TinyOS Installation

Matt M

Matt Miller

TOS Programming Environment

TinyOS Installation on MS-Windows

Tool Description

Other Topics

Conclusion – TOS Installed!

TinyOS Development Flow

- Design & Create Application Source Files (*.nc)
- Build Application Executable for a Specific Platform
- Load Application Executable onto Platform
- Run & Debug Application

TinyOS Development Tools

- CYGWIN Unix Shell Environment
- TinyOS Components (Modules & Configurations)
- NesC Pre-compiler
- AVR GNU Compiler, Assembler, & Linker
- UISP Device Programmer
- Host Application Tools – Java

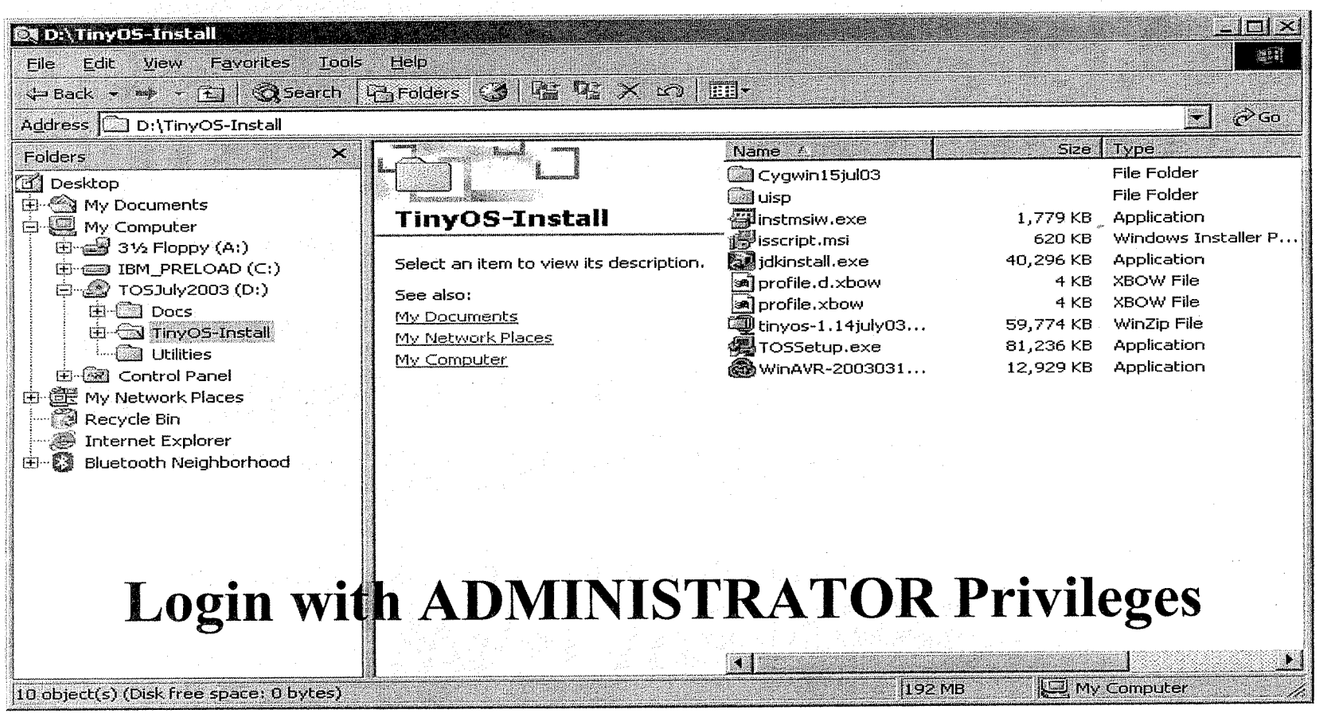
Installation Flow

- Install TinyOS 1.0
- Update Cygwin
- Update Compiler Tools – avr-gcc v3.3
- Update TinyOS - v1.x.14july03
- Install NesC – 1.1beta
- Update Utilities – uisp handle new hardware pre-config laptops
- Test all Tools
- If TOS Already installed – just do UPDATES !
- Problems? – Notify a Crossbow Trainer ☺

What We Need

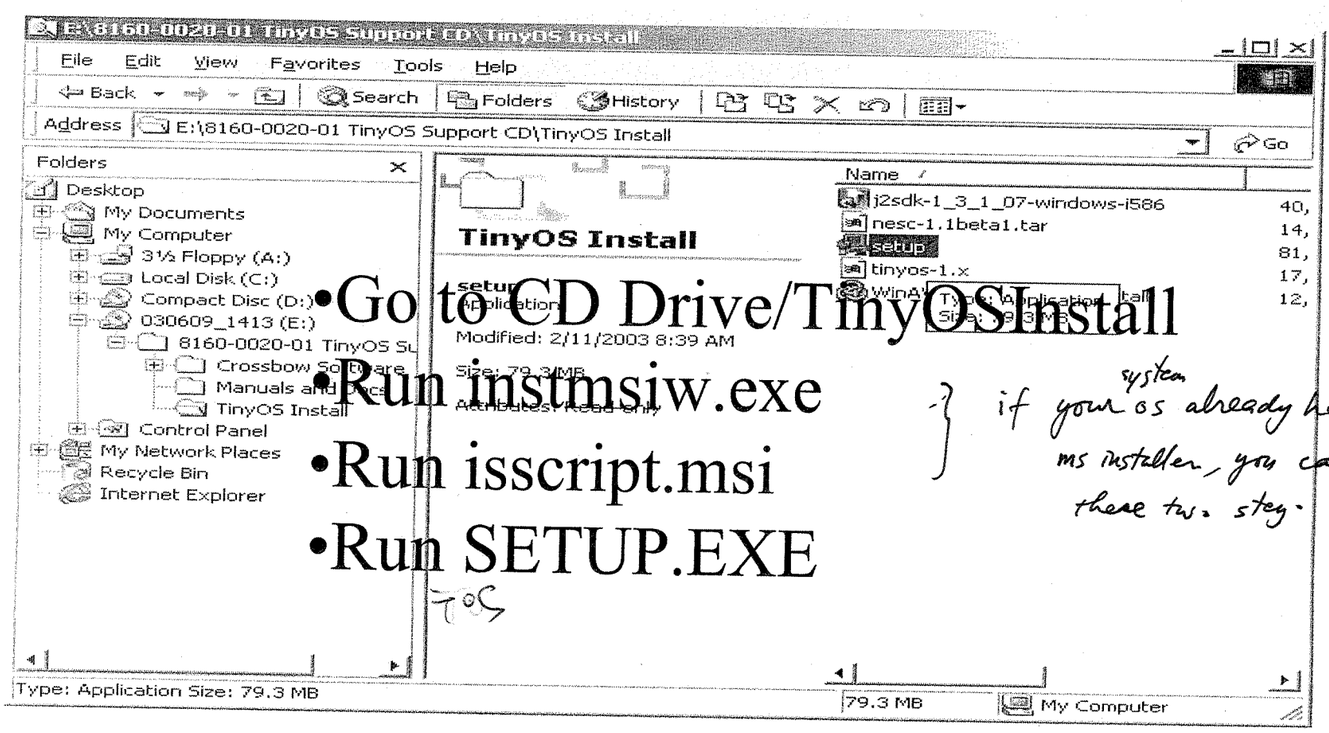
- Crossbow TOS Training CD-ROM
- MS Windows (XP, 2K, 98, NT)
- WINZIP
- Acrobat PDF Viewer
- Disk Space – 1GBytes
- Time
  - 50 minutes
- Mote Hardware
  - MIB500 Programmer, 2ea MICA2, Power & Batteries

TOS Installation from Xbow Disk



Login with ADMINISTRATOR Privileges

TinyOS Setup



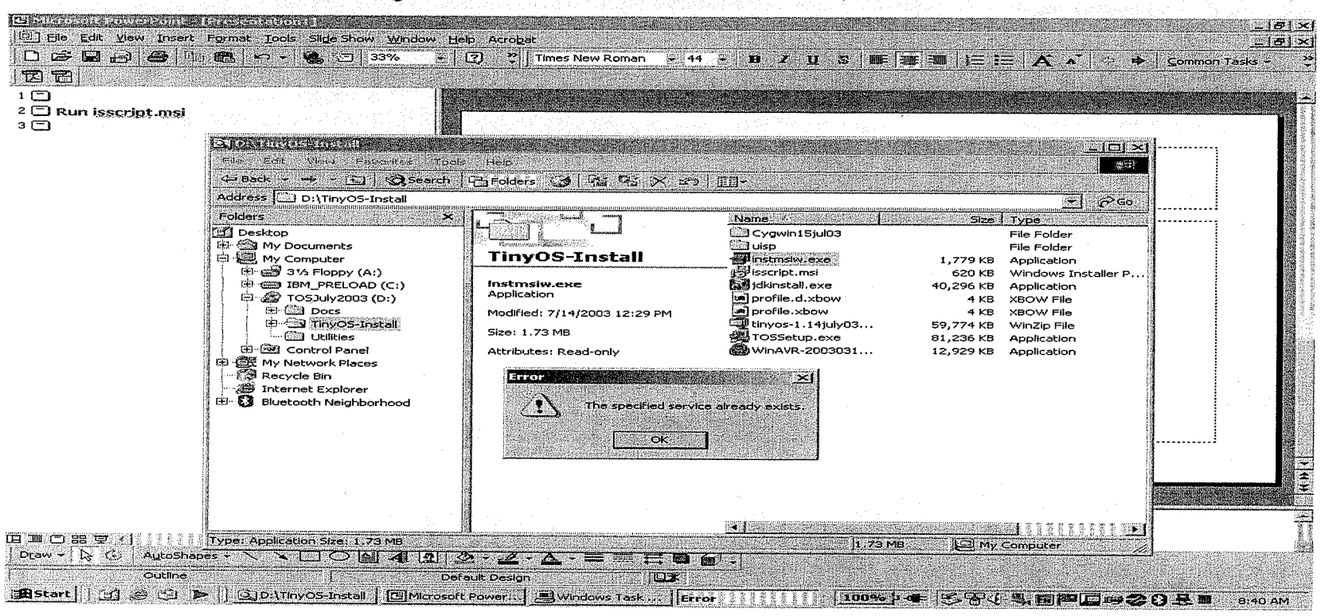
- Go to CD Drive/TinyOSInstall
- Run instmsiw.exe
- Run isscript.msi
- Run SETUP.EXE

