ESP Gen 2
Software Overview

Revision 1.1

Derived from Outline in
/ESP/gen2/software/plan/softdesign.rtf

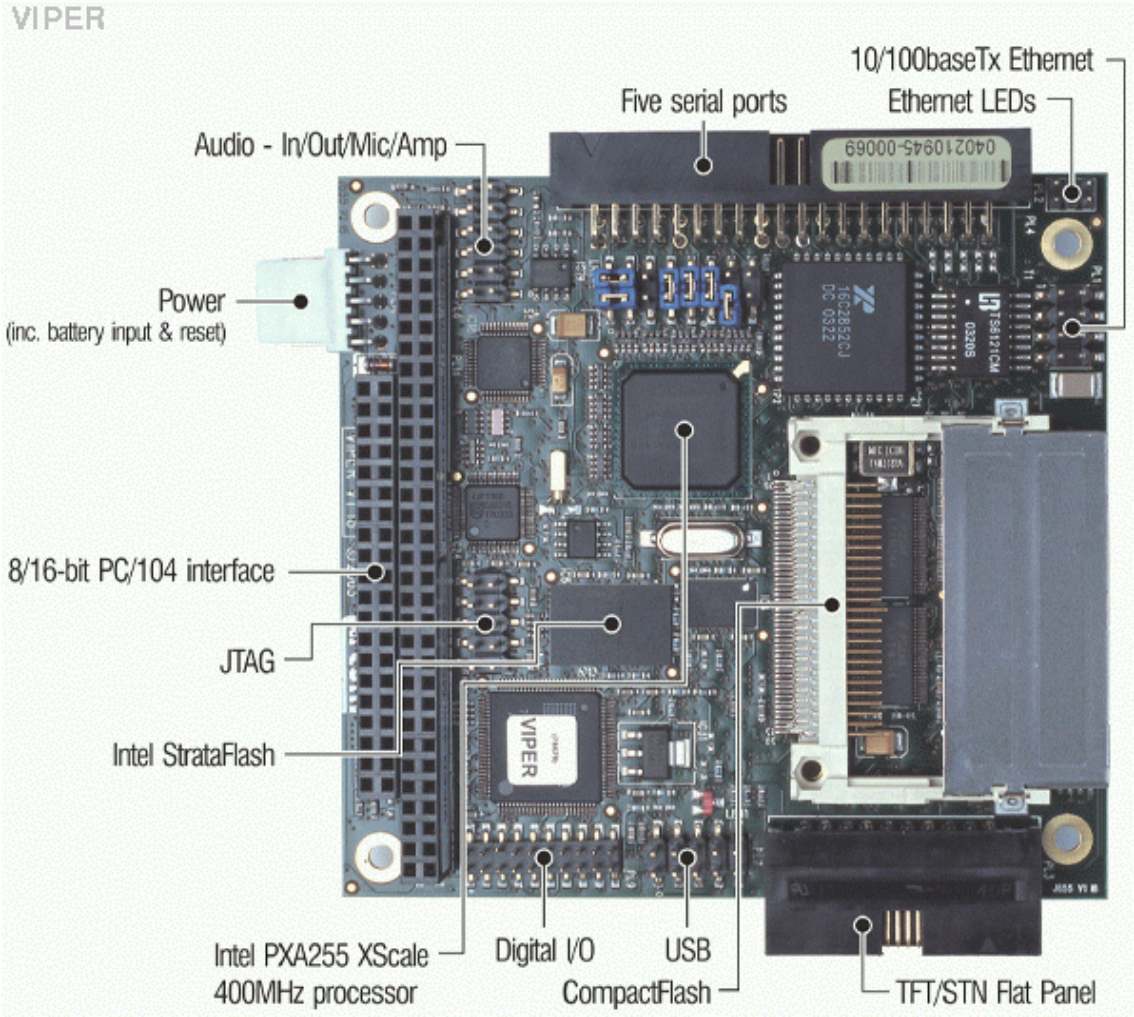
Brent Roman

December 1, 2004

Arcom's “Viper” PC/104 CPU

- Intel Xscale @400 Mhz, 2W @+5Vdc
  - Successor to StrongARM in our SideARM
- 64MB DRAM + 32MB flash
- Real-Time Clock with 5yr battery
- 100/10 Mbit Ethernet
- 5 serial ports
- Linux Support (Kernel version 2.4)
- Limited PC/104 bus support
- USB (v1) host interface

Arcom's “Viper” PC/104 CPU



MBARI

Host Software

- Any Linux platform
  - Had planned to support Windows, but why?!
- Same Ruby v1.6.8 Scripting Language
- X-windows graphics for:
  - Ruby's object class browser
  - text editors, CVS gui, etc.
  - Later for ESP's own GUI
- PC replaces embedded Host CPU for lab use

TI MSP430 Servo Controllers aka “Dwarves”

- 60KB of Flash, 2KB of RAM @~4Mhz
  - DCO servoed to a watch crystal in firmware!
- Programmed in ANSI 'C' + some assembly
- Bare metal, No real-time kernel
- All run exactly the same code:
  - Using ~40KB of code, 1.5KB of RAM
  - Multi-master I^2C @ 100kbps
    - I^2C bus address set by a tiny on board dip switch
  - Debug RS-232 port @ 38.4kbps

Why use I^2C?

