CSE 237A

Introduction to Embedded Systems

Tajana Simunic Rosing
Department of Computer Science and Engineering
UCSD

Welcome to CSE 237A!

- Instructor:
  - Tajana Simunic Rosing
    - Email: tajana-at-ucsd.edu; put CSE237a in subject line
    - Ph. 858 534-4868
  - Office Hours:
    - TW 1-2pm, CSE 2118
- **Admin:**
  - **Sheila Manalo**
    - Email: shmanalo@ucsd.edu
    - Phone: (858) 534-8873
    - Office: CSE 2272
- Class Website:
  - http://www.cse.ucsd.edu/classes/fa08/cse237a/
- Grades, announcements and discussion board:
  - http://webct.ucsd.edu

About This Course

- Part of a three course group
  - CSE 237A: Introduction to Embedded Systems
  - CSE 237B: Software for Embedded Systems
  - CSE 237C: Validation and Prototyping of ES
- Related course
  - ECE 284: Wireless Embedded and Networked Systems – mainly sensor nets
- Depth sequence:
  - Embedded Systems and Software

Course Objectives

- Develop an understanding of the technologies behind the embedded computing systems
  - technology capabilities and limitations of the hardware, software components
  - methods to evaluate design tradeoffs between different technology choices.
  - design methodologies
- Overview of a few hot research topics in ES
- For more details, see the schedule on the webpage

Course Requirements

- No official graduate course as prerequisite, but, many assumptions!
- Knowledge
  - Digital hardware, basic electrical stuff, computer architecture (ISA, organization), programming & systems programming, algorithms
- Skills
  - Advanced ability to program
  - Ability to look up references and track down pubs (Xplore etc)
  - Ability to communicate your ideas (demos, reports)
- Initiative
  - Open-ended problems with no single answer requiring thinking and research
- Interest
  - Have strong interest in research in this or related fields

Course Grading

- Homework (3-4): 10%
- Embedded systems project 40%
  - Install OS onto an embedded platform. implement an energy efficient media player and make kernel more energy efficient
- Final exam: 45%
- Class participation, attendance, engagement: 5%
  - Come prepared to discuss the assigned paper(s)

Reader & Textbooks

- No textbook
- A set of papers will be required reading
  - will relate to the core topic of that class
  - you are expected to read it BEFORE the class
- In addition I will give pointers to papers and web resources

Reference books

- Peter Marwedel, “Embedded Systems Design,” Kluwer, 2004.
- “Embedded, Everywhere: A Research Agenda for Networked Systems of Embedded Computers,” National Research Council. http://www.nap.edu/books/0309075688/html/
- John A. Stankovic and Kirthi Ramamritham, "Hard Real-Time Systems," IEEE Computer Society Press.
- G.D. Micheli, W. Wolf, R. Ernst, “Readings in Hardware/Software Co-Design,” Morgan Kaufman.
- S.A. Edwards, “Languages for Digital Embedded Systems,” Kluwer, 2000.
- R. Melhem and R. Graybill, “Power Aware Computing,” Plenum, 2002.
- M. Pedram and J. Rabaey, “Power Aware Design Methodologies,” Kluwer, 2002.
- Bruce Douglass, "Real-Time UML - Developing Efficient Objects for Embedded Systems," Addison-Wesley, 1998.
- Hermann Kopetz, "Real-Time Systems : Design Principles for Distributed Embedded Applications," Kluwer, 1997.
- Hassan Gomaa, "Software Design Methods for Concurrent and Real-Time Systems," Addison-Wesley, 1993.
- P. Lapsley, J. Bier, A. Shoham, and E.A. Lee, “DSP Processor Fundamentals: Architectures and Features,” Berkeley Design technology Inc,, 2001.
- R. Gupta, "Co-synthesis of Hardware & Software for Embedded Systems," Kluwer, 1995.
- Felice Balarin, Massimiliano Chiodo, and Paolo Giusto, "Hardware-Software Co-Design of Embedded Systems : The Polis Approach," Kluwer, 1997.
- Jean J. Labrosse, "Embedded Systems Building Blocks : Complete And Ready To Use Modules In C ," R&D Publishing, 1995.
- Jean J. Labrosse, "uC / OS : The Real Time Kernel," R&D Publishing, 1992.

Embedded Systems on the Web

- Berkeley Design technology, Inc.: http://www.bdti.com
- EE Times Magazine: http://www.eet.com/
- Linux Devices: http://www.linuxdevices.com
- Embedded Linux Journal: http://embedded.linuxjournal.com
- Embedded.com: http://www.embedded.com/
  - *Embedded Systems Programming* magazine
- Circuit Cellar: http://www.circuitcellar.com/
- Electronic Design Magazine: http://www.planetee.com/ed/
- Electronic Engineering Magazine: http://www2.computeroemonline.com/magazine.html
- Integrated System Design Magazine: http://www.isdmag.com/
- Sensors Magazine: http://www.sensorsmag.com
- Embedded Systems Tutorial: http://www.learn-c.com/
- Collections of embedded systems resources
  - http://www.ece.utexas.edu/~bevans/courses/ee382c/resources/
  - http://www.ece.utexas.edu/~bevans/courses/realtime/resources.html
- Newsgroups
  - comp.arch.embedded, comp.cad.cadence, comp.cad.synthesis, comp.dsp, comp.realtime, comp.software-eng, comp.speech, and sci.electronics.cad

Embedded Systems Courses

- Alberto Sangiovanni-Vincentelli @ Berkeley
  - EE 249: Design of Embedded Systems: Models, Validation, and Synthesis
    - http://www-cad.eecs.berkeley.edu/~polis/class/index.html
- Brian Evans @ U.T. Austin
  - EE382C-9 Embedded Software Systems
    - http://www.ece.utexas.edu/~bevans/courses/ee382c/index.html
- Edward Lee @ Berkeley
  - EE290N: Specification and Modeling of Reactive Real-Time Systems
    - http://ptolemy.eecs.berkeley.edu/~eal/ee290n/index.html
- Mani Srivastava @ UCLA
  - EE202A: Embedded and Real Time Systems
    - http://nesl.ee.ucla.edu/courses/ee202a/2003f/
- Bruce R. Land @ CMU
  - EE476: Designing with Microcontrollers
    - http://instruct1.cit.cornell.edu/courses/ee476

Conferences and Journals

- Conferences & Workshops
  - ACM/IEE DAC
  - IEEE ICCAD
  - IEEE RTSS
  - ACM ISLPED
  - IEEE CODES+ISSS
  - CASES
  - Many others…
- Journals & Magazines
  - ACM Transactions on Design Automation of Electronic Systems
  - ACM Transactions on Embedded Computing Systems
  - IEEE Transactions on Computer-Aided Design
  - IEEE Transactions on VLSI Design
  - IEEE Design and Test of Computers
  - IEEE Transactions on Computers
  - Journal of Computer and Software Engineering
  - Journal on Embedded Systems

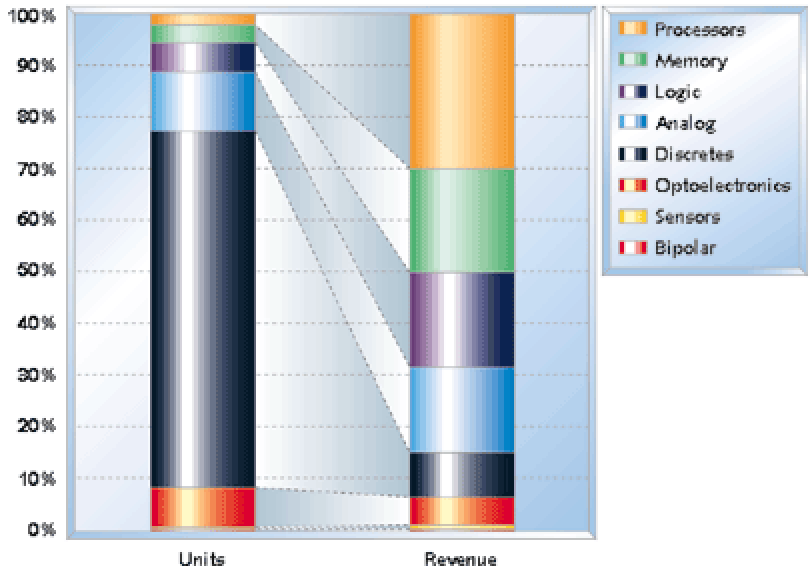
What are embedded systems and why should we care?

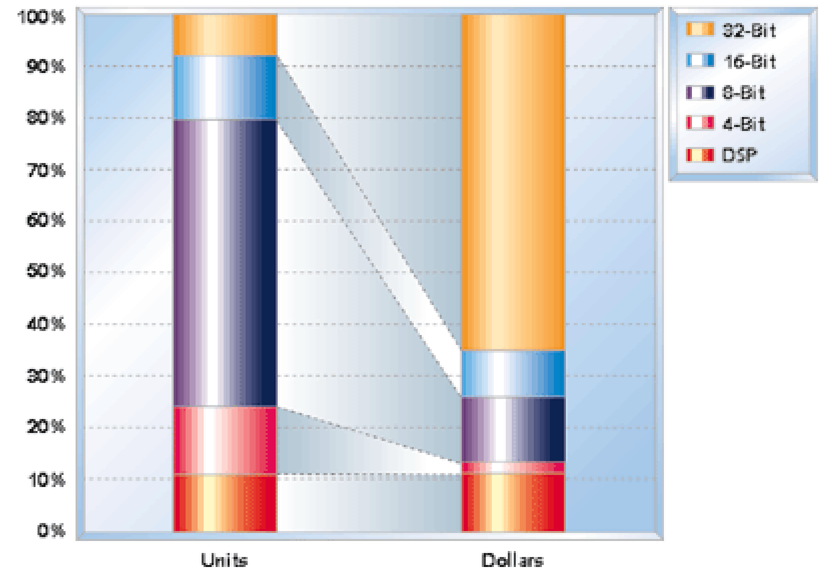
What are embedded systems?

- Systems which use computation to perform a *specific function*
- embedded within a larger device and environment
- Heterogeneous & reactive to environment

Main reason for buying is **not** information processing

Embedded processor market





- Processors strongly affect SW development – keeps their prices high
- Only 2% of processors drive PCs!
- ARM sells 3x more CPUs then Intel sells Pentiums
- 79% of all high-end processors are used in embedded systems

Source: EETimes

Tied to advances in semiconductors

- A typical chip in near future
  - 50 square millimeters
  - 50 million transistors
  - 1-10 GHz, 100-1000 MOP/sq mm, 10-100 MIPS/mW
- Cost is almost independent of functionallity
  - 10,000 units/wafer, 20K wafers/month
  - $5 per part
  - Processor, MEMS, Networking, Wireless, Memory
    - But it takes $20M to build one today, going to $50+M
- So there is a strong incentive to port your application, system, box to the "chip"

Source: RG UCSD

Trends in Embedded Systems

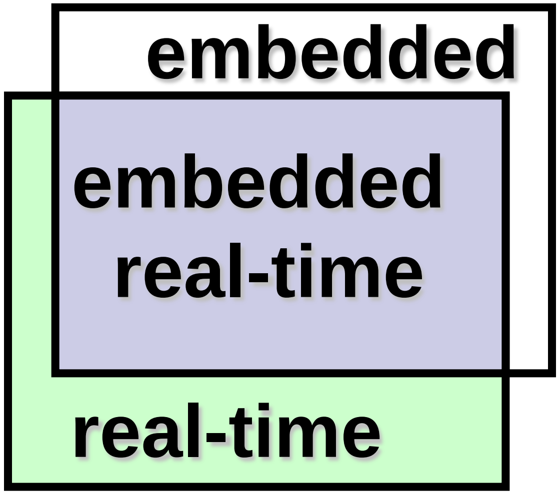
- Increasing code size
  - average code size: 16-64KB in 1992, 64K-512KB in 1996
  - migration from hand (assembly) coding to high-level languages
- Reuse of hardware and software components
  - processors (micro-controllers, DSPs)
  - software components (drivers)
- Increasing integration and system complexity
  - integration of RF, DSP, network interfaces
  - 32-bit processors, IO processors (I2O)

Structured design and composition methods are essential.