TOS

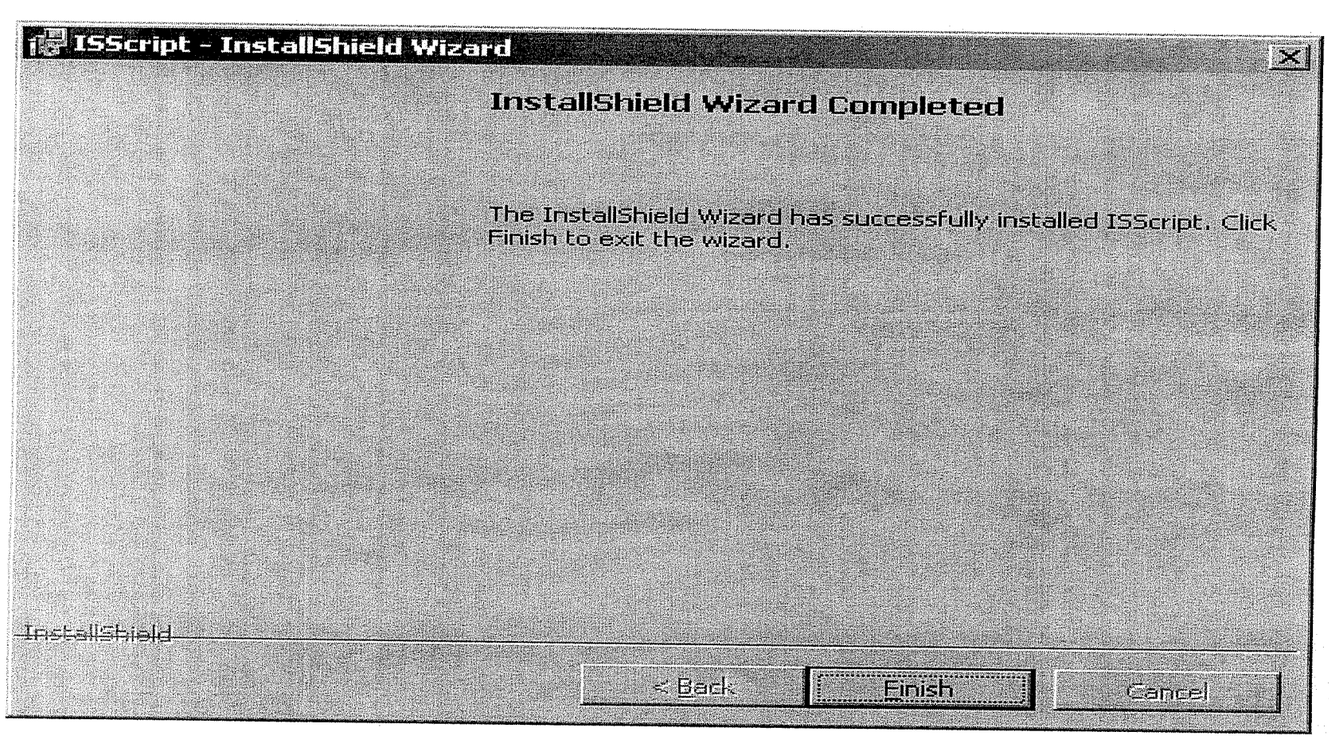
} if your system OS already has ms installer, you can skip these two steps.

Run Install Services

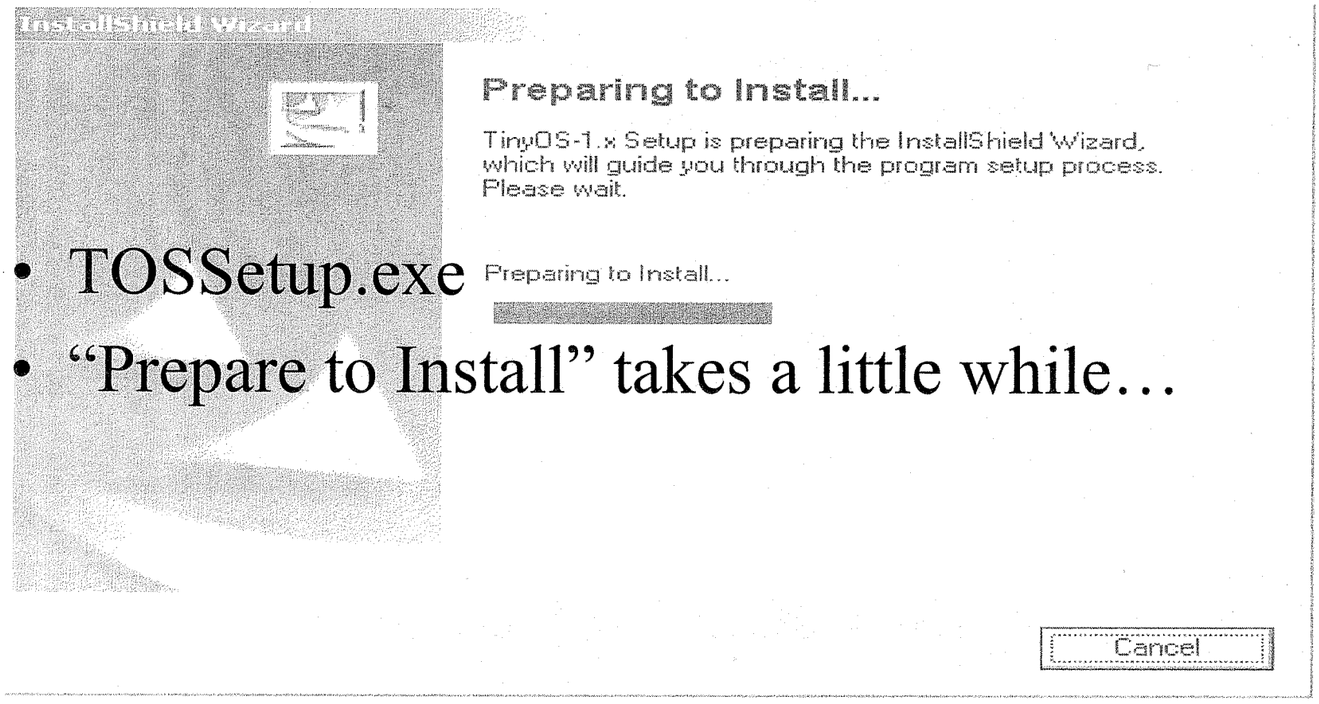
- cdrom:/Tinyos-Install/instmsiw.exe



Run isscript.msi

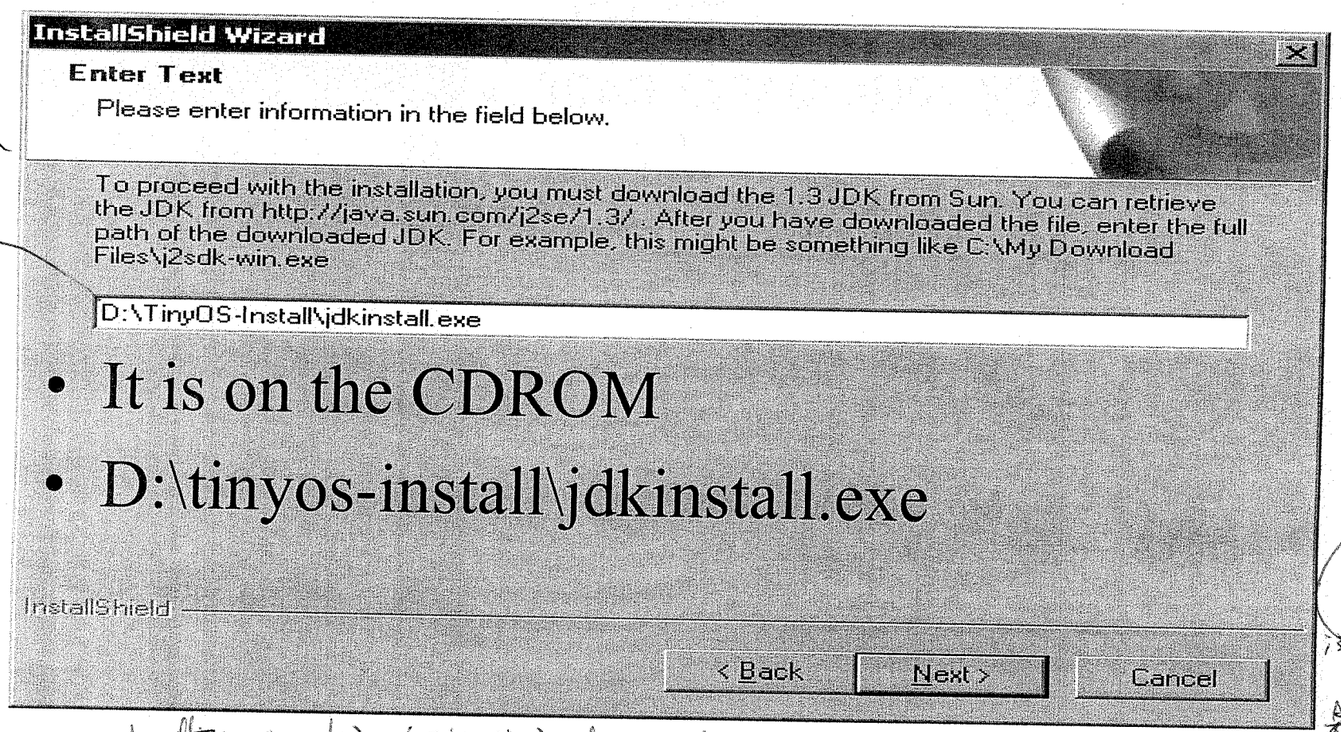


TOS1.0 Baseline Install



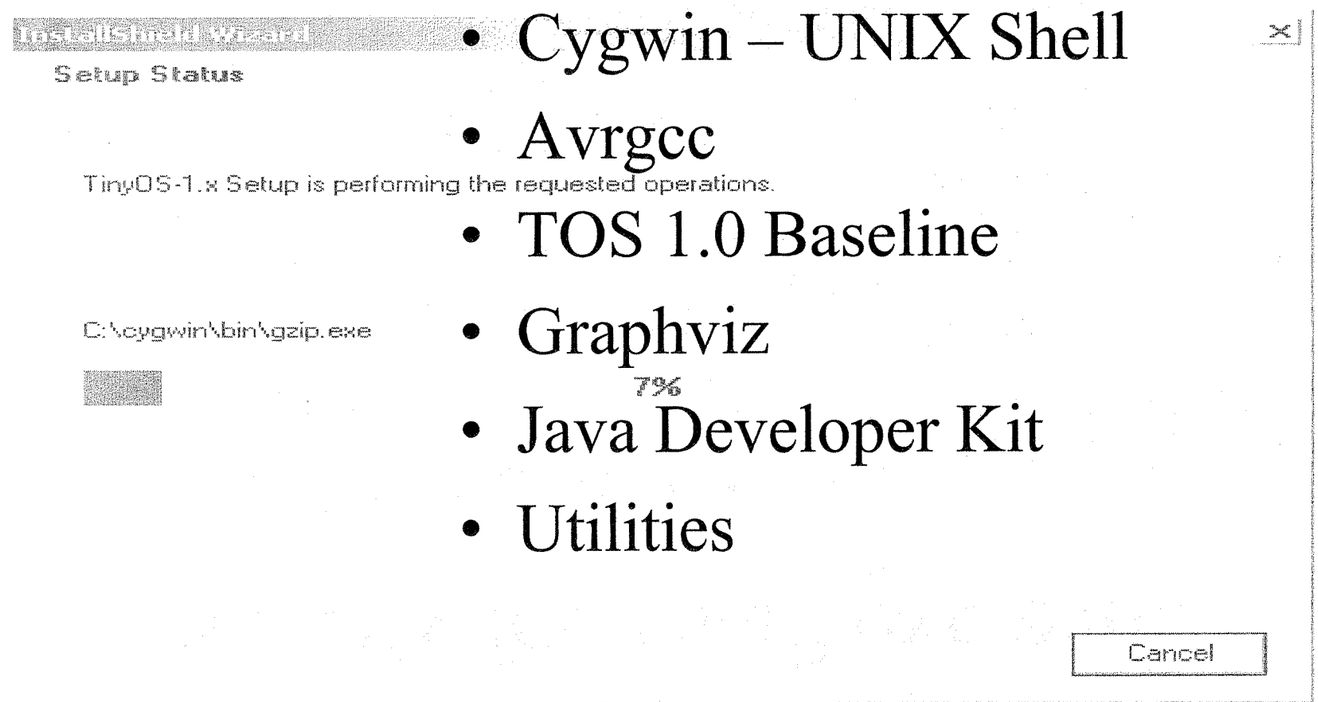
- TOSSetup.exe
- "Prepare to Install" takes a little while…