- Very low power
- Adequate data rate (as high as 400kbs)
- Supported by new MSP430 variants
  - Many sensor chips also connect directly
- Supports Peer-to-Peer networking
  - Defines flow control, collision detection
- Supports up to 126 nodes (~40 electrically)
- Unlimited packet length
  - Not as amenable to hard real time as CAN

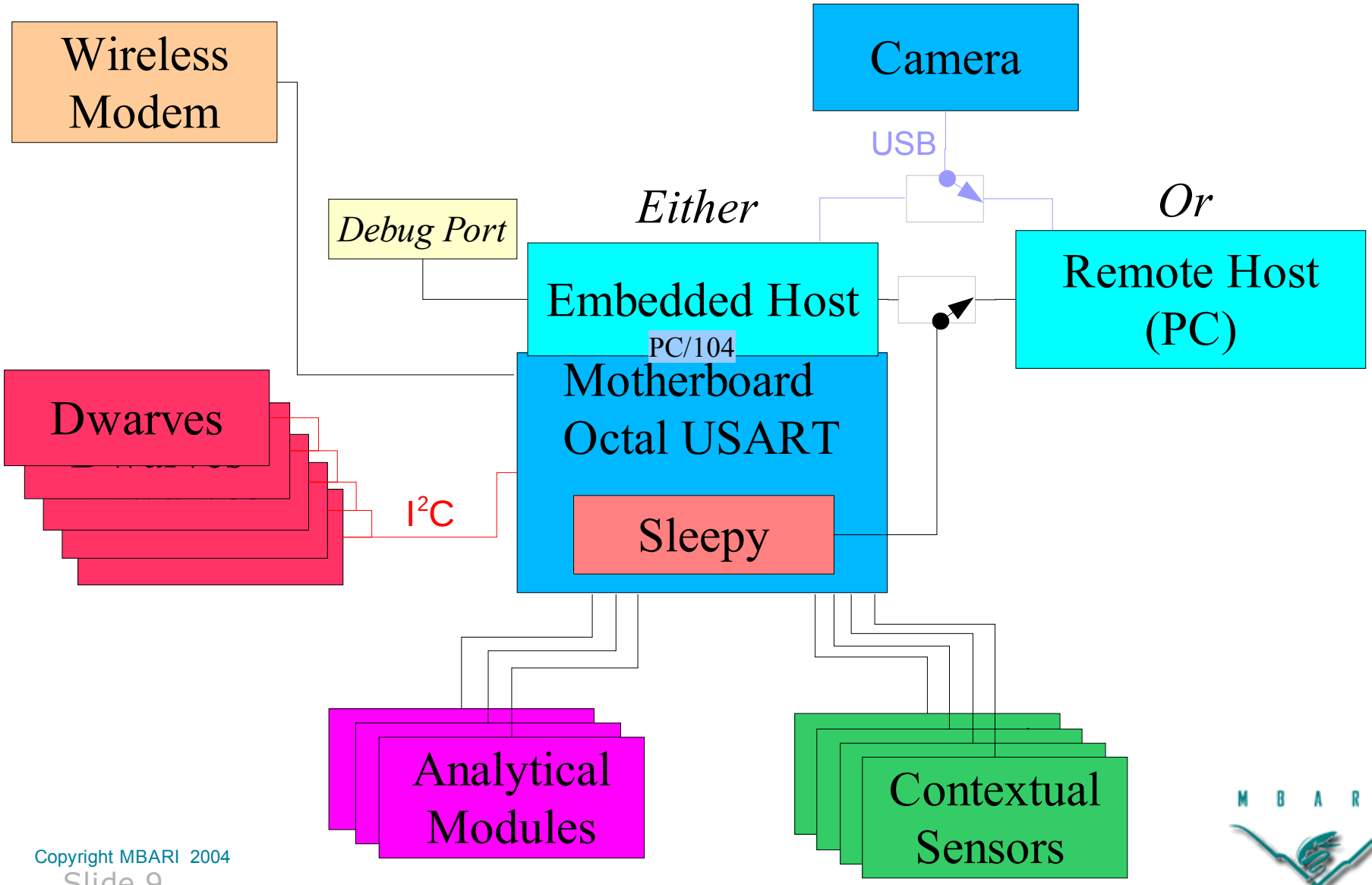
MSP430 Sleep Mode Controller aka “Sleepy”