Source: RG UCSD

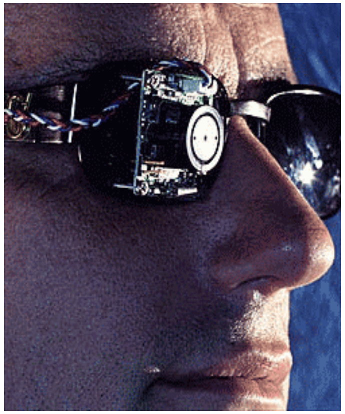
Characteristics of Embedded Systems

- Application specific
- Efficient
  - energy, code size, run-time, weight, cost
- Dependable
  - Reliability, maintainability, availability, safety, security
- Real-time constraints
  - Soft vs. hard
- Reactive - connected to physical environment
  - sensors & actuators
- Hybrid
  - Analog and digital
- Distributed
  - Composability, scalability, dependability
- Dedicated user interfaces



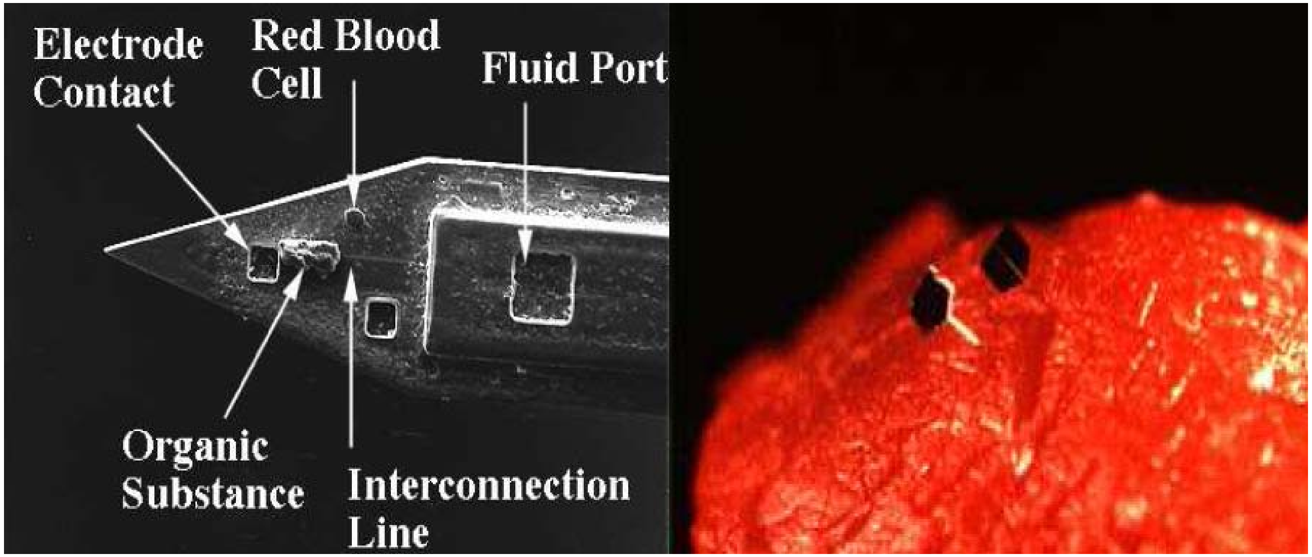
Applications

- Medical systems
  e.g. “artificial eye”



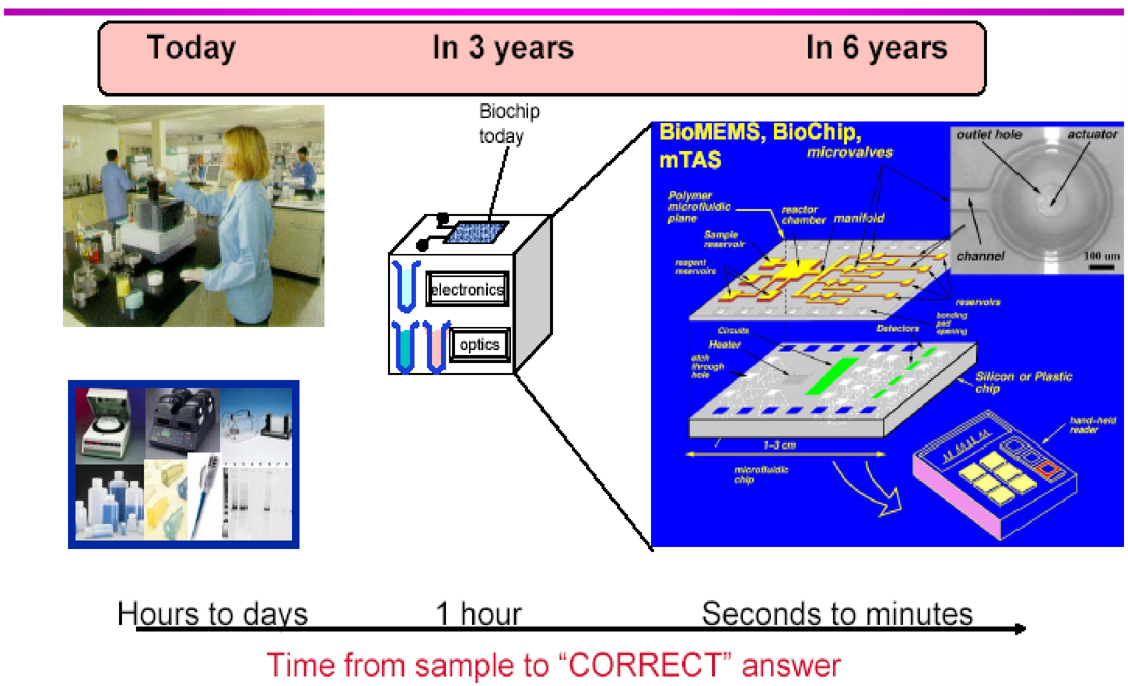
www.dobelle.com

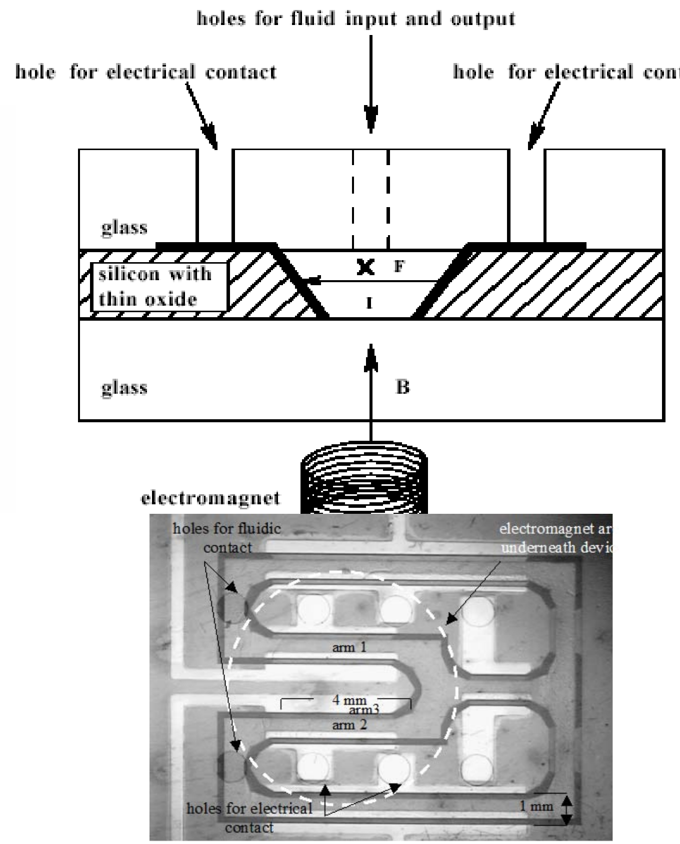
- e.g. “micro-needles”



Source: ASV UCB

On-chip Chemistry





Abraham P. Lee, Ph.D.

Pedometer

- Obvious computer work:
  - Count steps
  - Keep time
  - Averages
  - etc.
- Hard computer work:
  - Actually identify when a step is taken
  - Sensor feels motion of device, not of user feet

If you want to play

- Lego mindstorms robotics kit
  - Standard controller
    - 8-bit processor
    - 64 kB of memory
  - Electronics to interface to motors and sensors
- Good way to learn embedded systems

Mobile phones

- Multiprocessor
  - 8-bit/32-bit for UI
  - DSP for signals
  - 32-bit in IR port
  - 32-bit in Bluetooth
- 8-100 MB of memory
- All custom chips
- Power consumption & battery life depends on software

Inside the PC

- Custom processors
  - Graphics, sound
- 32-bit processors
  - IR, Bluetooth
  - Network, WLAN
  - Hard disk
  - RAID controllers
- 8-bit processors
  - USB
  - Keyboard, mouse

Mobile base station

- Massive signal processing
  - Several processing tasks per connected mobile phone
- Based on DSPs
  - Standard or custom
  - 100s of processors

Telecom Switch

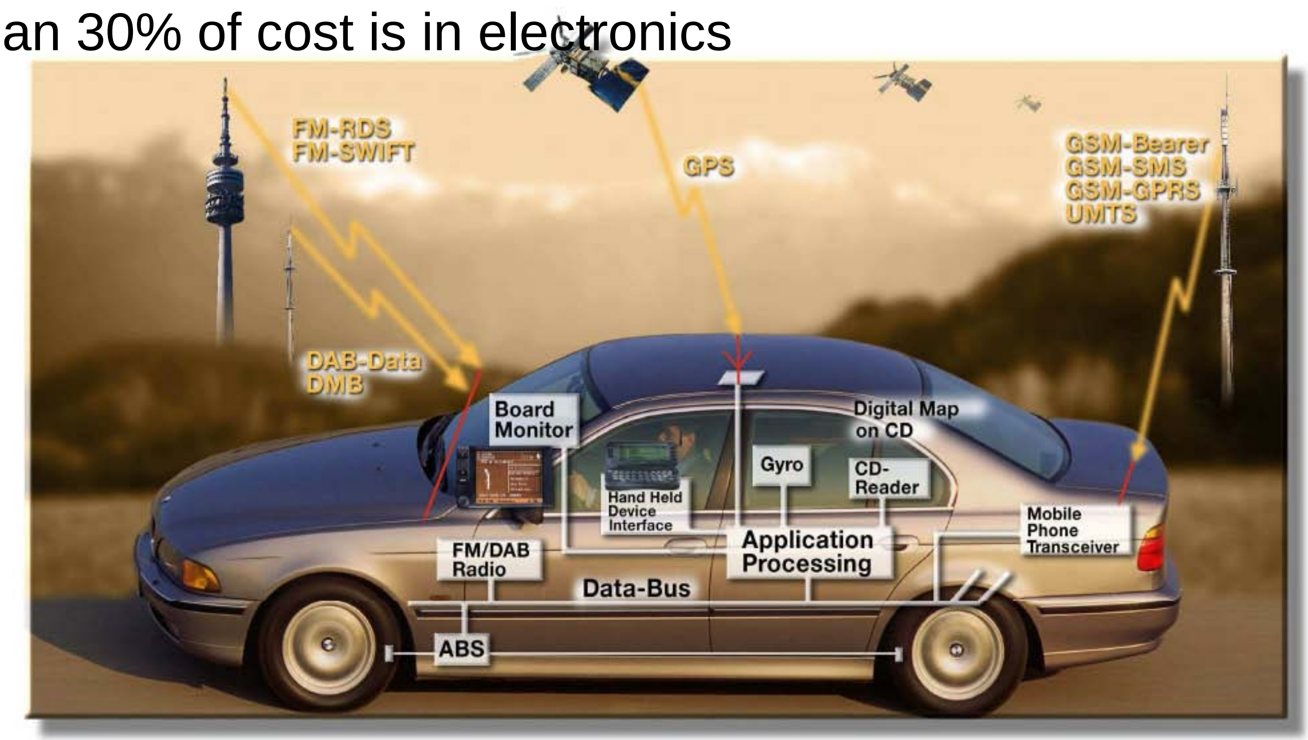
- Rack-based
  - Control cards
  - IO cards
  - DSP cards
  - ...
- Optical & copper connections
- Digital & analog signals

Smart Welding Machine

- Electronics control voltage & speed of wire feed
- Adjusts to operator
  - kHz sample rate
  - 1000s of decisions/second
- Perfect weld even for quite clumsy operators
- Easier-to-use product, but no obvious computer

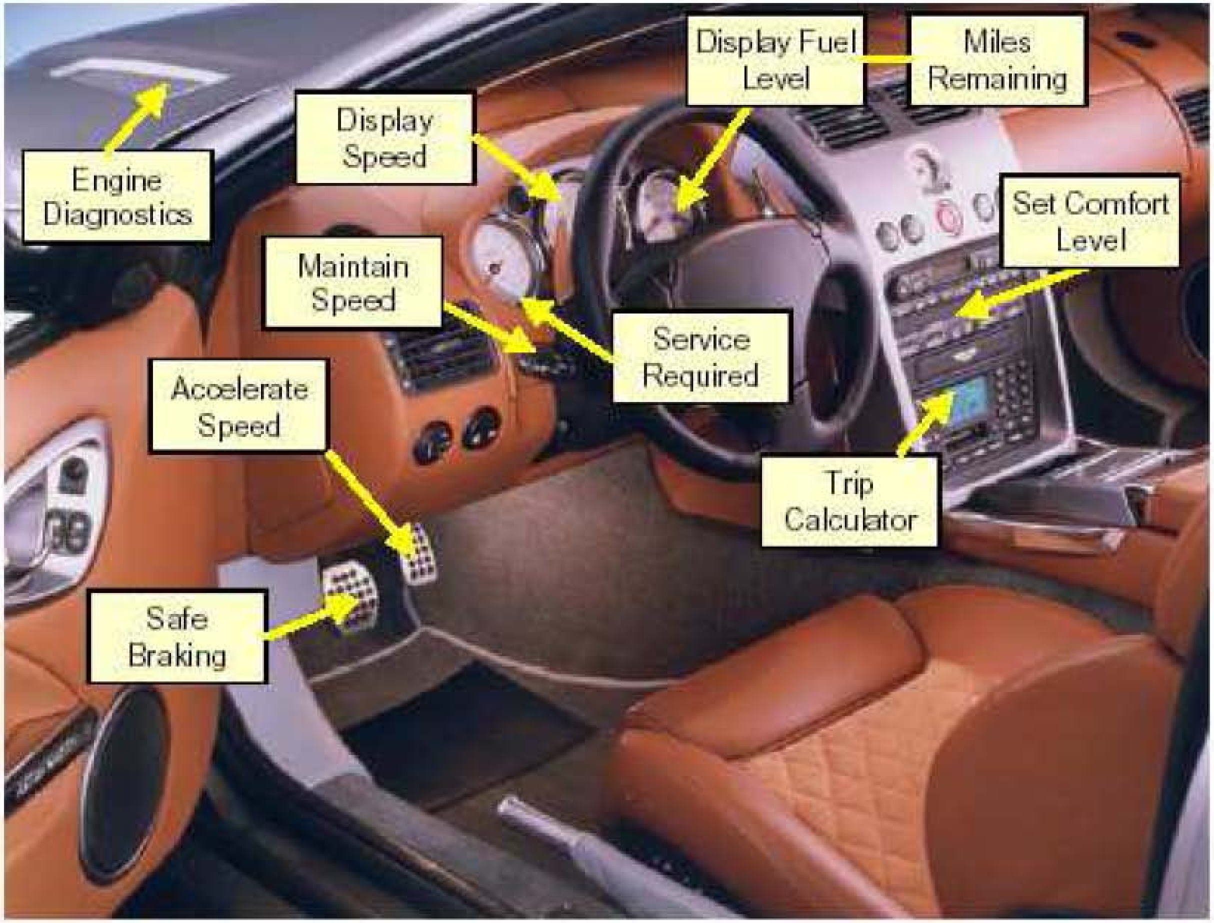
Cars

- Multiple processors networked together (~100), wide variety of CPUs:
  - 8-bit – door locks, lights, etc; 16-bit – most functions; 32-bit – engine control, airbags
- Multiple networks
  - Body, engine, telematics, media, safety
- 90% of all innovations based on electronic systems
- More than 30% of cost is in electronics