Specify jdkinstall.exe path



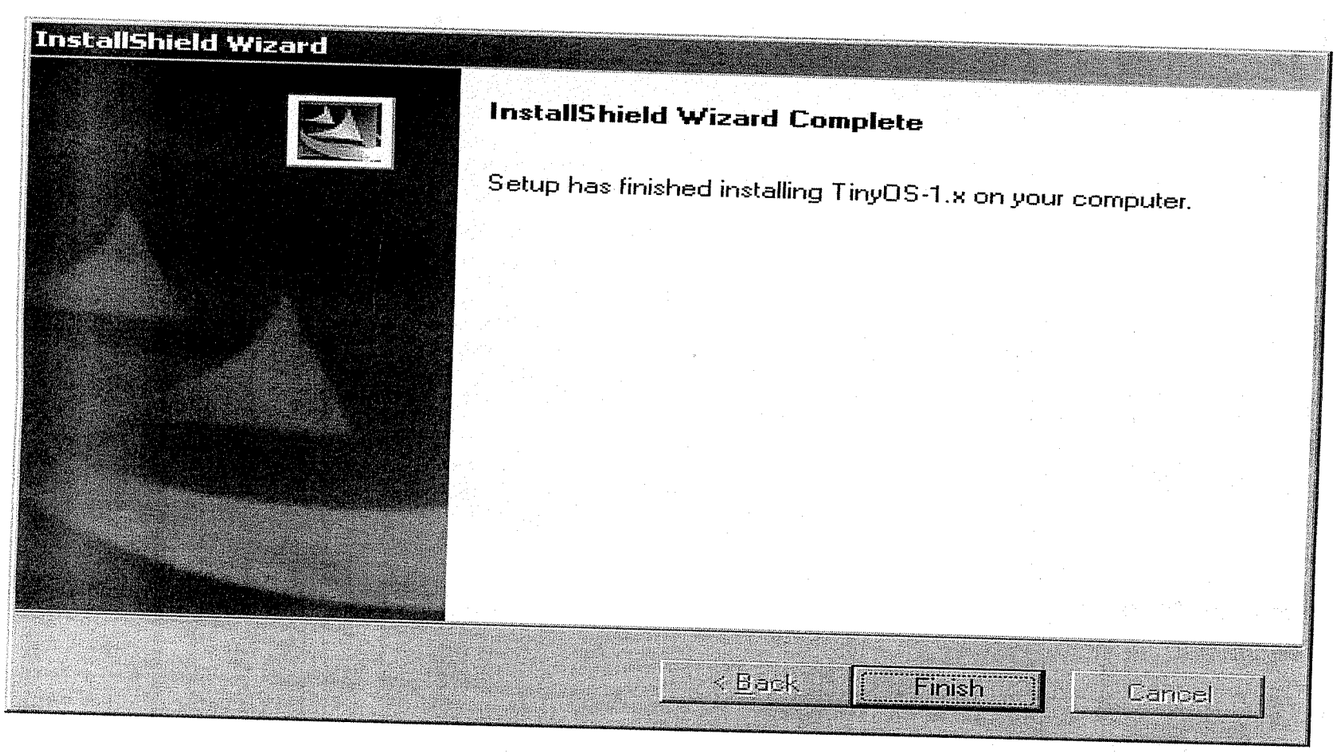
- It is on the CDROM
- D:\tinyos-install\jdkinstall.exe

TOS Install Setup Status



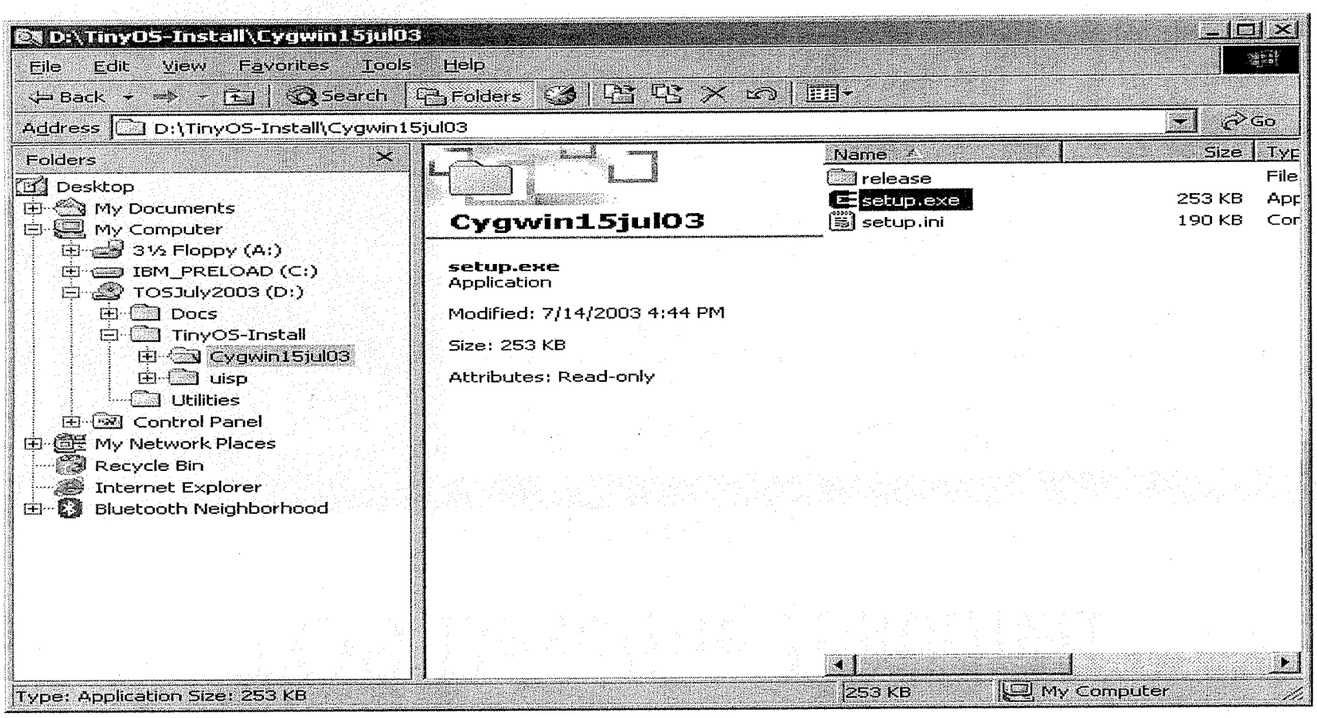
- Cygwin – UNIX Shell
- Avrgcc
- TOS 1.0 Baseline
- Graphviz
- Java Developer Kit
- Utilities

