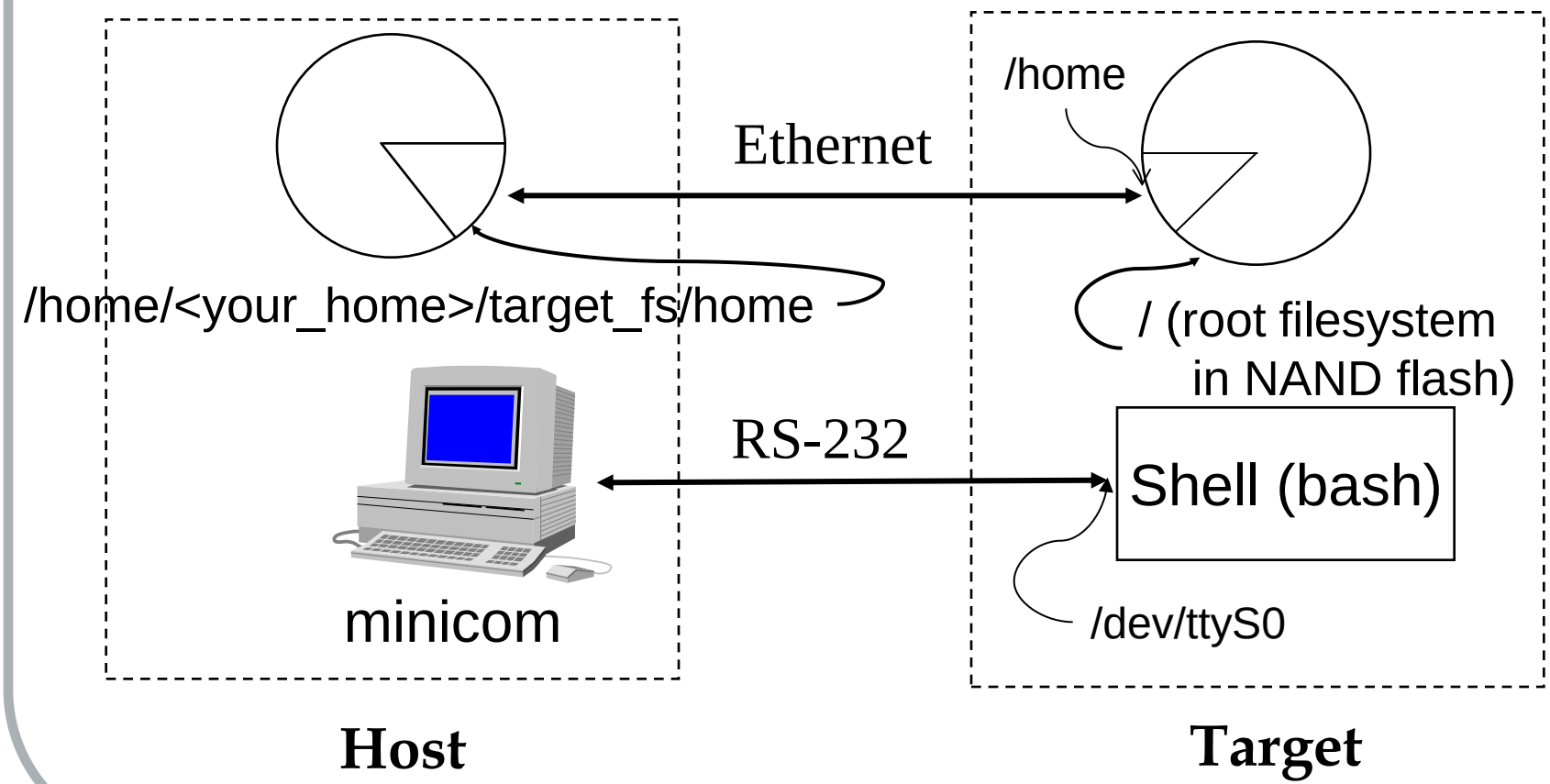
The target Linux environment

- ls /home
  - This is where our sample code is stored
- List the /proc directory
  - cat /proc/interrupts
- ls –l bin
  - BusyBox!
- ifconfig
  - IP address was set by kernel command line