FUNCTION OF CONTROLS

Typical minivan application

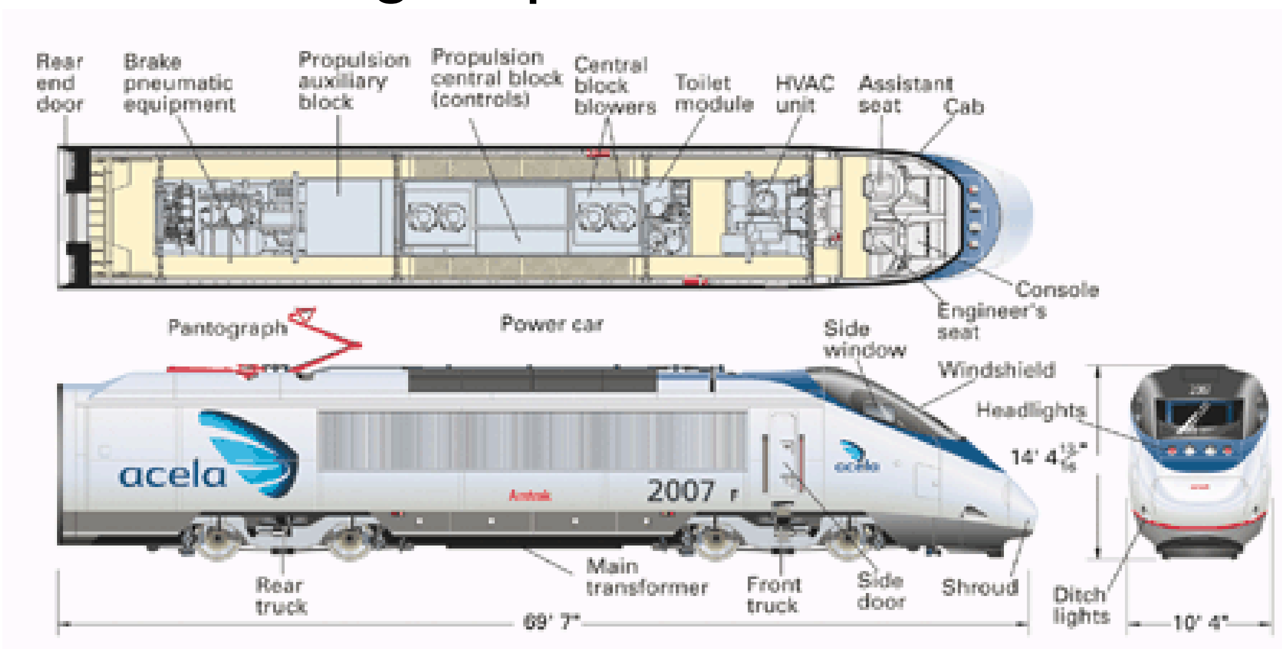


- Configure
- Sense
- Actuate
- Regulate
- Display
- Trend
- Diagnose
- Predict
- Archive

Source: ASV UCB

Transportation

Amtrak Acela High Speed Train



- High speed tilting train service between Boston, New York, and Washington, D.C.
- Built by Bombardier, uses FT-10 free topology twisted pair channel to monitor and control propulsion, power inverters, braking, fire protection systems, ride stability, safety, and comfort.

Source: Echelon

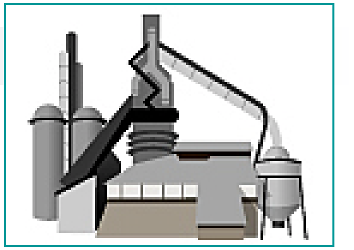
Building Automation

Coeur Défense, Paris

- Location and access
  - The biggest office property complex in Europe located at the heart of the central esplanade of the Paris-La Défense business district
- The building
  - Property complex with a total floor area of 182,000 m² in two towers 180 metres high (39 floors) and 3 small (8-floors) buildings linked to each other by a "glass cathedral".
- Building Automation System
  - 15000 embedded control devices
  - One (1) *i.*LON™ 100 per floor (150 floors) for routing data

Source: Echelon

Process Control



Bellagio Hotel, Las Vegas NV

- Water fountain show
- Fountain and sprinkler systems controls
- Pump controls
- Valve controls
- Choreographed lights and music
- Leak detection

Source: Echelon

Embedded system metrics

- Some metrics:
  - *performance*: MIPS, reads/sec etc.
  - *power*: Watts
  - *cost*: Dollars
    - Nonrecurring engineering cost, manufacturing cost
  - *size*: bytes, # components, physical space occupied
  - Flexibility, Time-to-prototype, time-to-market
  - Maintainability, correctness, safety
- MIPS, Watts and cost are related
  - technology driven
  - to get more MIPS for fewer Watts
    - look at the sources of power consumption
    - use power management and voltage scaling

Source: MS HPL

Example: PDA design

Why did they design it this way?

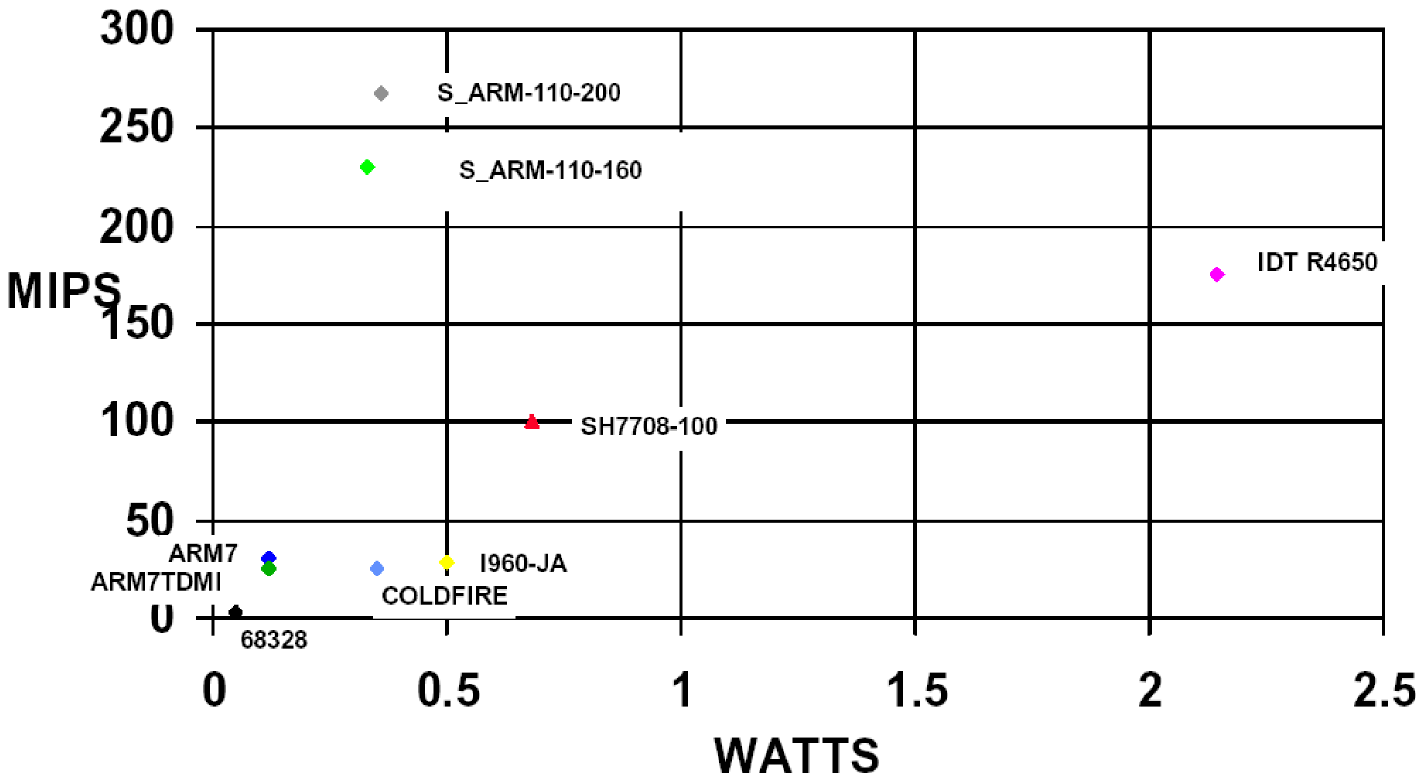
A 'Dragonball*' processor?