TOS Baseline Installed

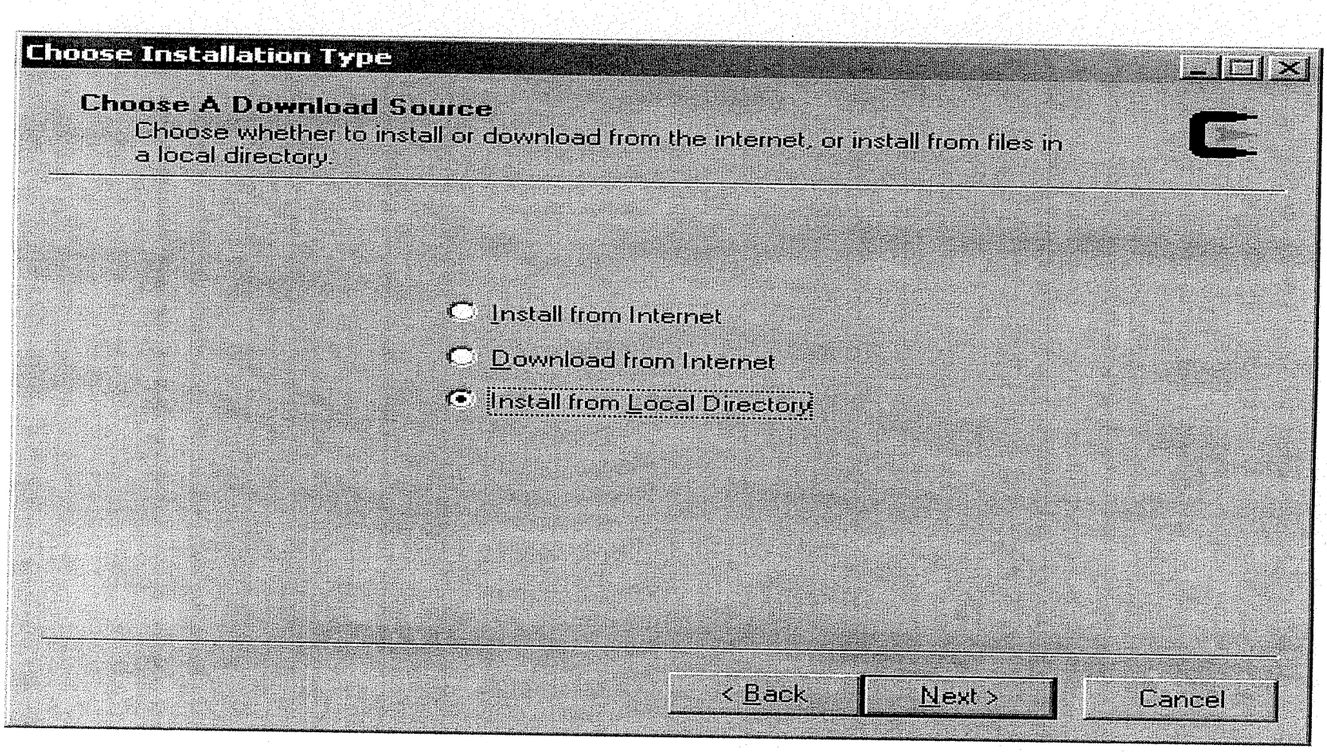


Cygwin Update

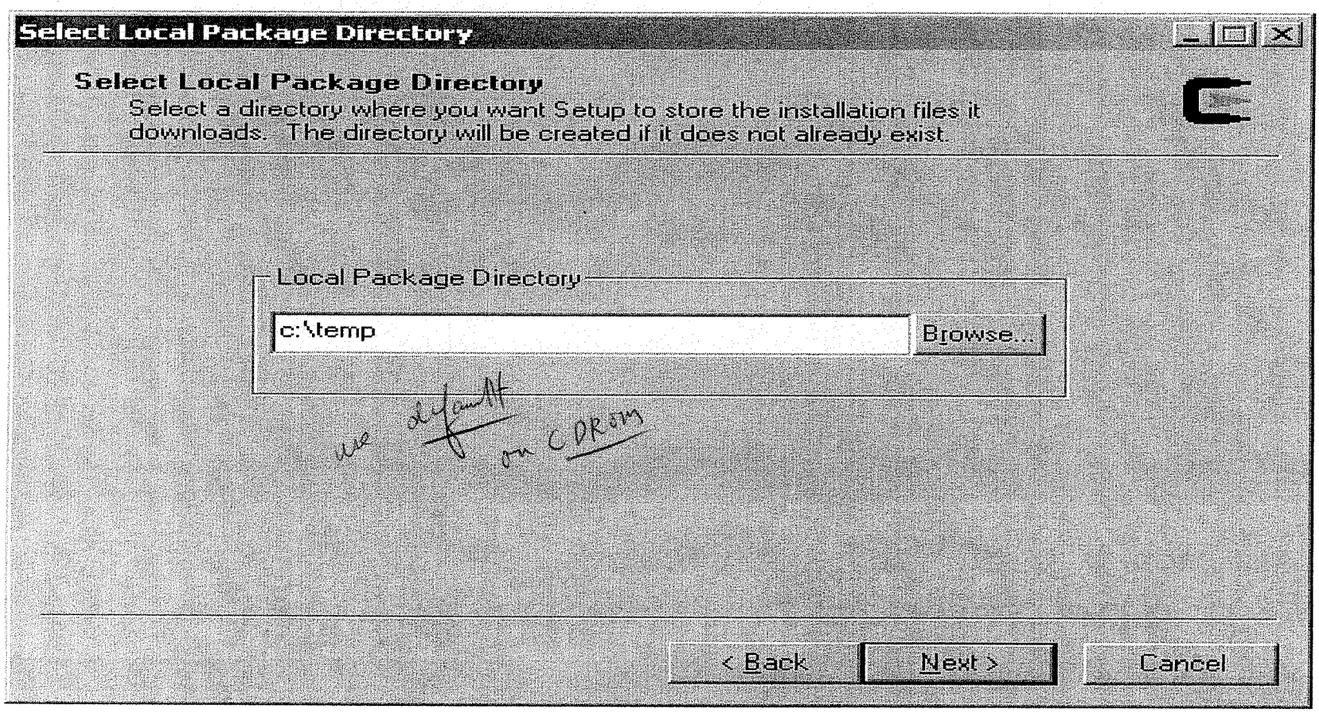
we should already install cygwin. (this is for person already install old version)