- Marks time while the system is shut down
- Can access motherboard's 8 serial ports
- Wakes host and other micros when:
  - Preset delay expires, or
  - External event detected by contextual sensor, or
  - Wake-up packet received on the radio modem
- Serves as the host's gateway to the I^2C bus
- Runs at ~8Mhz to cope with high data rates

The Gateway Concept

- All high-level control multplexed onto a single 115kbps serial port
- Embedded host connects via this port
  - Uses PC/104 only for:
    - faster access to Octal USART
    - Access to optional PC/104 cards
- Development Host connects via same port
- Remote control possible via terminal server
  - Or even a freewave modem!

System Comms Overview



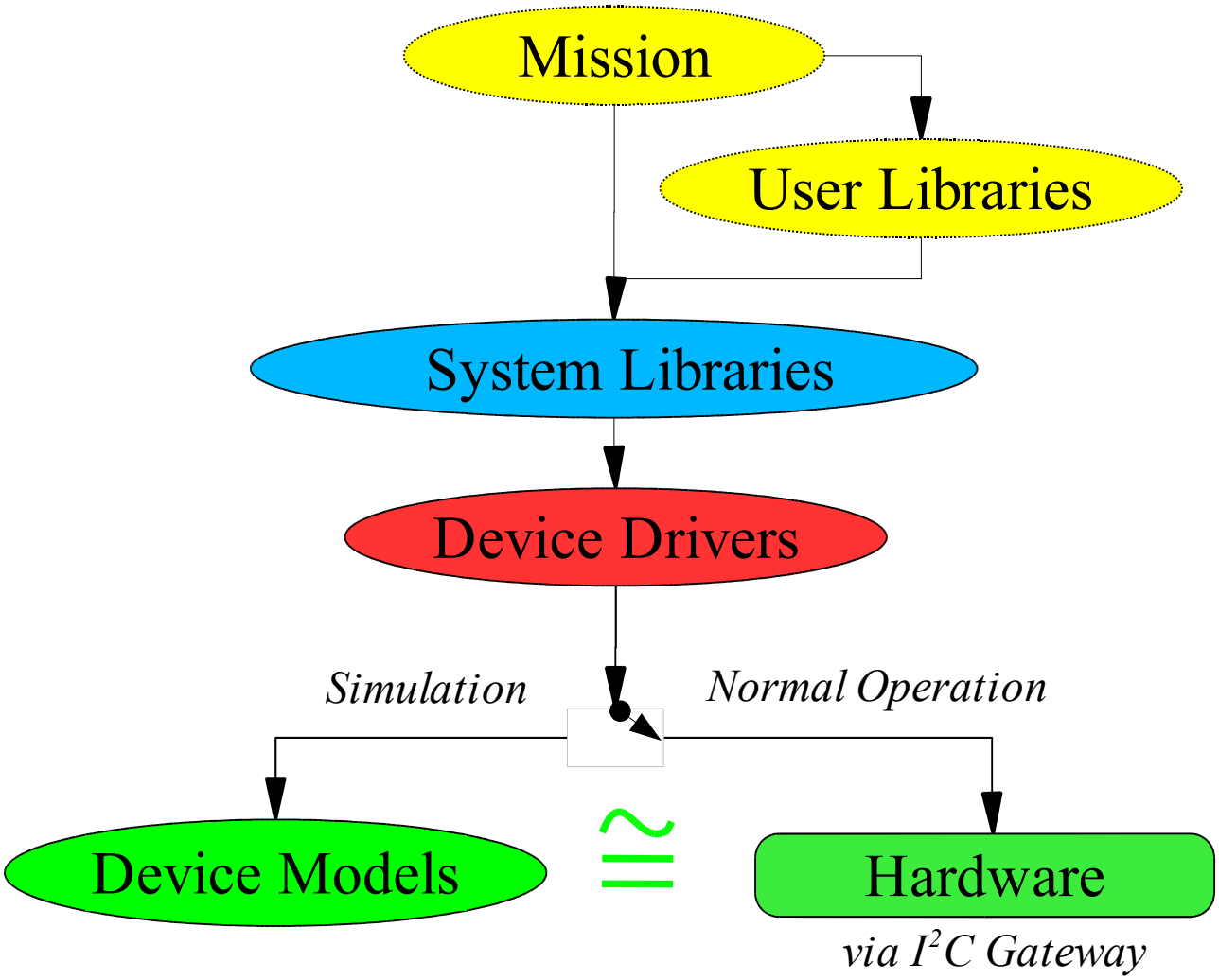
"Dwarf" Microcontroller Firmware Concepts

- Dwarves Close all real-time servo loops
  - So the host can contemplate its cybernetic navel
- Identical microcontrollers configured by host
  - They may be moved from one subsystem to another by changing cables and a network address dip switch
  - Less firmware development, fewer spares, easier config cntrl
- No Message Polling!
  - Dwarves master the I^2C bus when events warrant it
- All supported operations can be done in parallel
  - Heating while moving servo motors, etc.