We all wanted StrongARMs

*The Dragonball used in the early Palm Pilots is a Motorola 68328

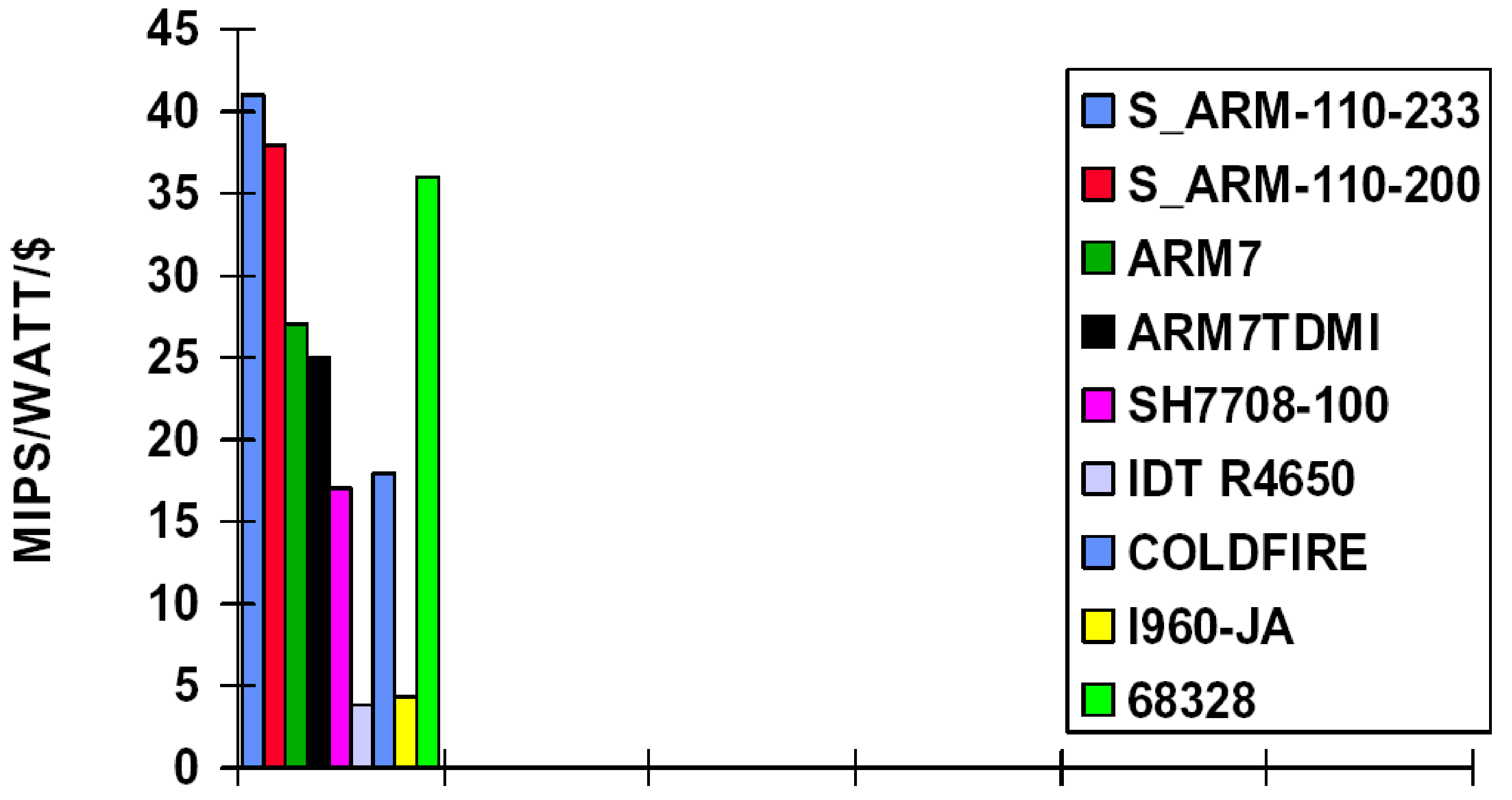
Source: MS HPL

MIPS vs. Watts



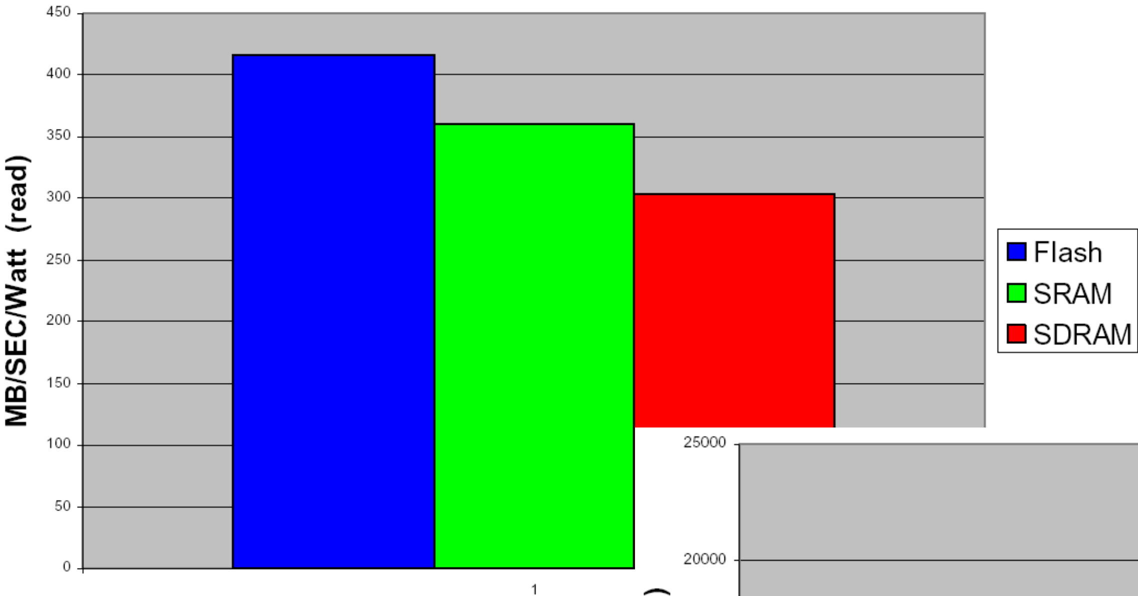
Source: MS HPL

MIPS/W/$

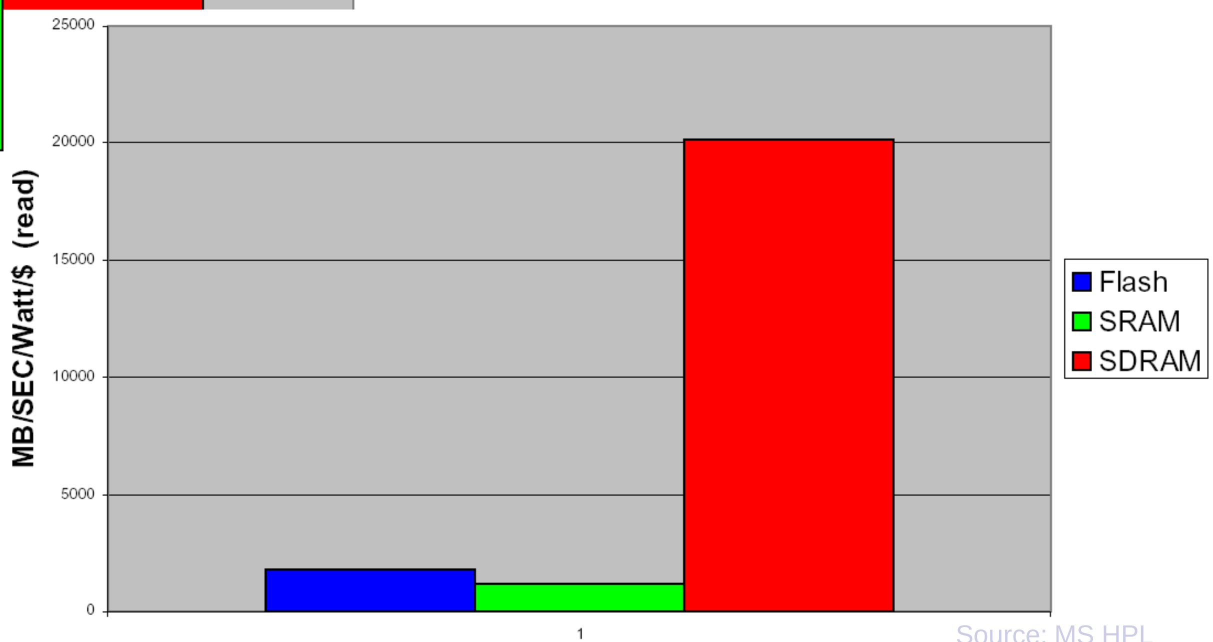


Source: MS HPL

Bandwidth vs. Watt and $

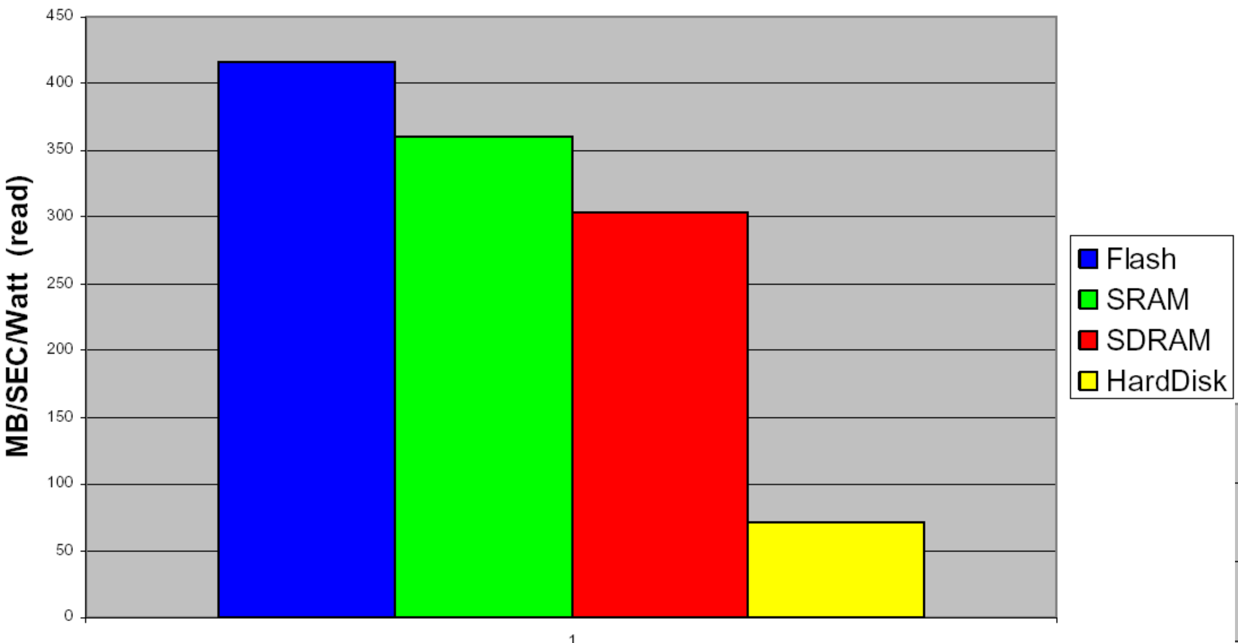


This is why PDAs use SDRAM

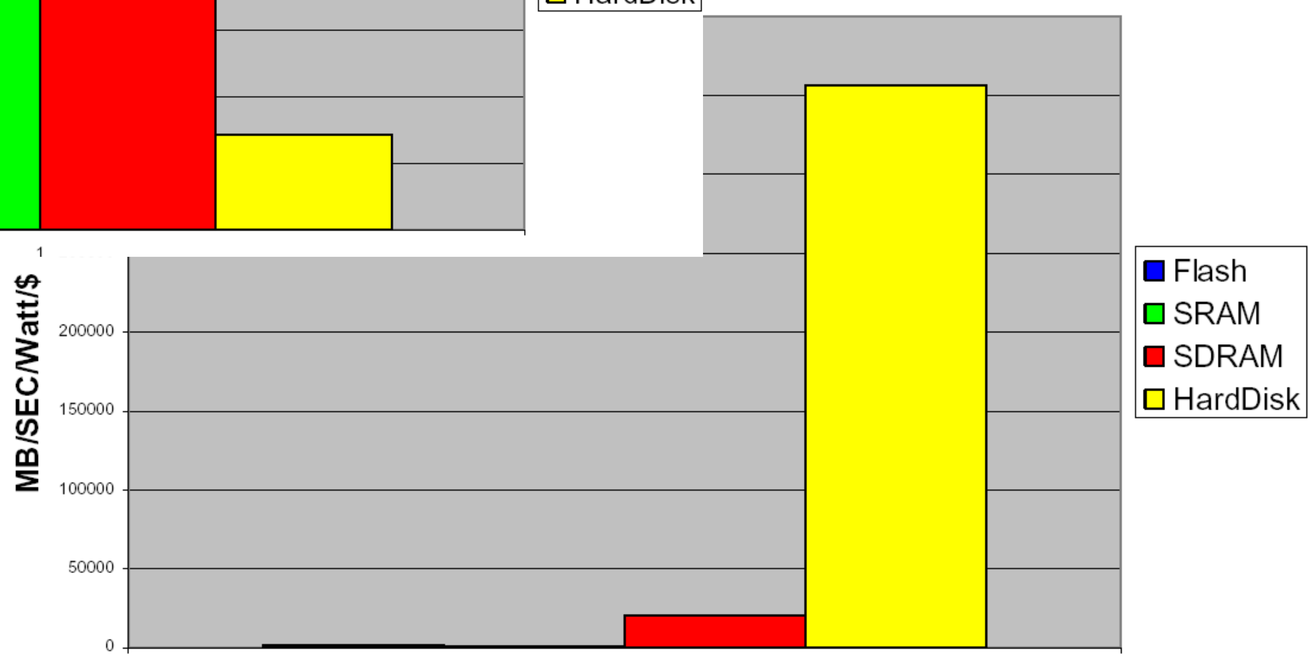


Source: MS HPL

BW/W/$ with hard disk

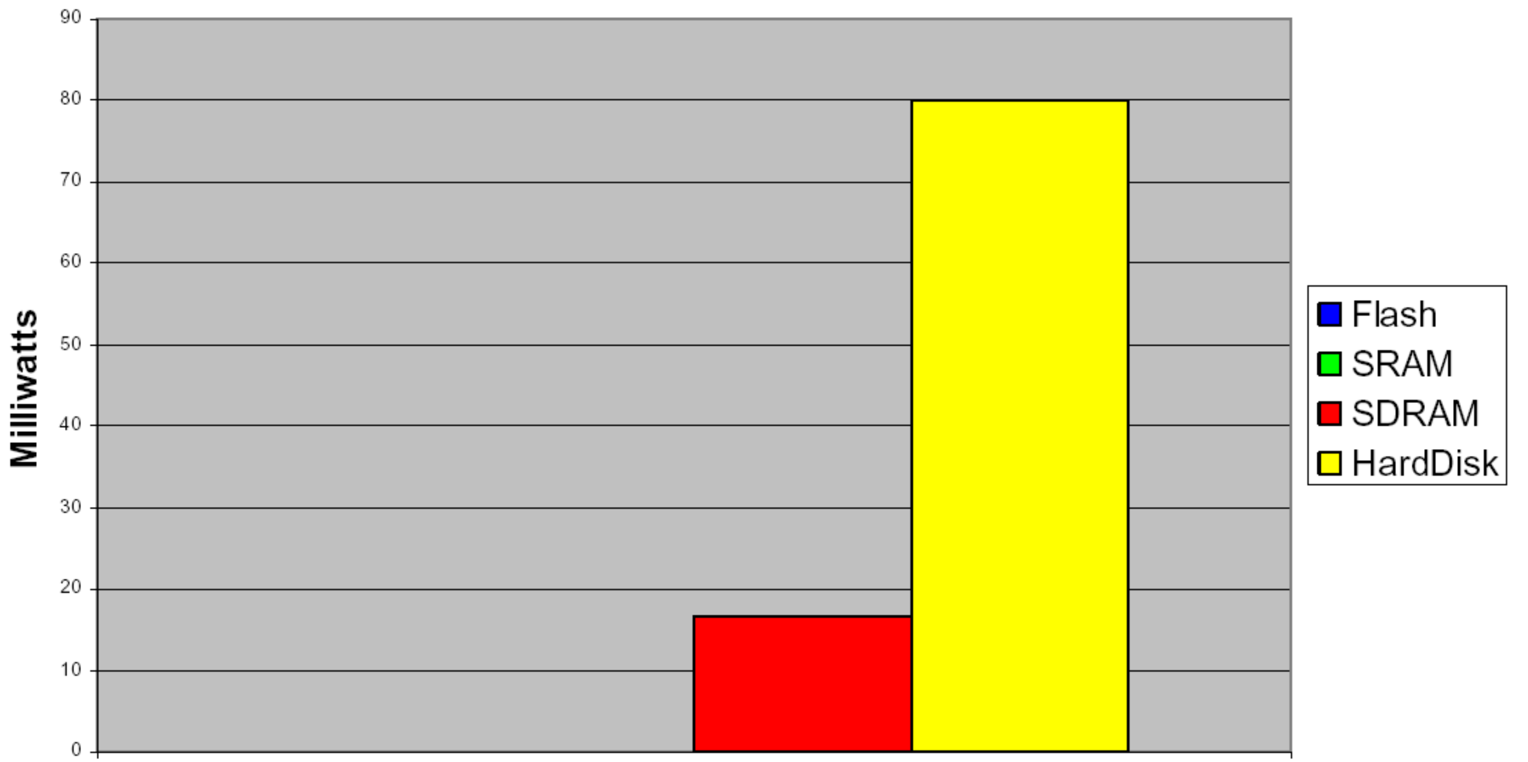


Why IPOD used hard disk



Source: MS HPL

Standby power



Here is why cell phone battery lasts longest, PDA shorter and IPOD only a few hours