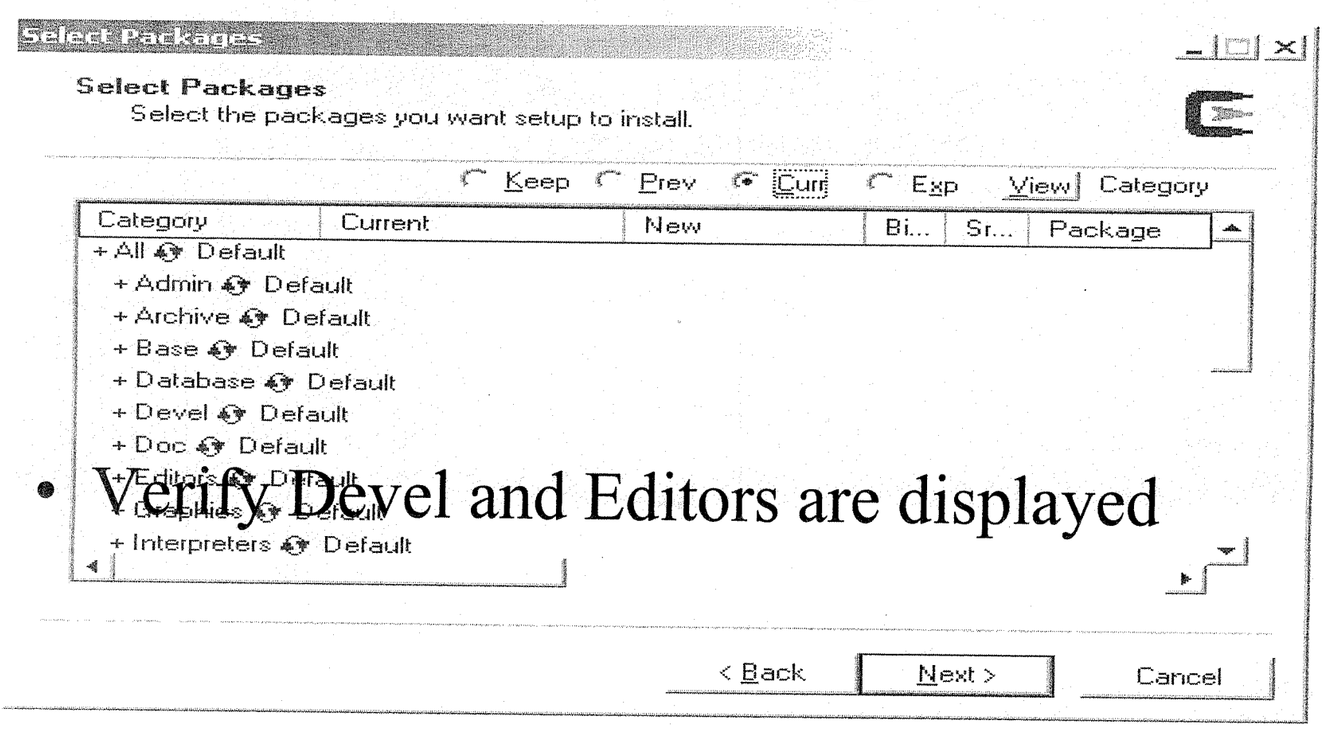
Cygwin Install from Local



Cygwin Local Package Dir

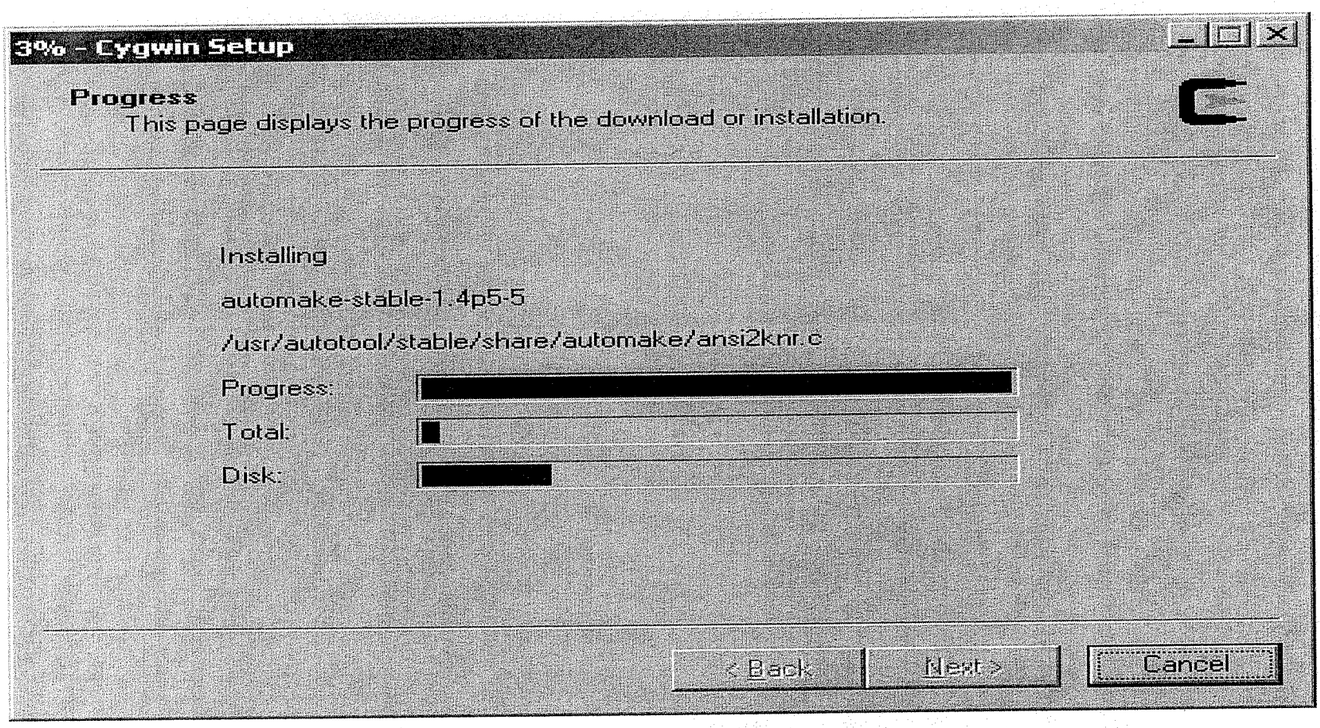


Cygwin Update Select Packages

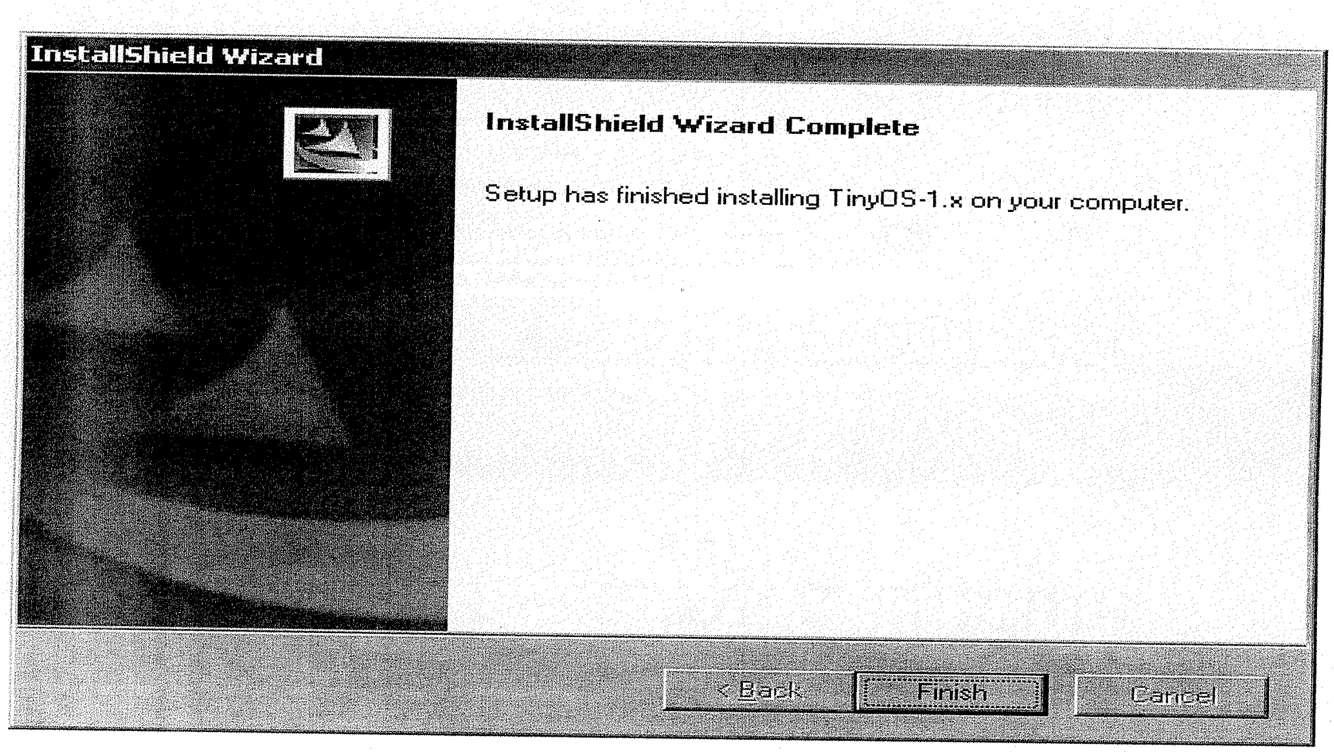


- Verify Devel and Editors are displayed

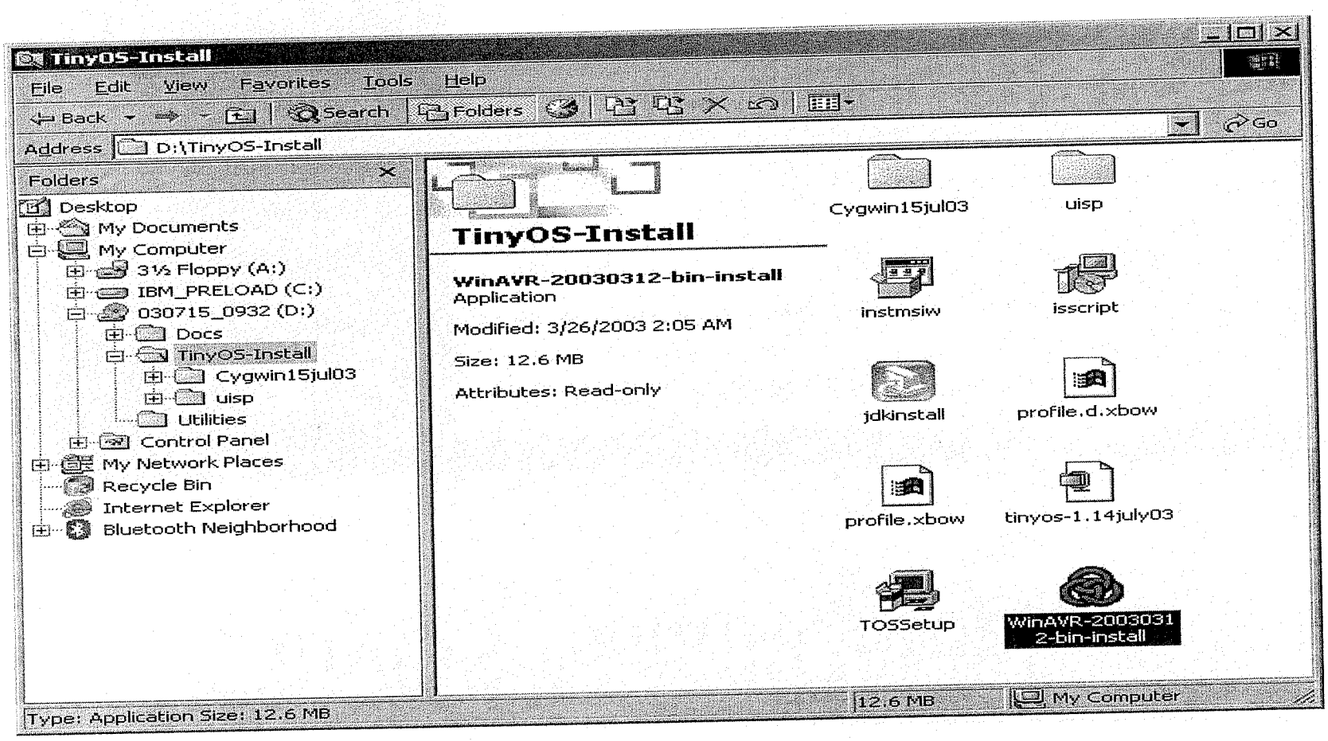
Cygwin Updating



TinyOS Install Complete

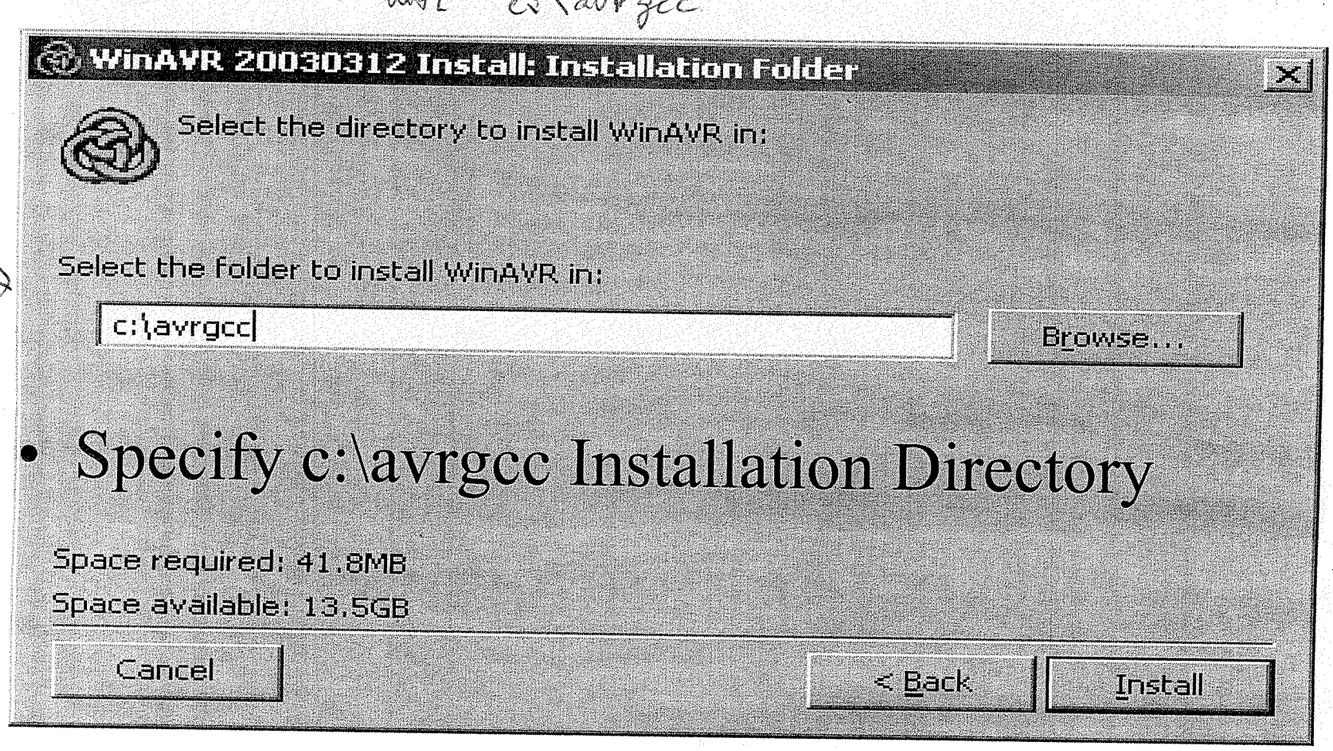


WinAvr – avrgcc Update



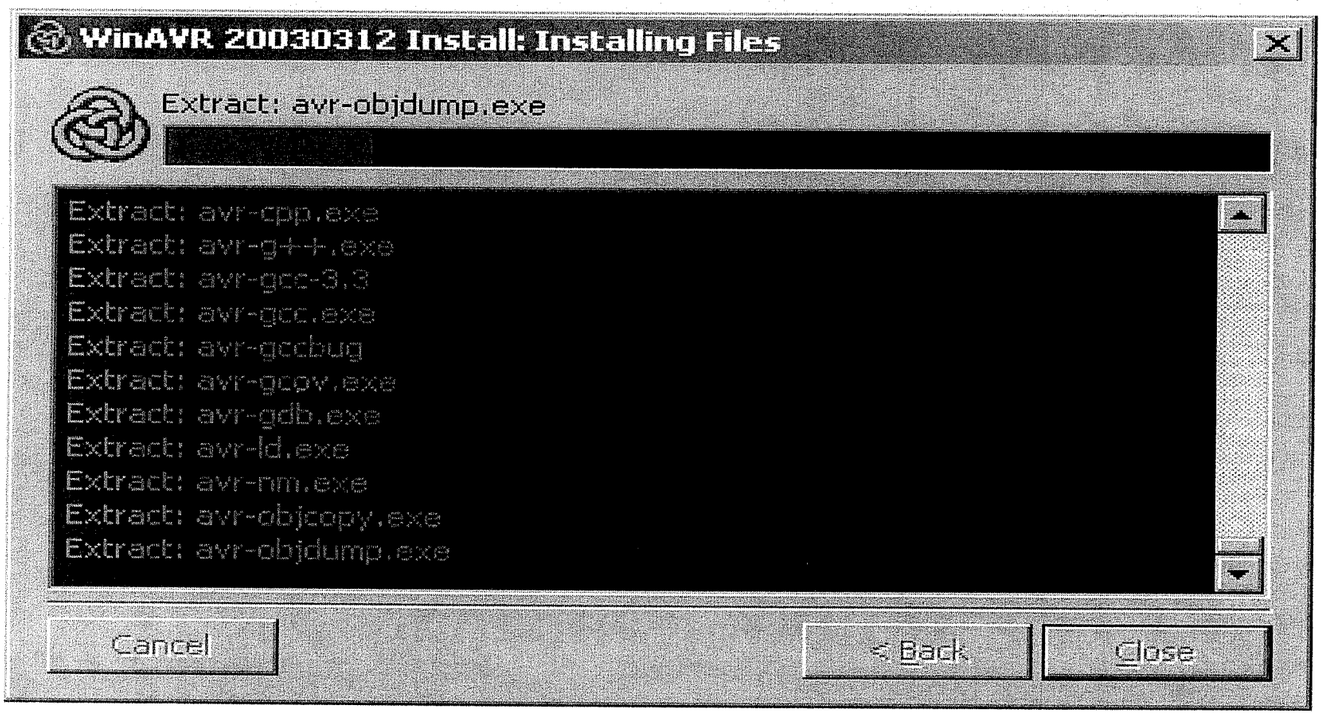
Avrgcc Update

replace c:\winavr with c:\avrgcc



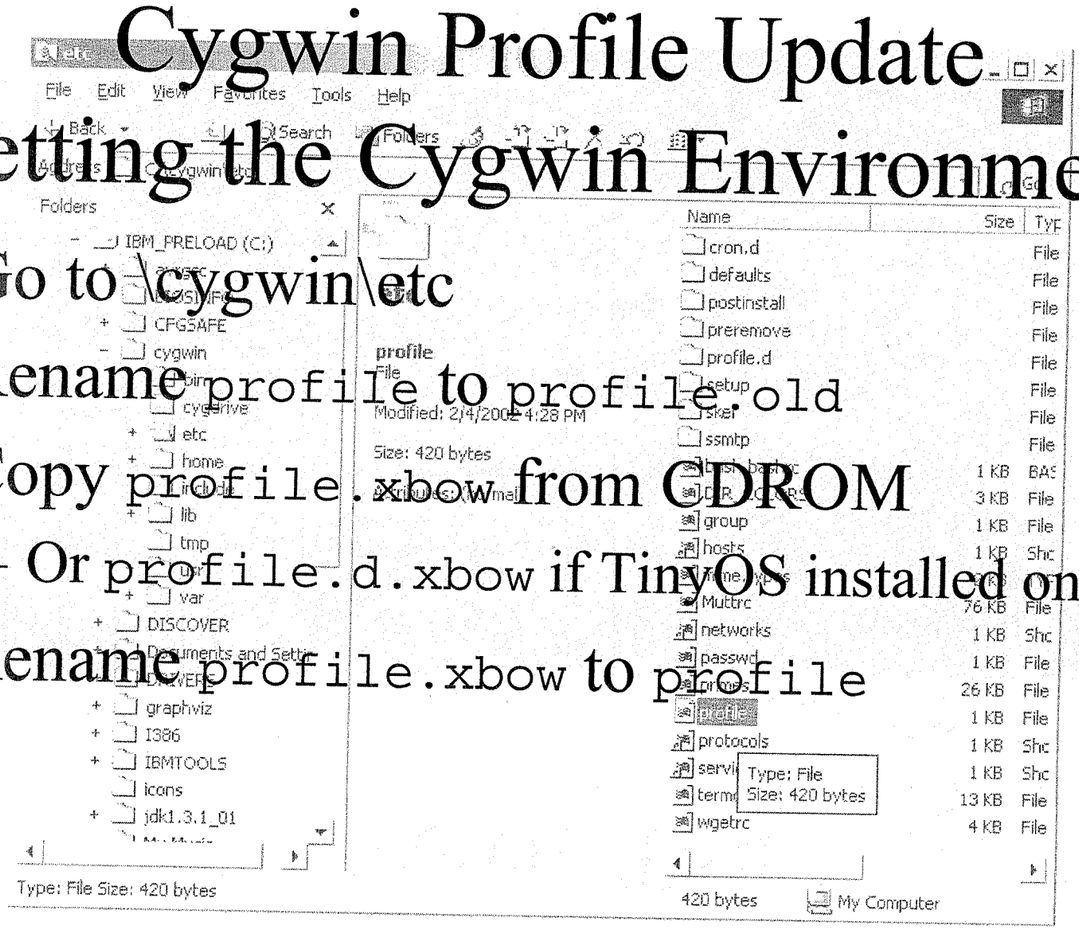
- Specify c:\avrgcc Installation Directory

Avrgcc Update
Installing Files



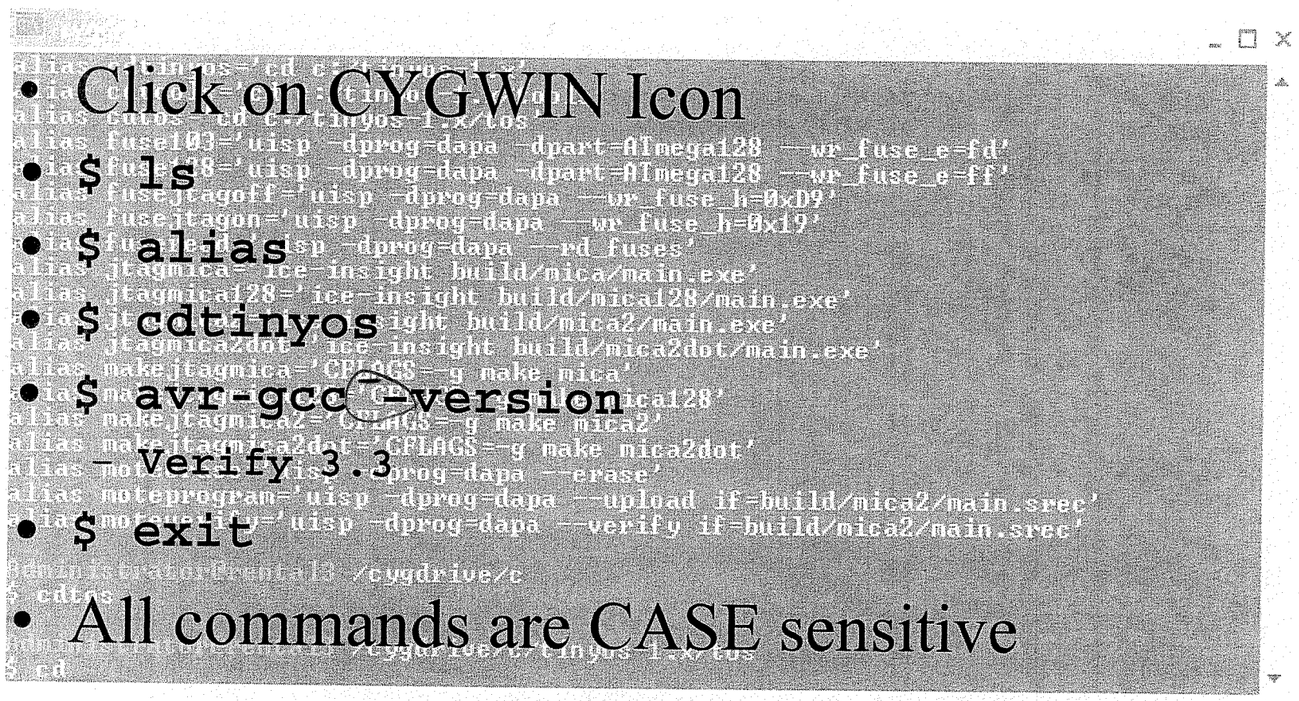
Cygwin Profile Update
Setting the Cygwin Environment

- Go to \cygwin\etc
- Rename `profile` to `profile.old`