Our First Program

- On the workstation (from your home directory)
  - cd target_fs/home/src/led
  - make
- On the target
  - cd /home/src/led
  - ./led

What's Going On Here



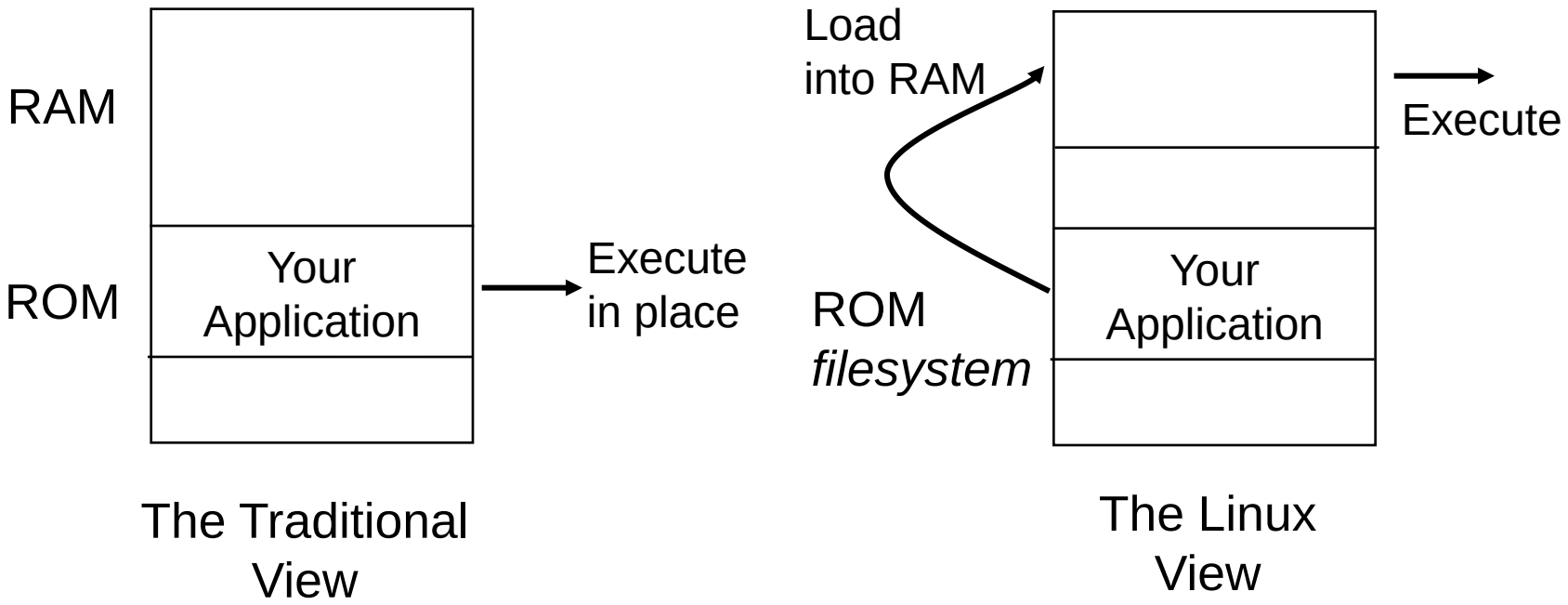
Memory Map


Internal SRAM (64 KB)
0
10000
Ethernet Controller
DM9000
20000000
30000000
DRAM (128 MB)
38000000
Internal
Peripherals
48000000
5c000000

Accessing hardware from Linux

- Open led.c
- Peripherals share address space with memory
- Must “map” peripheral space into user space process
  - open (“/dev/mem”);
  - mmap ();

Two Views of Embedded Programming



Review I

- Install class software
  - Cross build tools
  - Eclipse
  - Kernel source tree
  - Sample code
- Configure Workstation
  - Configure minicom
  - Extend PATH

Review II

- Configure Workstation (cont)
  - Networking
    - Fixed IP address
    - NFS Server
    - Export directory
- Target
  - Fixed IP address specified on kernel command line
  - Kernel booted from flash
  - NFS-mounted /home directory