Source: MS HPL

CSE237a Project: Energy efficient multimedia player

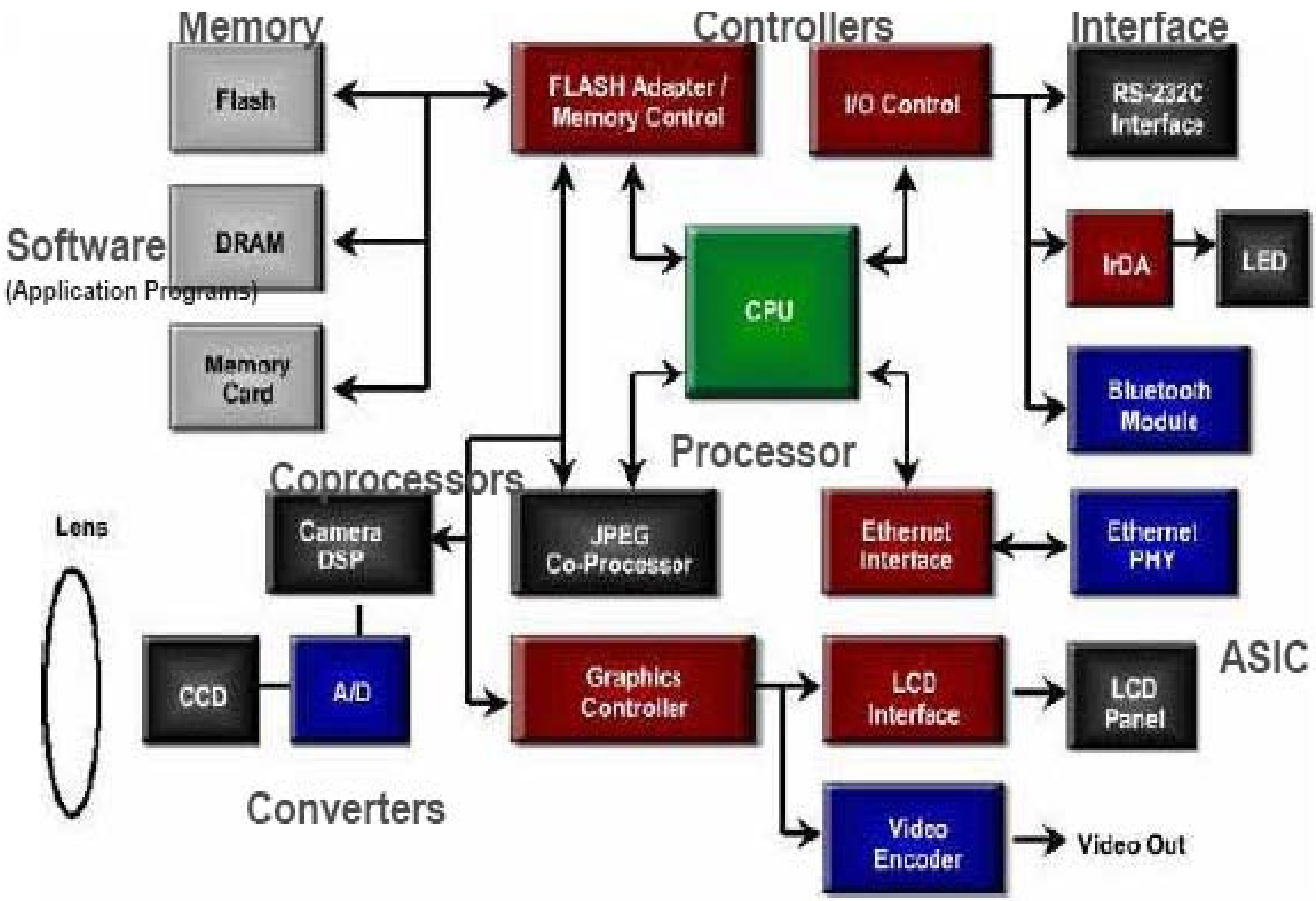
Project overview

- Get access to CSE 3219 lab
  - Robin Knox: rsknox@ucsd.edu
    - EBU3B Room 2248
    - Card programming times posted on the door
- Teams of two in each of the two groups
  - Group A: Project out today, due 4/27
  - Group B: Project out 4/27, due 5/25
- Part 1:
  - Install OS onto an embedded platform
  - Application-independent implementation of voltage scaling to ensure media player uses less energy
- Part 2:
  - Develop a loadable kernel module to make system more energy efficient

CSE 237A
Platforms

Tajana Simunic Rosing
Department of Computer Science and Engineering
University of California, San Diego.

Hardware Platform Architecture



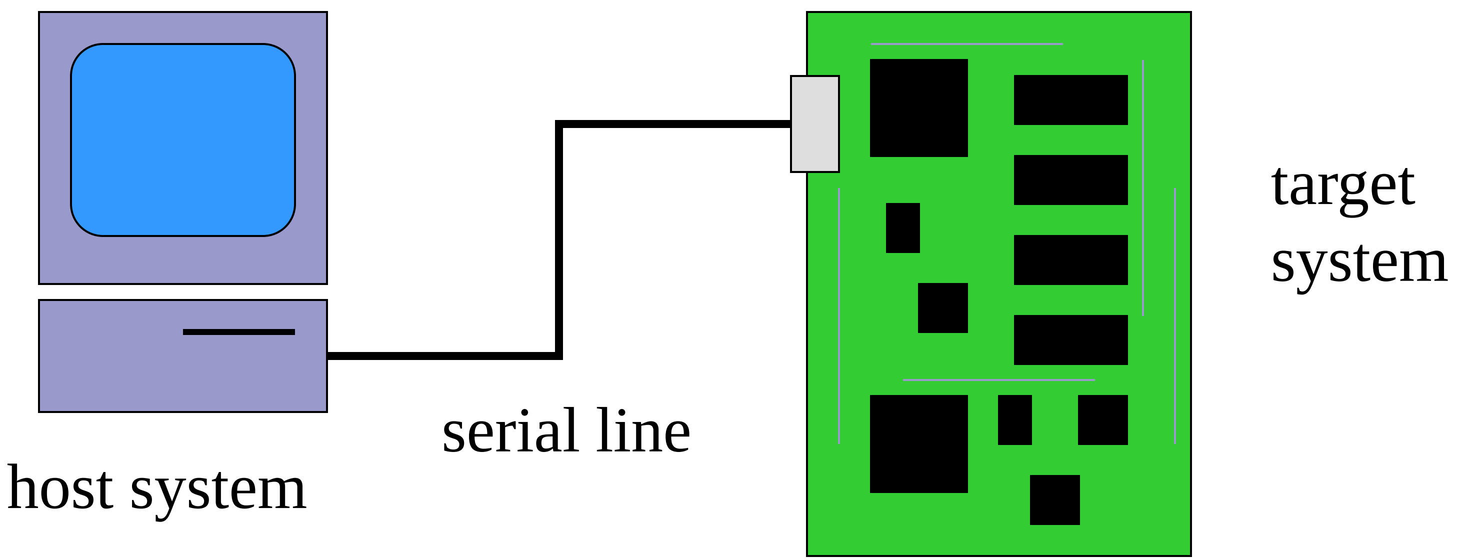
The PC as a Platform

- Advantages:
  - Cheap and easy to get
  - Rich and familiar software environment

- Disadvantages:
  - Requires a lot of hardware resources
  - Not well-adapted to real-time

Host / Target Design

- Use a host system to prepare software for target system:



Host-Based Tools

- Cross compiler:
  - Compiles code on host for target system
- Cross debugger:
  - Displays target state, allows target system to be controlled

Evaluation Boards

- Designed by CPU manufacturer or others
- Includes CPU, memory, some I/O devices
- May include prototyping section
- CPU manufacturer often gives out evaluation board netlist---can be used as starting point for your custom board design

Adding Logic to a Board

- Programmable logic devices (PLDs) provide low/medium density logic
- Field-programmable gate arrays (FPGAs) provide more logic and multi-level logic
- Application-specific integrated circuits (ASICs) are manufactured for a specific purpose

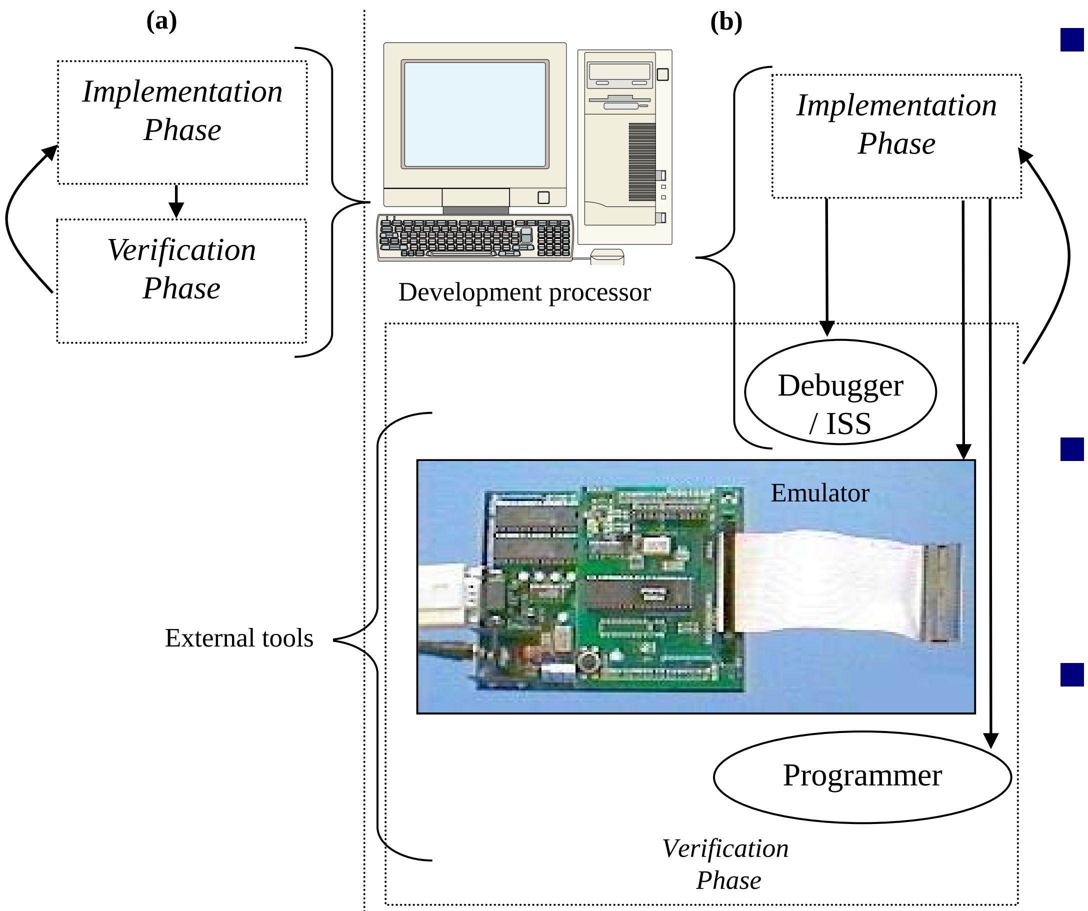
How To Exercise Code

- Run on:
  - Host system
  - Target system
  - Instruction-level simulator
  - Cycle-Accurate simulator
  - Hardware/Software co-simulation environment

Debugging Embedded Systems

- Challenges:
  - Target system may be hard to observe
  - Target may be hard to control
  - May be hard to generate realistic inputs
  - Setup sequence may be complex

Testing and Debugging



- ISS
  - Gives us control over time – set breakpoints, look at register values, set values, step-by-step execution, ...
  - But, doesn't interact with real environment
- Download to board
  - Use device programmer
  - Runs in real environment, but not controllable
- Compromise: **Emulator**
  - Runs in real environment, at speed or near
  - Allows you to stop execution, examine CPU state, modify registers.

Debuggers

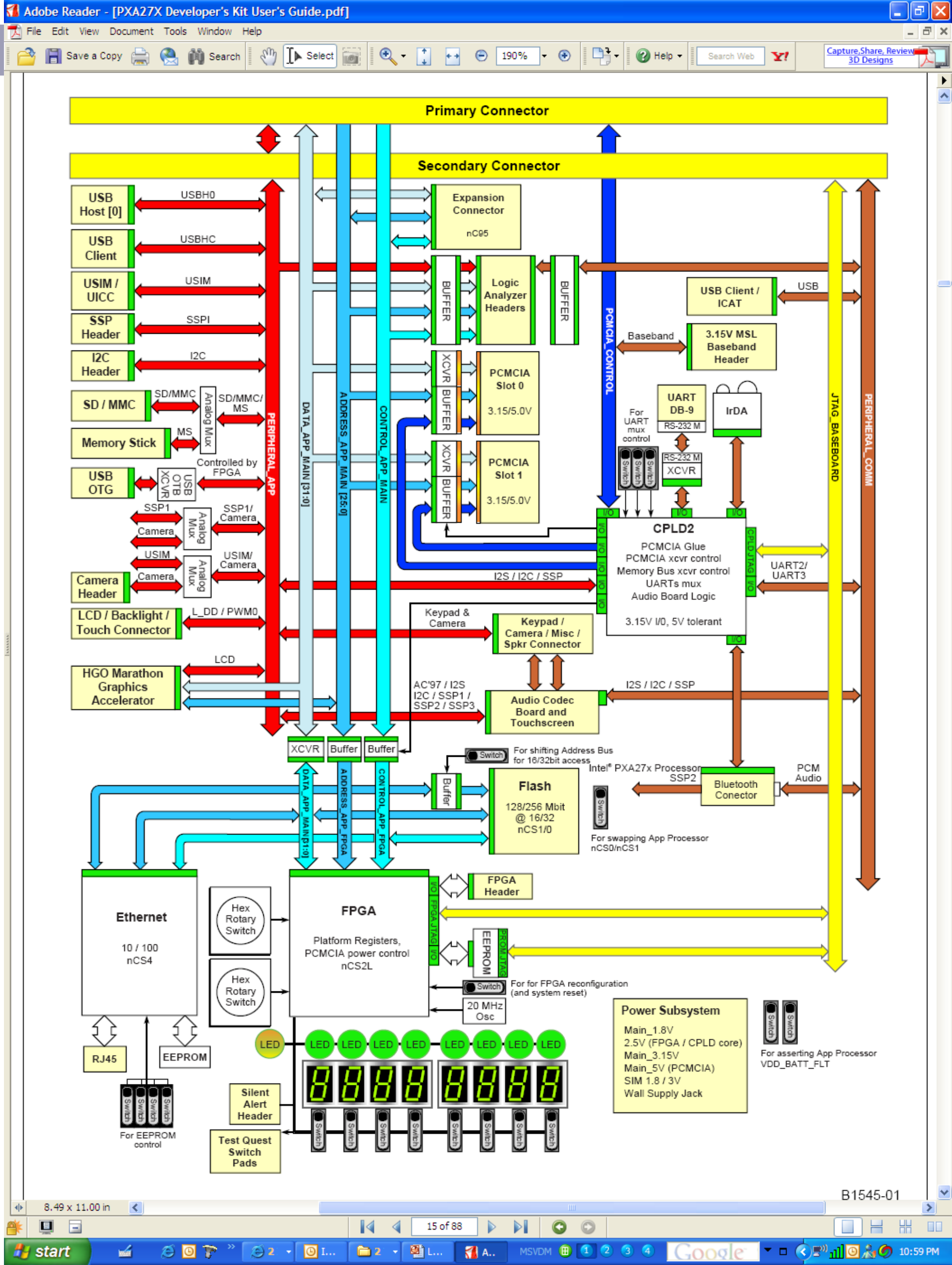
- A monitor program residing on the target provides basic debugger functions
- Debugger should have a minimal footprint in memory
- User program must be careful not to destroy debugger program, but should be able to recover from some damage
- Breakpoints are very useful
  - Replace the break-pointed instruction with a subroutine call to the monitor program

Breakpoint Handler Actions

- Save registers
- Allow user to examine machine
- Before returning, restore system state
  - Safest way to execute the instruction is to replace it and execute in place
  - Put another breakpoint after the replaced breakpoint to allow restoring the original breakpoint