- Copy `profile.xbow` from CDROM
  - Or `profile.d.xbow` if TinyOS installed on D:
- Rename `profile.xbow` to `profile`



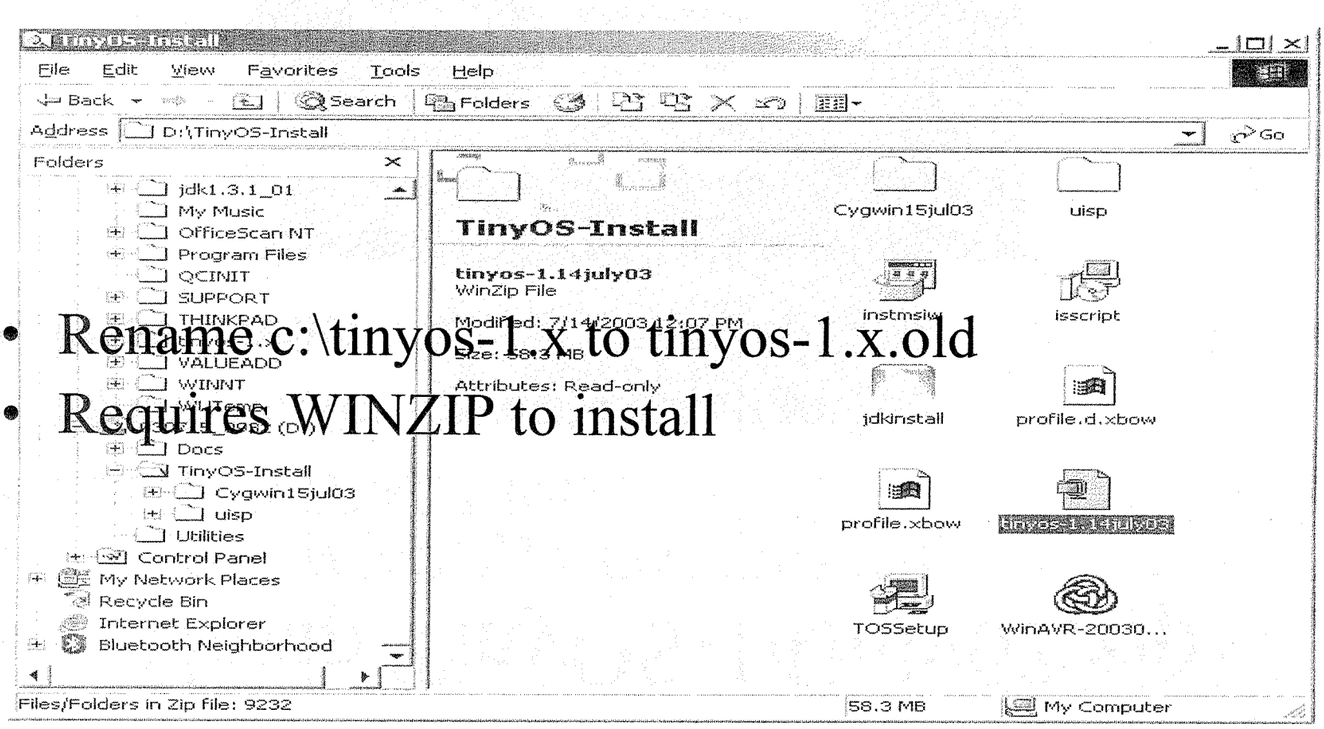
rename aliases in profile from c:\ to d:\

Cygwin Test



- Click on CYGWIN Icon
- $ ls
- $ alias
- $ cdtinyos
- $ avr-gcc –version
  - Verify 3.3
- $ exit
- All commands are CASE sensitive

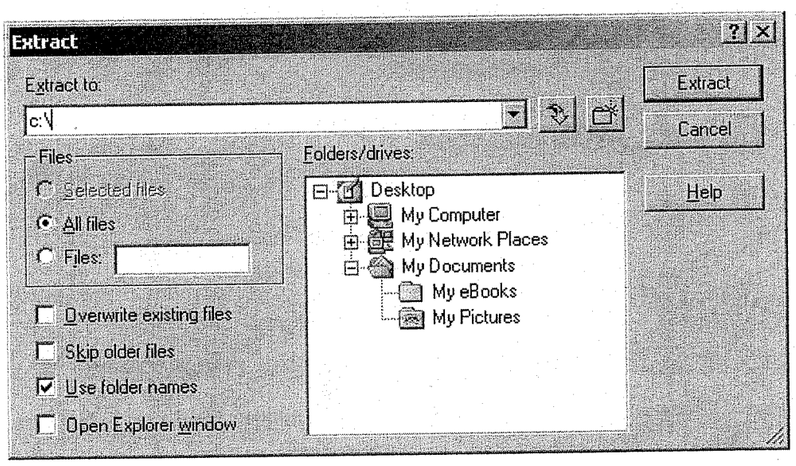
TinyOS 1.1Beta Update



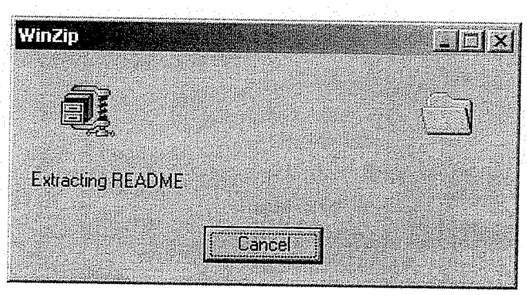
- Rename c:\tinyos-1.x to tinyos-1.x.old
- Requires WINZIP to install

Unzip TinyOS-14July03.zip

- Select EXTRACT Panel
- Specify c:\ (or d:\) as "Extract to:"
- Press Extract



TinyOS 14July03 Update

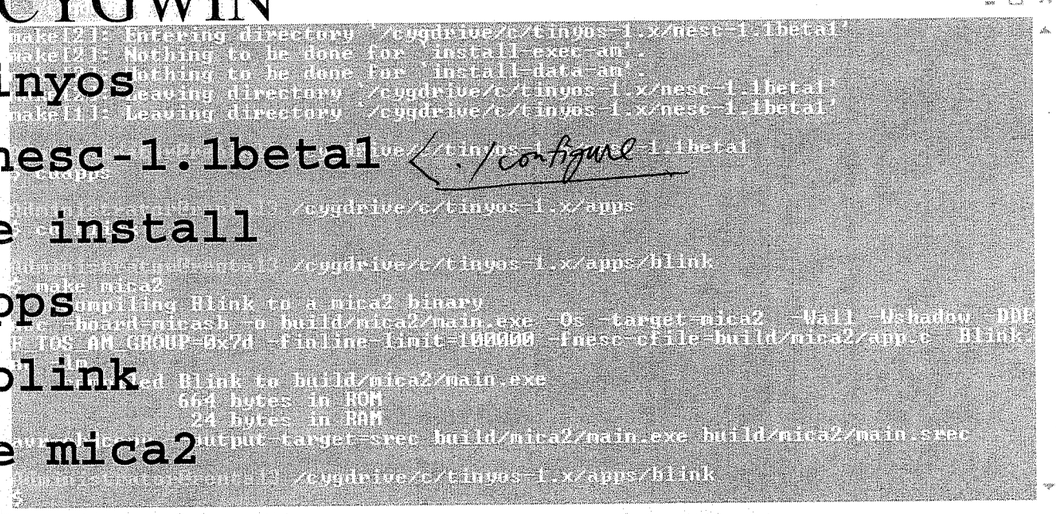


- Close WINZIP when finished

cygwin
setup
tiny-os
Edit
path env.