Host Application Software

- Written entirely in Ruby
  - Aside from 1 (tiny) custom 'C' extension
- Supports three distinct operational modes:
  - With Actual hardware (normal operation)
  - Simulated hardware in simulated time
  - Simulated hardware in real time
- Runs protocol scripts as OS commands
- Or, takes them from its own interactive shell
  - Modified Interactive Ruby (aka 'irb')

Application Module Hierarchy



Mission Phases & Sequencing

- A mission is a sequence of “phases”
  - Each phase is typically a single Ruby script file
    - But it may be any OS command
  - Mission schedule determines when phases run
  - The ESP is powered off between phases
  - Phases may also be invoked from OS prompt
    - For testing in the lab

Mission Schedule

- Mission schedule defines when phases run
  - May be edited as the mission progresses
  - Is simply an association of Times with phases
    - Example ruby script defining a mission schedule:

```
Mission = schedule "UTC
   1/13/04  7:52",  "esp sample 2",     #sample puck in tube 2
  "         9:14",  "esp archive 4",    #archive puck in tube 4
  "1/15    22:10",  "esp process",      #1/15/04
  "         3:19",  "date",             #1/16/04-any OS command
  "2/03    12:19pm","esp sandhyb 6",    #2/03/04
  "from 4:19pm until 2/5 every +6:23", #checkContext on the tide
                   "esp checkContext" #...until 4:19pm 2/5

4.times {Mission.add "Monday 15:14:52pst", "esp phoneHome"}
```

Simulation Notes

- Simulate on embedded or development host
  - Simulating on the target insures all files present
  - Verifies reagents & pucks properly loaded, or
  - Tells users how to configure pucks & reagents
- Catches logic errors
  - Including mechanical interference between axes
- As fast as can be computed or in “real-time”
  - Next year's effort to include real-time animation

What's done

- User-Friendly, Object Oriented Control of:
  - Valves (Rotary & Solenoid), Manifolds
  - Syringes (Collecter & Sampler)
  - Thermal (just Collection Puck heat for now)
  - Multiple threads
- Logging with Timestamps of:
  - Executed High-Level Commands
  - Low-level Dwarf Messages sent and received
  - Unhandled Exceptions (Errors)
  - User Comments

Simulation

- Operational modes:
  - Real-Time with Real Hardware in the loop
  - Real-Time with Simulated hardware
  - Simulated Time with Simulated hardware
- Dynamic Simulation of:
  - Valves, Syringes and Puck Heating & Cooling
- Framework for adding more...

MSP430 Firmware

- Robust I^2C networking
- Complete Servo Motor control
  - 2 Channels, Trapezoidal Trajectory
  - Dynamic drive voltage compensation
  - Overcurrent Sensing
- Temperature Servo control
  - Some ongoing work
- Control of Rotary & Solenoid Valves
  - done

Embedded Host

- Tested Arcom Viper PXA-250 PC/104 CPU
  - Runs Ruby in ~4MB of RAM
  - Lots of room to spare
  - Boots to application in <30secs
  - USB host works in general
- USB camera tested
  - Works under Windows and Linux
  - Need to test Camera with Arcom board

Development Tasks in 2005

- Implement firmware for “sleepy” MSP430
  - A servo dwarf is I^2C gateway in our lab system
- Contextual Sensors & Analytical Modules
- Enhance Simulation
  - Track Fluid & Power use
  - Detect mechanical interference
- Sequence Mission “Phases”
  - Boot, run next script, save state, shutdown
- Port Everything to the embedded target
  - Not as hard as it sounds...

GUI Development

- We didn't get to it this year
  - All aspects of firmware development ran long
  - Firmware (I^2C) development was worst
    - 'Experimental' MSP430 variants are properly labeled
    - Worked around and documented numerous I^2C bugs
      - TI says “Thank you very much for your feedback”!
    - Planned one month task became 4+ months
    - But, our firmware workarounds are very stable
      - No significant performance hit
      - Haven't changed this code in 3 months!

GUI Plans

- Cartoon depiction of ESP state
  - Dynamically updated
  - Simulated or real hardware
  - Also depict resource use: reagents, pucks, etc.
- Commands in text window change state
- Build using National Instruments LabView
  - Already well established among our target users
  - OS agnostic (Windows, Linux, MacOS, Unix)
- Add input icons as time allows
  - ESP script in LabView dataflow diagrams?