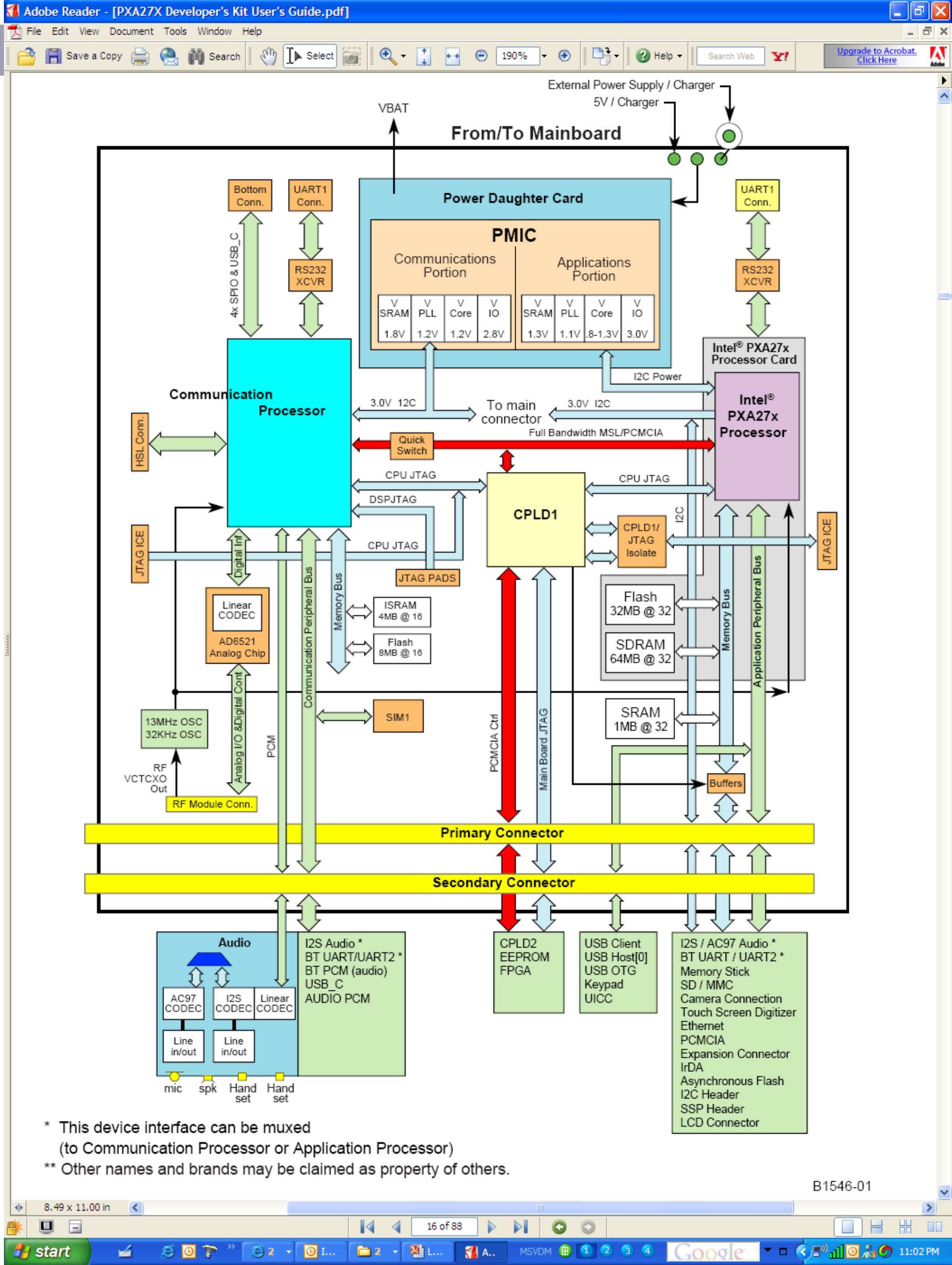
Platforms: XScale

- It's got it all!
  - Everything one would need to develop a next generation cell phone or PDA
- Main board block diagram:



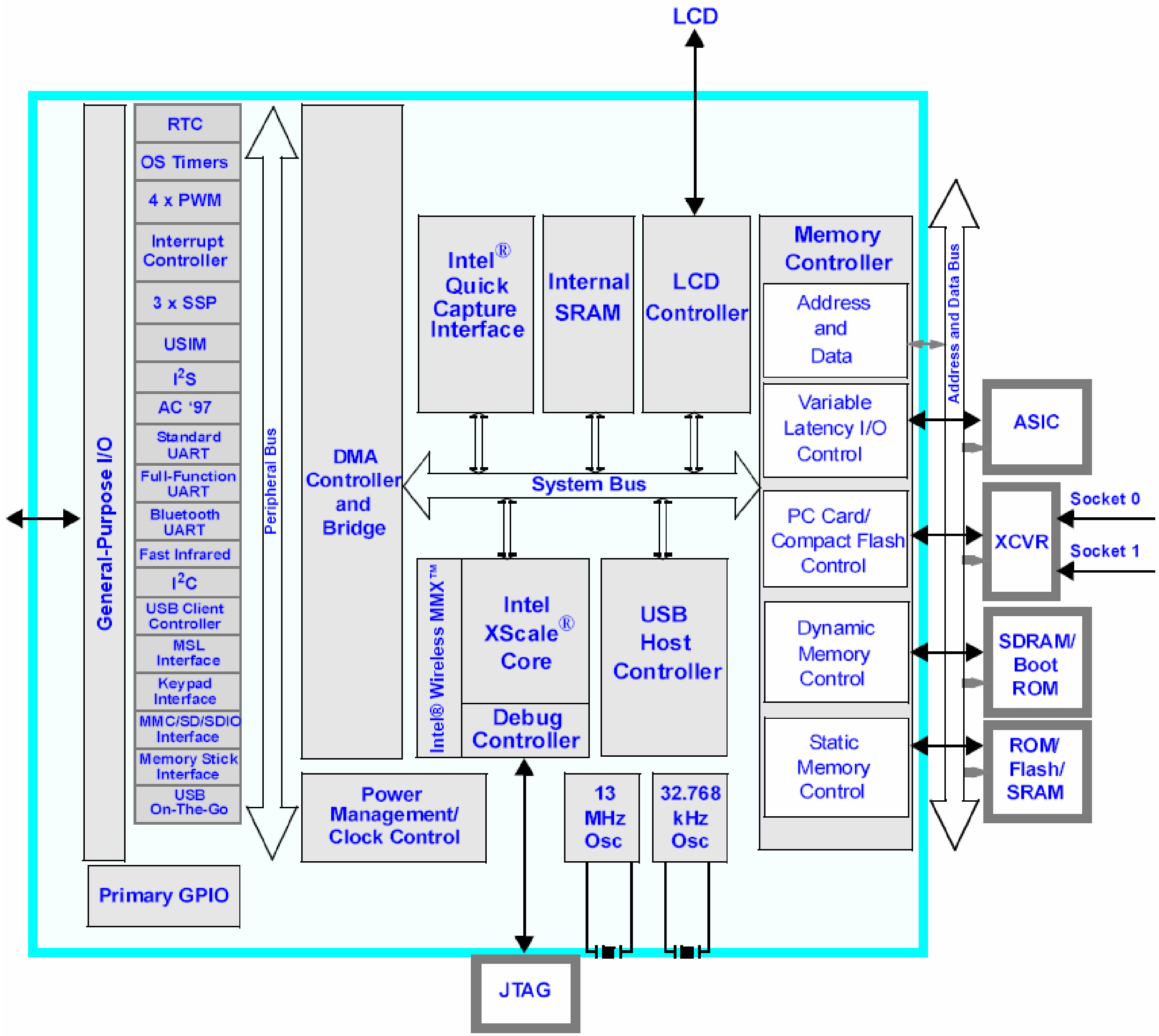
Platforms: XScale

- Daughter board block diagram

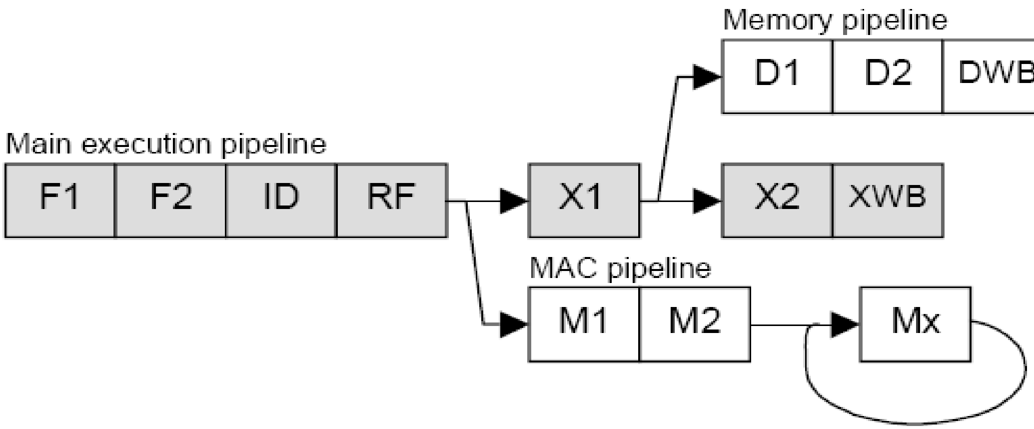


XScale Processor

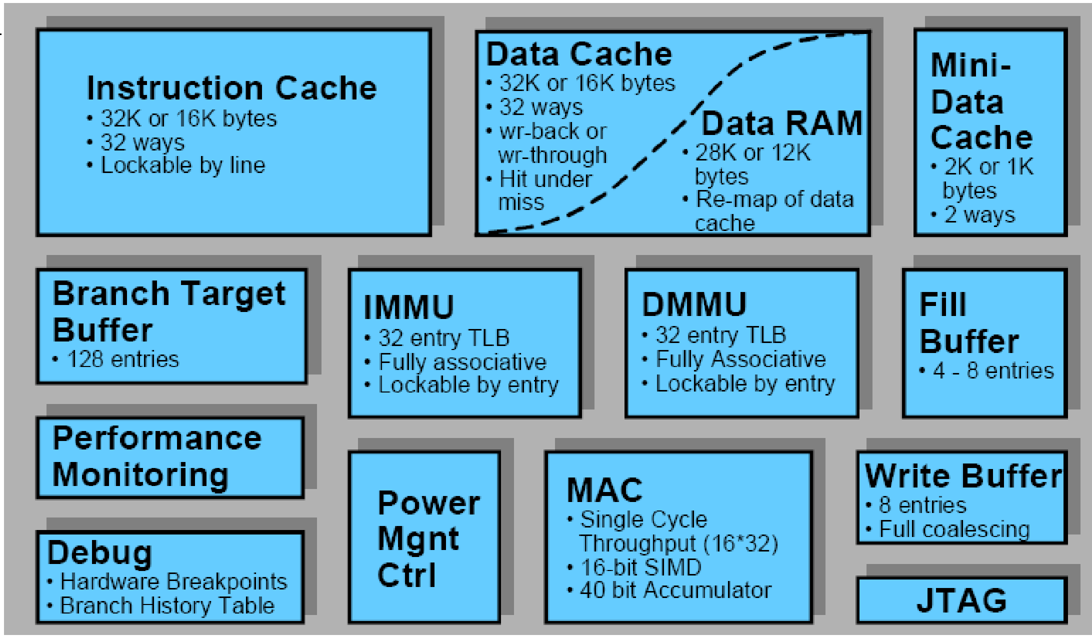
- 7 stage pipeline
- 32bit
- 32 KB instr/data cache; 2kb mini data cache
- 256 Kb SRAM
- Support for various peripherals
- CPU freq: 100-600 MHz; voltage down to 0.85 V
- Found in Blackberry, Treo, IPAQ, etc.



XScale Architecture



Pipe / Pipestage	Description
Main Execution Pipeline	Handles data processing instructions
IF1/IF2	Instruction Fetch
ID	Instruction Decode
RF	Register File / Operand Shifter
X1	ALU Execute
X2	State Execute
XWB	Write-back
Memory Pipeline	Handles load/store instructions
D1/D2	Data Cache Access
DWB	Data cache writeback
MAC Pipeline	Handles all multiply instructions
M1-M5	Multiplier stages
MWB (not shown)	MAC write-back - may occur during M2-M5



CSE237a Project Part 1: Install OS & the development environment

Introduction to Mainstone

- Mainstone
  - Referred as “TARGET”
  - Has enough power to run Linux

- Specifications
  - Intel PXA27x Processor
  - SMSC Ethernet Chip(10/100)
  - QVGA LCD
  - 9-pin Serial Port
  - JTAG Port
  - SD Card
  - Camera
  - Detailed information in doc/PXA27X Developer’s Kit User’s Guide.pdf

Introduction :: Your PC

- Referred as “HOST”
- Dual operating systems
  - Linux Redhat 9.0 for…
    - Kernel building
    - Application developing and debugging
  - Windows XP for…
    - Flash programming
    - Serial communication
- Communicate with the target via…
  - Serial cable
  - Cross ethernet cable (tftp, NFS, remote debugging..)
  - JTAG cable

Introduction :: Software Package 1/2

- 02-25-2005(extracted)
  - bin
    - Prebuilt binaries
    - Prebuilt toolchains
  - src
    - blob
    - kernel ←- kernel 2.6.9 and patch
    - qpe
    - rootfs
      - gdb ←- gdb build dir
      - camera ←- sample app build dir
    - usbdnet

Introduction :: Software Package 2/2

- doc
  - Related documents
- util
  - JFlash_win
    - JFlash flash programming program for windows
  - tftp_win
    - TFTP server program for windows

Introduction :: Mainstone vs Your Cell Phone

- Board Bring-up
  - Programming a firmware update program in your cell phone ROM
    - usually done only in factory
- Running a Linux Kernel
  - Update your phone with a new firmware
- Building Your Own Linux Kernel
  - Build your own phone firmware
- Developing Your Own Linux App
  - Develop a game for your phone