Nesc-1.1Beta Update

- Open CYGWIN
- $ cdtinyos
- $ cd nesc-1.1beta1
- $ make install
- $ cdapps
- $ cd blink
- $ make mica2
- $ exit

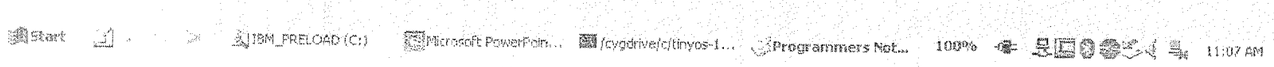


Makerules Update

We will edit make rules in Tinyos-1.x/apps so that all app generated will use specific GroupID

- Group ID
- PROGRAMMER FLAGS
- Use Programmer Notepad (icon on desktop) to edit the following file
  - Tinyos-1.x/apps/makerules

```
# Specify user values in Makelocal to override
the defaults hereifndef DEFAULT_LOCAL_GROUP
DEFAULT_LOCAL_GROUP := 0x7d #USE YOURBADGE ID
#if Thinkpad
PROGRAMMER_EXTRA_FLAGS := -dlpt=3
```



uisp Universal In System Programmer Build

- Open Cygwin
- $ cdtools
- $ cd src/uisp
- $ ./configure
- $ cd src
- $ make
- $ make install
- $ cd ../kernel/win32
- $ make

$ make install

uisp Test

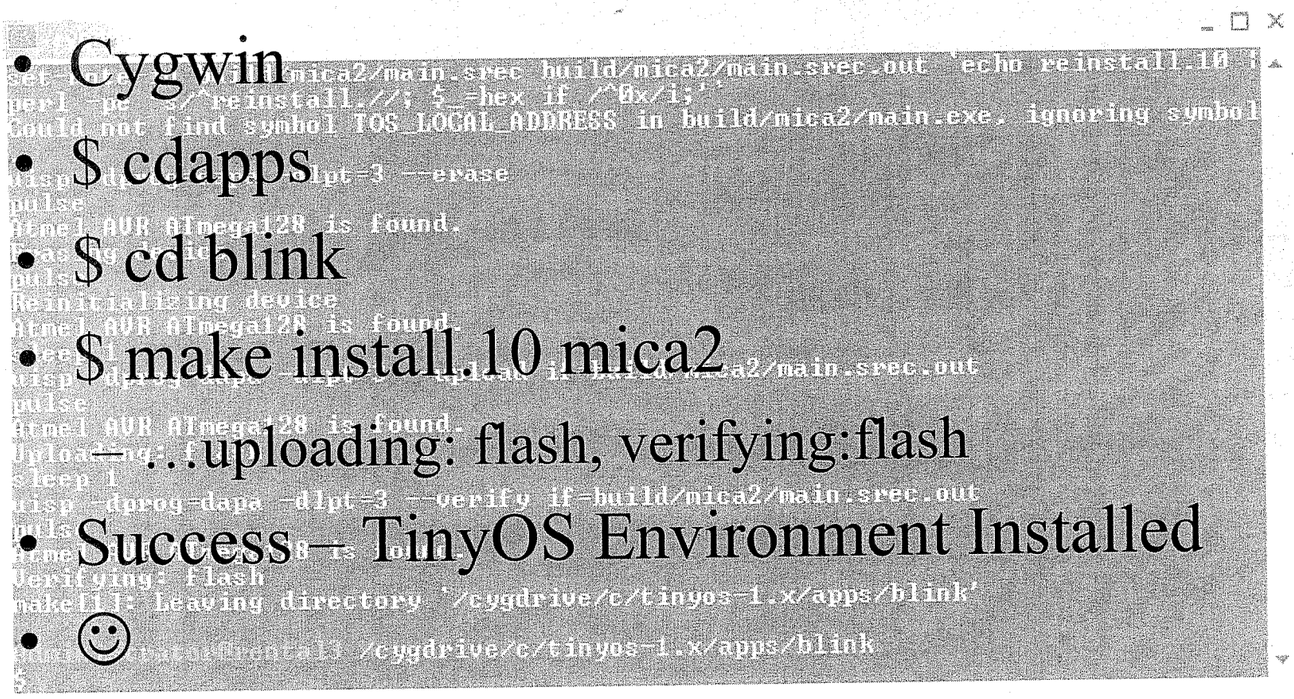
- Plug MIB500 in to LPT Port → we can also use serial port (my laptop does not have parallel port)
- Plug MICA2 in to MIB500
- Plug-in Power
- $ uisp --version
  - Version 20030618
- $ uisp -dprog=dapa --rd_fuses
  - Displays ATMega CPU configuration
  - $ uisp -dprog=dapa -dlpt=3 --rd_fuses (IBM Thinkpad)

Try the USB dock station but its serial port does not work we got an serial USB 5in1 instead)

use -dlpt=1 (since the parallel port is in 1)

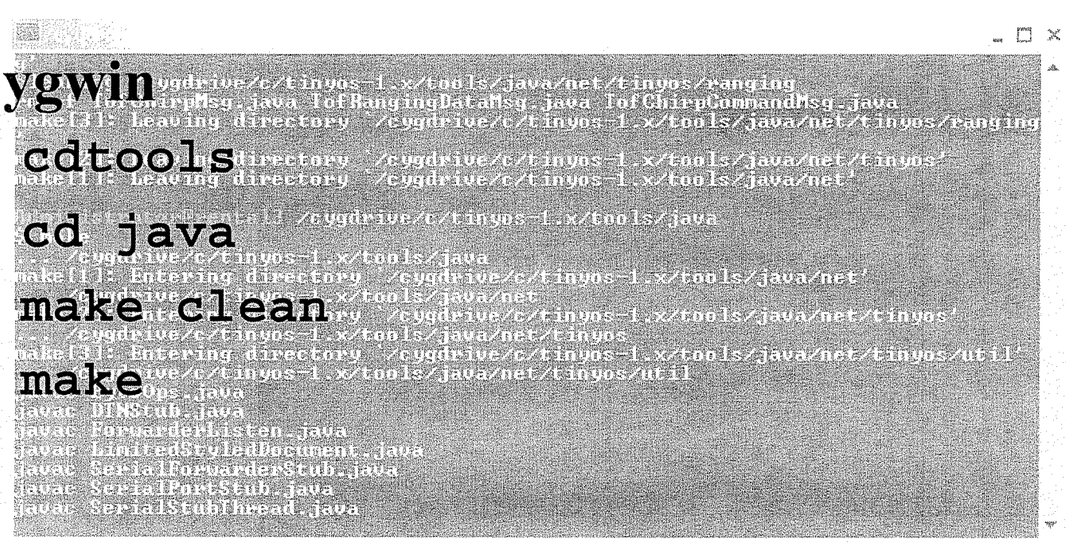
Programming a MICA2

- Cygwin
- $ cdapps
- $ cd blink
- $ make install.10 mica2
- …uploading: flash, verifying:flash
- Success – TinyOS Environment Installed
- ☺



Java Tools Build

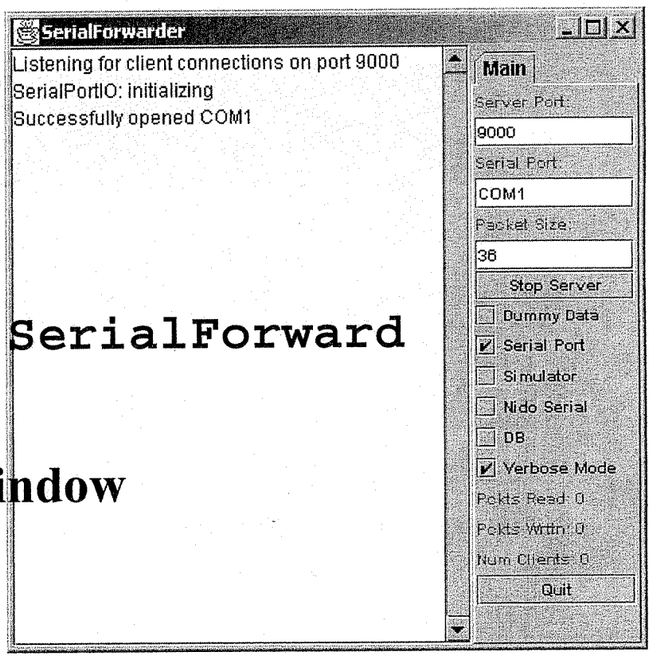
- Cygwin
- $ cdtools
- $ cd java
- $ make clean
- $ make