Cell Phone Using PXA27x

Board Bring-up

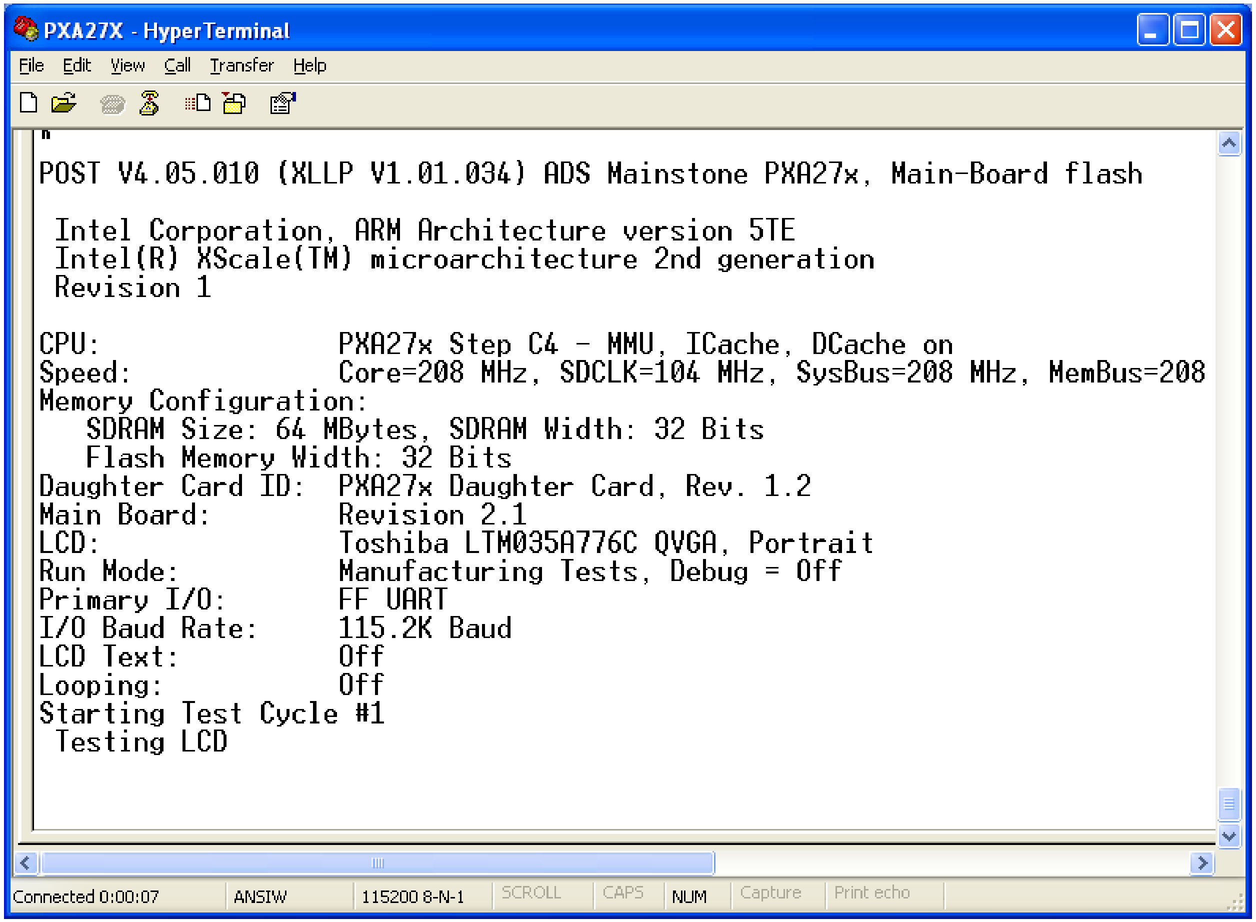
- Objective
  - Get the board up and running
    - Give a birth to the board!
- Tasks
  - Serial Communication
  - Flash Programming
  - Bootloader

Board Bring-up :: Serial Communication 1/2

- Serial Communication?
  - To check what is going on the platform
  - UART(serial device) driver is very easy to implement

- To Do
  - Connect a serial cable between the target and the host
  - Run a terminal program with 115200-8-N-1
    - Start->Accessories->Communications->HyperTerminal
  - Switch on the platform
  - Check the POST(Power On Self Test) works
    - Refer to doc/Diagnostics.pdf

Board Bring-up :: Serial Communication 2/2



POST using serial communication

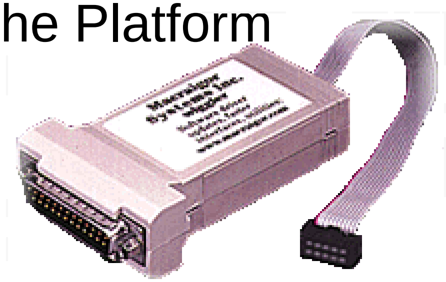
Board Bring-up :: Flash Programming 1/3

- Flash Programming?
  - Embedded systems also need a kind of “BIOS” program
  - BIOS(or bootloader) resides in non-volatile memory
  - Some external force is needed on board bring-up stage
    - ROM programmer
    - JTAG

- JTAG?
  - A way to manipulate hardware by external communication channel
  - With JTAG, download image or program flash image is possible
  - To learn more, see http://www.embedded.com/story/OEG20021028S0049

Board Bring-up ::
Flash Programming 2/3

- To Do
  - Install JFlash(util\Jflash_win\Jflash_MM_V5_01_007.exe)
    - Refer to RelNote_JFlashmm_v5_01007.pdf
    - Make sure giveio.sys be installed
  - Extract util\JFlash_win\JFlash_PXA27x_DataFiles.zip to JFlash directory
    - e.g.) c:\Program Files\Intel Corporation\JFlash_MM
  - Connect JTAG cable between the PC and the Platform



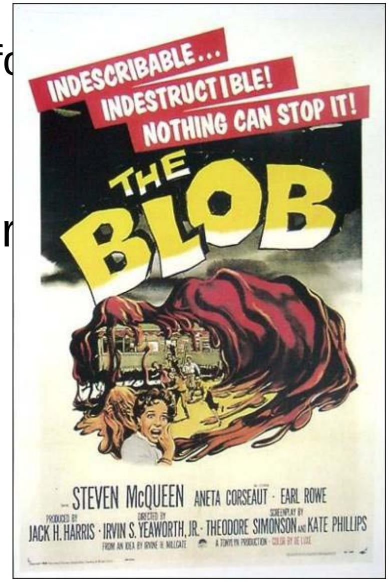
NOTE : JFlash is Windows ONLY!!!

Board Bring-up :: Flash Programming 3/3

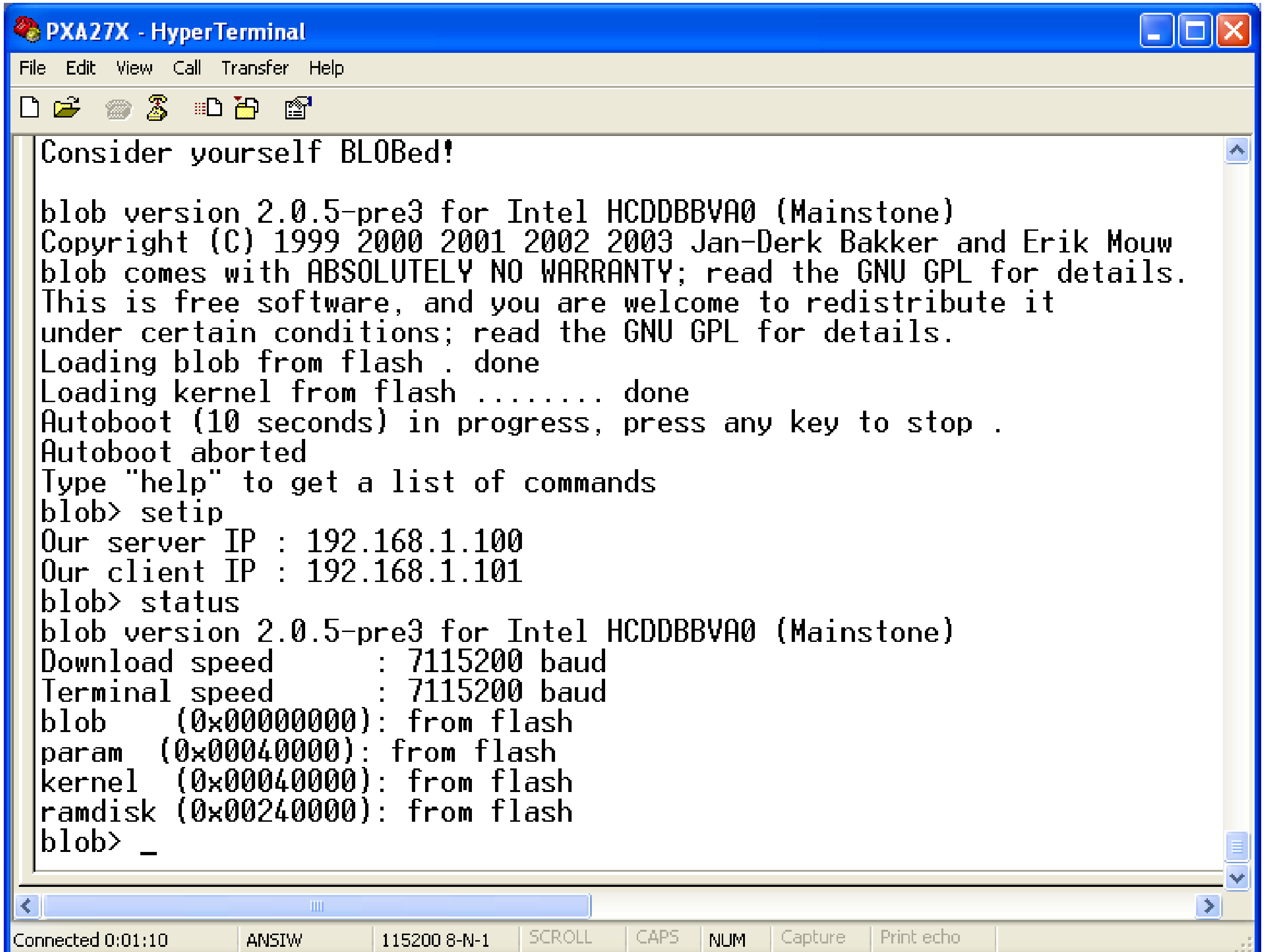
- To Do (continued)
  - Set SW7 of the platform to SWAP mode
    - SW7 selects which of the two flash memories to be mapped to CS0(the memory bank including address 0x0 - boot address)
    - You can still run POST program when you restore SW7 to normal mode
  - Switch on the platform
  - Run JFflashmm.exe
    - Platform file : bulbcx
    - Binary file : \bin\blob-smc91x
  - Reboot the platform
    - It takes a little(20 - 30 seconds), please be patient
  - See Blob prompt appears

Board Bring-up :: Boot Loader 1/2

- Boot Loader?
  - Role
    - Initialize basic hardware
    - Transfer the machine control to an OS
  - Ex) BLOB, UBoot, RedBoot, Lilo…
  - In many cases, porting a boot loader is the first task for embedded systems
    - Fortunately, Intel already ported BLOB for Mainstone platf
- BLOB (Boot Loader OBject)
  - Originally developed for StrongARM (the direct ancestor
  - Features
    - TFTP download via ethernet
    - Flash programming
    - Linux booting
    - Many others
  - http://www.lart.tudelft.nl/lartware/blob/



Board Bring-up :: Boot Loader 2/2



BLOB

Running Linux Kernel

- Objective
  - Make Linux run on your target
- Tasks
  - TFTP Communication
  - Install Linux Images
  - Explore Embedded Linux

Running Linux Kernel:: TFTP Communication 1/2

- Server
  - Windows Host
    - Install /util/tftp_win/tftpd32.280.zip
  - Linux Host
    - Edit /etc/inetd.conf and uncomment a line
      - "tftp dgram udp wait root /usr/sbin/tcpd in.tftpd"
    - Edit /etc/xinetd.d/tftp
      - disable=yes -> disable=no
      - Server_args = -s /$(TFTP_PATH)

NOTE : Check firewall is off!

Running Linux Kernel:: TFTP Communication 2/2

- Client
  - Connect a cross ethernet cable between the host and the target
  - Set IPs of server and client
    - blob> setip server ($server_IP)
      - Ex) setip server 192.168.1.100
    - blob> setip client ($client_IP)
      - Ex) setip client 192.168.1.101
    - blob> setip
      - Our server IP : 192.168.1.100
      - Our server IP : 192.168.1.101

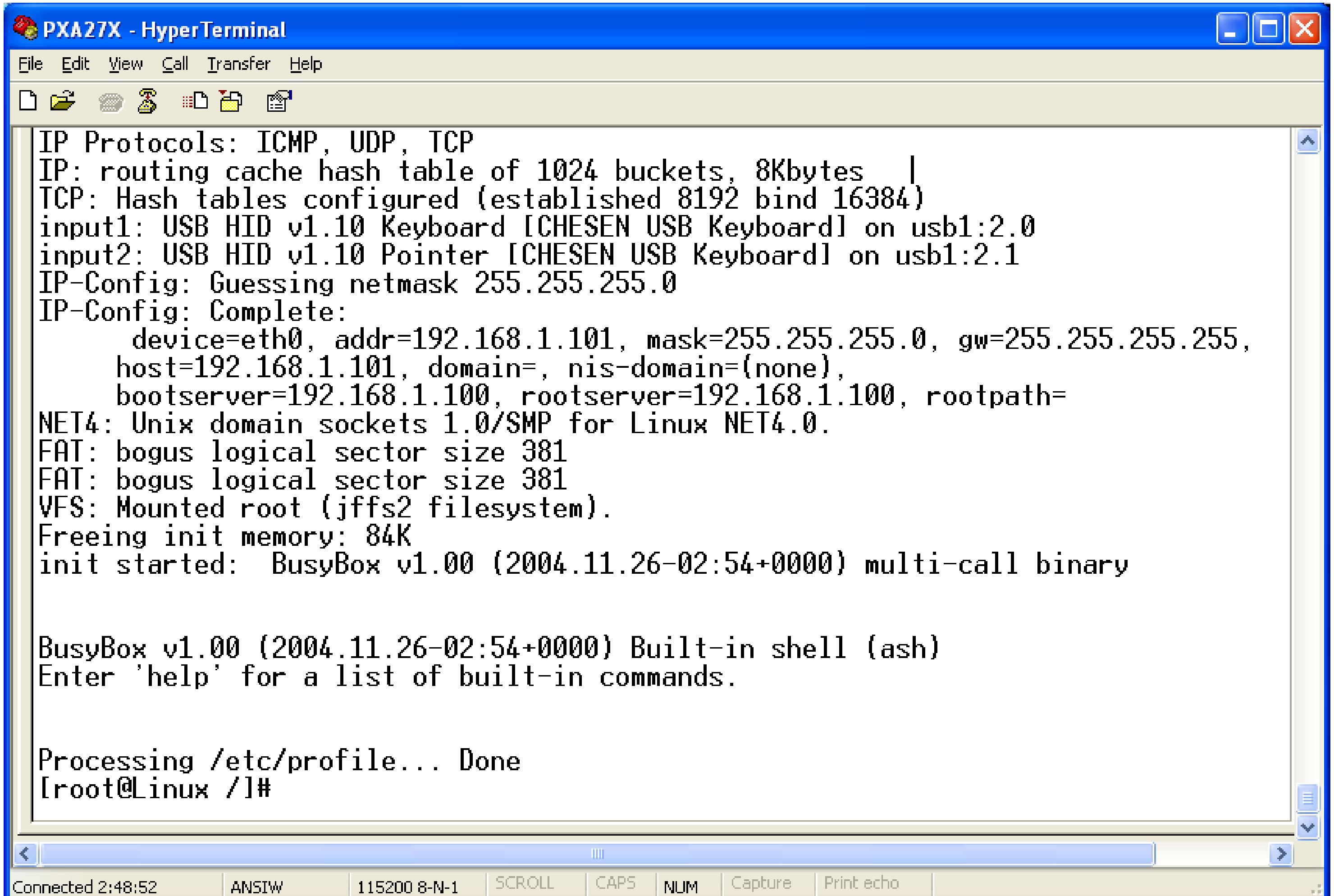
  - Download
    - blob> tftp filename
    - The requested file should be in server tftp directory

Running Linux Kernel:: Install Linux Images 1/2

- Install root filesystem image to flash
  - blob> tftp rootfs_x32_16M.jffs2
  - blob> fwrite 0xa1000000 0x240000 0x1000000
    - Write from 0xa1000000 to 0x240000 with size 0x100000
- Install kernel image to flash
  - blob> tftp zImage.qvga
  - blob> fwrite 0xa1000000 0x40000 0x200000
    - Write from 0xa1000000 to 0x40000 with size 0x200000
- Reboot
  - blob> reload kernel
  - blob> boot

NOTE: Programming flash memory in blob is much faster than JTAG

Running Linux Kernel:: Install Linux Images 2/2



Linux on Mainstone!

Running Linux Kernel:: Explore Embedded Linux 1/2

- Busybox
  - Essential utilities in one binary
  - http://www.busybox.net/downloads/BusyBox.html
- ci-capture
  - Capture your face with Mainstone camera
  - [root@Linux /]#ci-capture 176 144

Running Linux Kernel:: Explore Embedded Linux 2/2

- Setup Network
  - % /sbin/ifconfig eth0 $IPADDR netmask $NETMASK broadcast $BROADCAST
  - % /sbin/route add default gw $GATEWAY metric 1
  - Add DNS entries in /etc/resolv.conf
    - Cross cable communication don't need DNS, though.
  - Try ping, telnet, tftp…

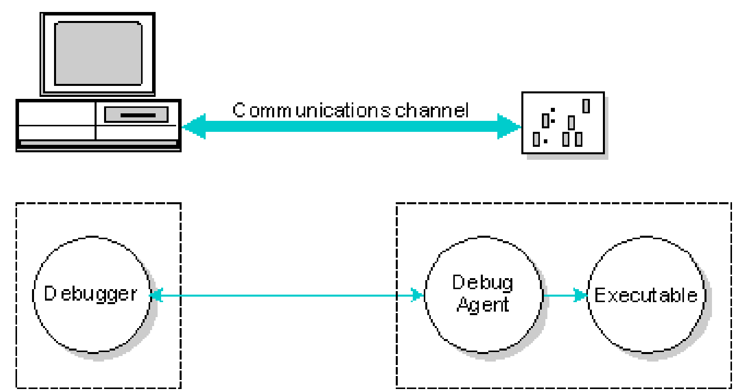
Building Your Own Linux Kernel

- Objective
  - Make your own kernel!

- Tasks
  - Install Cross Toolchain
  - Build a Customized Kernel

Building Your Own Linux Kernel :: Install Cross Toolchain 1/2

- Cross Toolchain?
  - Tools for building target applications
    - Compiler
    - Assembler
    - Linker
    - Runtime Library
  - 'Cross'
    - running on a host(i386)
    - making results for a target(xscale)
  - GNU Cross Toolchain
    - binutil 2.14.90
    - gcc 3.3.4
    - glibc 2.3.2

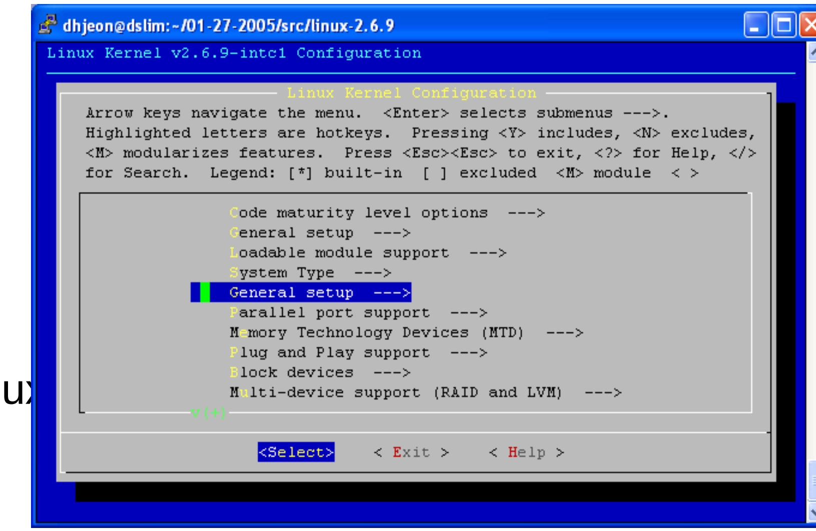


Building Your Own Linux Kernel :: Install Cross Toolchain 2/2

- To Do
  - Untar
    - % tar –xvzf arm-linux-toolchain-bin-11-26-04.tar.gz –C /home/embedded/local
  - Add path
    - % export PATH=/home/embedded/local/arm-linux/bin:$PATH
      - It is a good idea to add the path in your .bashrc file
  - Test
    - % arm-linux-gcc –v
      - Print out your cross compiler configuration

Building Your Own Linux Kernel :: Build a Customized Kernel 1/2

- Basic Menu Configuration
  - Patch kernel
    - % tar –xvzf linux-2.6.9.tar.gz
    - % cat patch-2.6.9-intc1 | patch -p1
  - Set environment variables
    - % export CROSS_COMPILE=arm-linux
    - % export ARCH=arm
  - Make a default configuration
    - % make mainstone_defconfig
  - Set your own configuration (optional)
    - % make menuconfig
  - Build
    - % make oldconfig
    - % make zImage
  - Check
    - New kernel image file is /$(linux)/arch/arm/boot/zImage
    - Download the image in blob, and test it!



Building Your Own Linux Kernel :: Build a Customized Kernel 2/2

- Mc

.config	kernel configuration parameters for the BSP
Documentation/	documentation on various areas of the kernel
Makefile	top-level Linux Makefile, uses Makefiles in subdirectories
Rules.make	common make rules shared among Makefiles
arch/	**architecture (CPU)-specific source code**
crypto/	cryptographic algorithms
drivers/	**device driver source code**
fs/	file system source code
include/	header files, subdirs for linux/ and asm-<arch>/
init/	after bootup, Linux uses this code to init user-space realm
ipc/	SysV IPC source code
kernel/	common kernel source (CPU-specific parts are under arch/)
lib/	miscellaneous system utilities
mm/	common virtual memory source (more under arch/)
net/	source for network protocols, stacks, standards
scripts/	scripts to support configuring and building Linux

Developing Your Own Apps

- Objective
  - Build and Debug Your Own Apps
- Tasks
  - Build Your Own Apps
  - Setup NFS
  - Build Cross GDB
  - Debug Remote Target

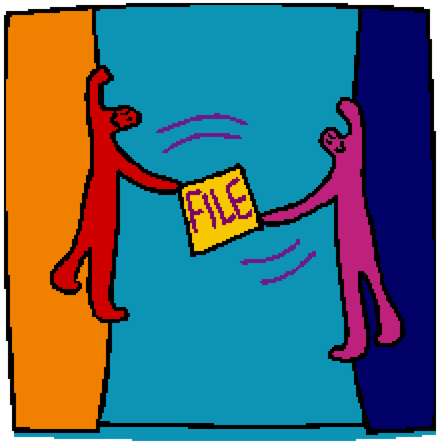
Developing Your Own Apps :: Build Your Own Apps

- Application Building
  - Cross development
    - Native toolchain is too large for many embedded systems

- To Do (ci-capture example)
  - Change directory
    - % cd /src/rootfs/
  - Uncompress a tar file
    - % tar –xvzf camera.tar.gz
  - Change directory
    - % cd camera
  - Edit Makefile
    - LINUX_INCLUDE = $(linux_dir)/include
  - Build
    - % make

Developing Your Own Apps :: Setup NFS

- NFS?
  - Host share its disk with other hosts(including target)
    - Compile on the host
    - Without explicit download, execute binary on the target



- To Do (host)
  - Add a line in /etc/exports
    - /$(export_path) $(target_ip_addr)(rw, no_root_squash)
    - e.g. /home/embedded 192.168.1.101(rw, no_root_squash)
  - Init NFS service
    - % /etc/rc.d/init.d/nfs stop
    - % /etc/rc.d/init.d/nfs start
    - % exportfs –rav

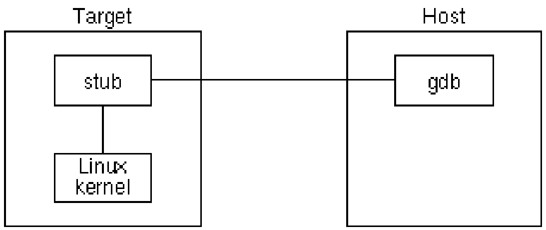
- To Do (target)
  - Mount NFS drive
    - % mount -t nfs –o nolock,nfsvers=3,tcp $(host_ip_addr):/$(export_path) /mnt/$(mnt_path)
    - e.g. % mount –t nfs –o nolock,nfsvers=3,tcp 192.168.1.100:/home/xscale /mnt/arm
  - Check the mounted directory
    - Use as if it is your local drive

Developing Your Own Apps :: Build Cross GDB

- To Do
  - Extract souce file
    - % cd /02-25-2005/src/rootfs/gdb
    - % tar –xvzf gdb-6.0.tar.gz
  - Configure for cross development
    - % cd gdb-6.0
    - % ./configure –target=arm-linux –prefix=/home/embedded/local/arm-linux
  - Build
    - % make
    - % make install

Developing Your Own Apps :: Debug Remote Target 1/2

- Remote Debugging
  - Target and host communicate debuggi
    - Small agent(gdbserver) on target
    - Cross GDB on host



- To Do (target)
  - % gdbserver $(target_ip):$(port_num) $(exec_name) $(args)
  - e.g. %gdbserver 192.168.1.100:1234 ./ci_capture 144 128
- To Do (host)
  - % arm-linux-gdb $(session_name)
  - (gdb) target $(target_ip):$(port_num)
  - e.g. (gdb) target remote 192.168.1.101:1234

Developing Your Own Apps :: Debug Remote Target 2/2

- Sample Session
  - ci-capture program
  - See GDB manual for detailed information

dhjeon@dslim:~/02-25-2005/src/rootfs/camera

```
[dhjeon@dslim camera]$ arm-linux-gdb
GNU gdb 6.0
Copyright 2003 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=arm-linux".
(gdb) target remote 132.239.17.104:1234
Remote debugging using 132.239.17.104:1234
0x41001550 in ?? ()
(gdb) symbol ci-capture
Reading symbols from ci-capture...done.               <- load symbol
(gdb) break overlay2_open                             <- set breakpoint
Breakpoint 1 at 0x86c8: file ci-capture.c, line 53.
(gdb) continue                                        <- go until breakpoint
Continuing.

Breakpoint 1, overlay2_open (dev=0x92c8 "/dev/fb2", bpp=24, format=3, xpos=50,
    ypos=100, xres=144, yres=128, map=0xbffffe44, yoff=0xbffffe3c,
    ylen=0xbffffe40, cboff=0xbffffe34, cblen=0xbffffe38, croff=0xbffffe2c,
    crlen=0xbffffe30, pitch=0xbffffe28) at ci-capture.c:53
53              if ( (!map) || (!yoff) || (!ylen) || (!cboff) || (!cblen) || (!c
roff) || (!crlen) )
(gdb) frame                                         <- stack frame
overlay2_open (dev=0x92c8 "/dev/fb2", bpp=24, format=3, xpos=50, ypos=100,
    xres=144, yres=128, map=0xbffffe44, yoff=0xbffffe3c, ylen=0xbffffe40,
    cboff=0xbffffe34, cblen=0xbffffe38, croff=0xbffffe2c, crlen=0xbffffe30,
    pitch=0xbffffe28) at ci-capture.c:53
53              if ( (!map) || (!yoff) || (!ylen) || (!cboff) || (!cblen) || (!c
roff) || (!crlen) )
(gdb) step                                          <- step 1 line
overlay2_open (dev=0x92c8 "/dev/fb2", bpp=24, format=3, xpos=50, ypos=100,
    xres=144, yres=128, map=0xbffffe44, yoff=0xbffffe3c, ylen=0xbffffe40,
    cboff=0xbffffe34, cblen=0xbffffe38, croff=0xbffffe2c, crlen=0xbffffe30,
    pitch=0xbffffe28) at ci-capture.c:56
56              fd = open(dev, O_RDWR);
```