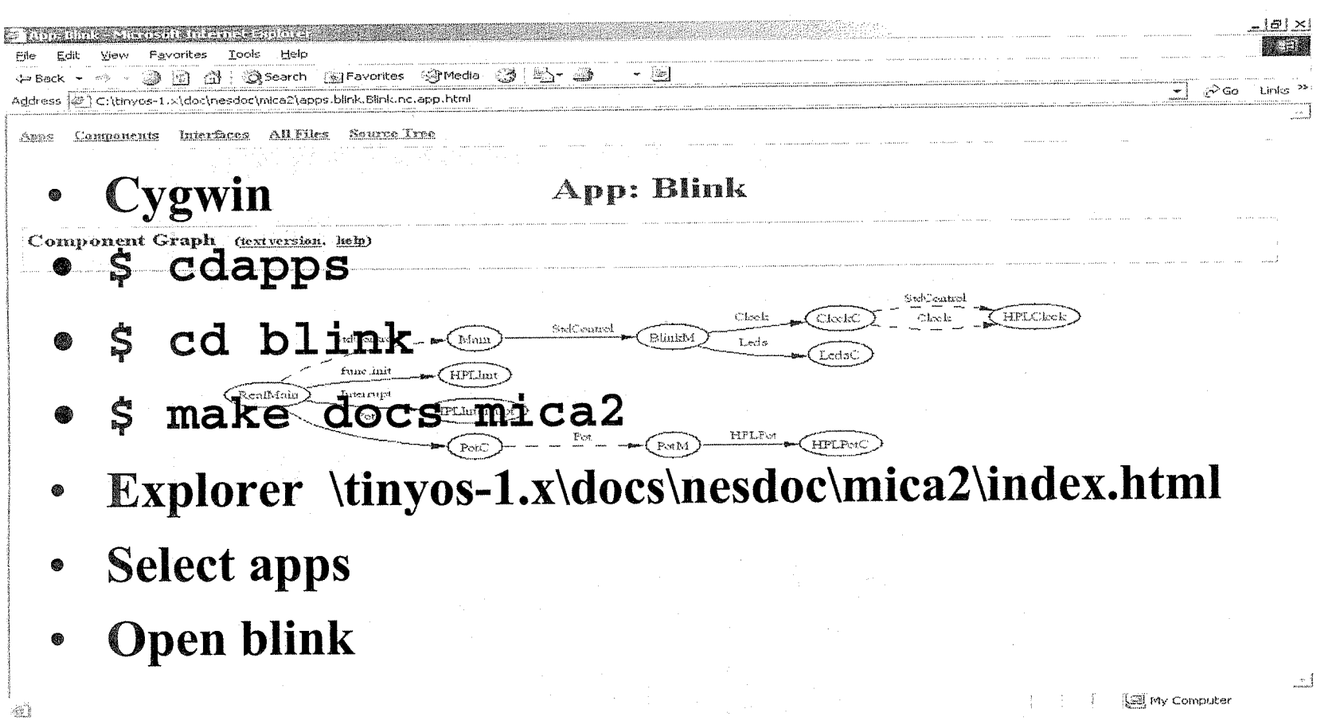
Java Tool Test

- Cygwin
- `$ cdtools`
- `$ cd java`
- `$ java net.tinyos.sf.SerialForward`

- Close the SerialForwarder window



TinyOS Document Generator



- Cygwin
- `$ cdapps`
- `$ cd blink`
- `$ make docs mica2`
- Explorer \tinyos-1.x\docs\nesdoc\mica2\index.html
- Select apps
- Open blink

TinyOS 1.1Beta
Install Complete

- ☺ **Installed**
  - **TinyOS-1.x, Cygwin, avrgcc, uisp, java are running**
- ☹ **Problems**
  - **Notify a Crossbow TOS Trainer**

TOS Development Flow

- Create Application wiring interfaces
- Build application via "make"
  - nesC locates & pre-compiles all required components
  - Avr-gcc gnu compiler compiles/assembles modules for specified platform
  - Avr-ld gnu links into an executable for specific target
- Install executable on Target via programming cable.
  - Or Simulate on PC "make pc"
- Debug Application using LEDs and/or JTAG

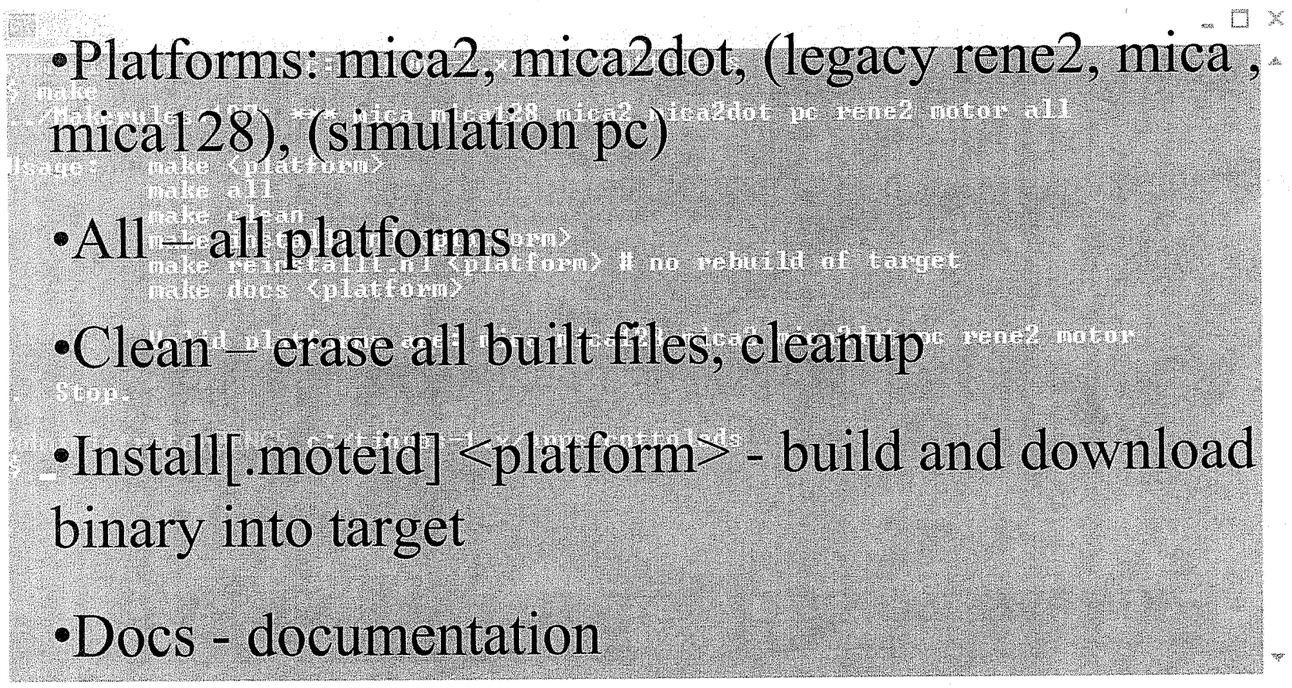
make

- Build is controlled by 2 script files
  - `makefile` in application's directory
    - Application specific options eg SensorBoard
  - `makerules` in ../apps directory
    - System specific options
    - Platform options
    - Debugging options
- Details will be covered in a later Session

Target IDs

- Group ID
  - Collection of Motes form a Group
    - Messages are local to a Group
  - GroupID is SET in makerules
    - Edit your tinyos/apps/makerules NOW using ID# on Name Tag
- Mote ID
  - Specific Mote with in a Group
  - MoteID set during install of exe into target
    - make install.[moteid] mica2
    - Use MoteID = 10 (decimal)

TinyOS `make` options



- Platforms: mica2, mica2dot, (legacy rene2, mica , mica128), (simulation pc)
- All – all platforms
- Clean – erase all built files, cleanup
- Install[.moteid] <platform> - build and download binary into target
- Docs - documentation

TinyOS Directory Structure

Apps	Standard TOS Applications and Test Programs
Contrib	User contributions
Doc	Documentation & Tutorials
Nesc	NesC pre-compiler source
Tools	Development utilities and programs
Tos	Tiny OS modules and interfaces

TOS Sub-Directory

Interfaces	TOS component interfaces
Lib	Libraries – incl TinyDB, Route
Platform	Mote Platform specific drivers
Sensorboards	Sensor Board drivers
System	Mote System drivers – EEPROM, UART
Types	Special type definitions

MICA2 Platform Topics

- The MICA2 uses an Atmel ATMega128
- ATMega128 Operating Modes controlled by "fuses"
  - CPU clock source, Debug, Legacy Modes
- Corruption/Inadvertent fuse changes can stop the ATMega CPU functioning
- Recover by knowing how to reset fuses

ATMega Fuses

- AT103 Compatibility Mode [OFF]
  - Disable Legacy (MICA). Note older MICAs were shipped with this fuse ENABLED
- JTAG [OFF]
- ON-CHIP Debug [OFF]
- BROWN-OUT Detect [OFF]

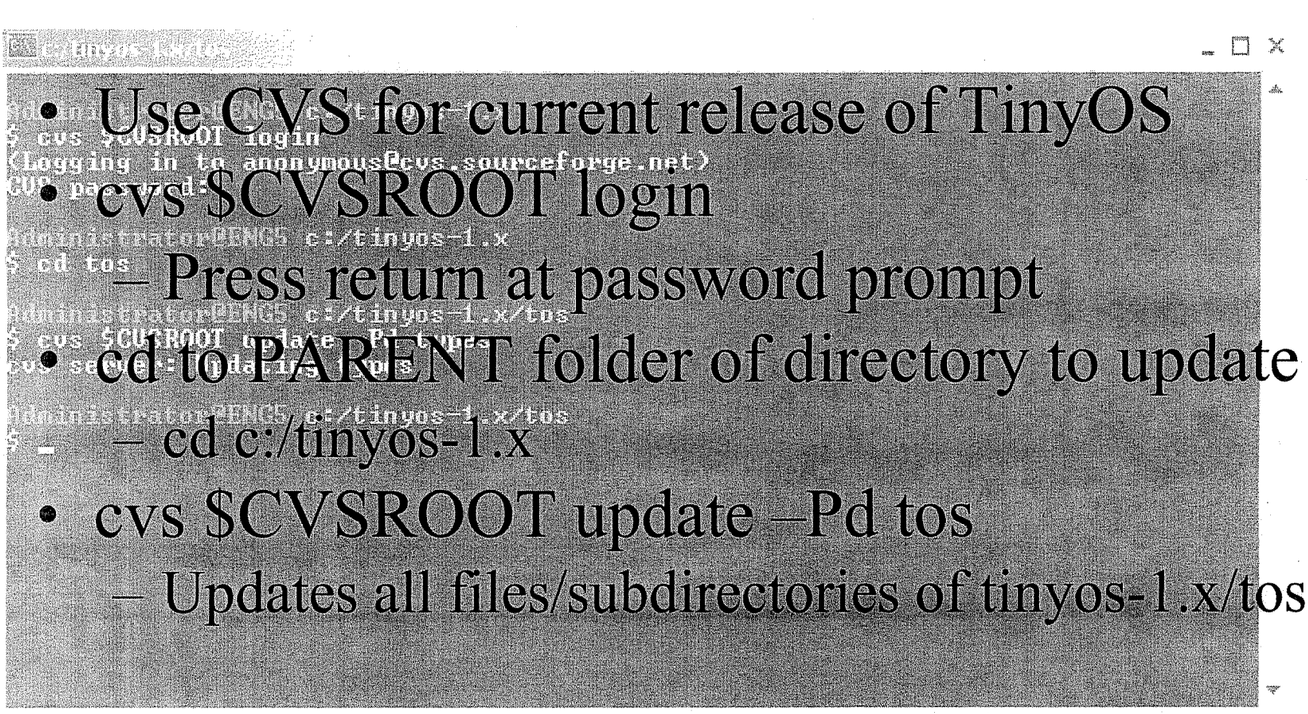
Standard ATMega-128 Fuse

- ATMega 128 Native (not 103 Compatible)
- JTAG Off (ADC Channels 4 thru 7 work)
- To Set fuses
  - `uisp -dprog=dapa --wr_fuse_h=0xd9`
  - `uisp -dprog=dapa --wr_fuse_e=0xff`
- Alias Scripts
  - `$ fusejtagon & $ fusejtagoff`
  - `$ fuse128 & $ fuse103`

JTAG ATMega-128 Fuse

- JTAG & ON-Chip Debug Enabled
- ADC Channels 4, 5, 6, & 7 reserved for JTAG
- Low Power SLEEP mode draws 2mA!
- To Set fuses for JTAG debug
  - `uisp -dprog=dapa --wr_fuse_h=0x19`
  - `uisp -dprog=dapa --wr_fuse_e=0xff`

Retrieving Current TOS Versions Using CVS



- Use CVS for current release of TinyOS
- cvs $CVSROOT login
  - Press return at password prompt
- cd to PARENT folder of directory to update
  - cd c:/tinyos-1.x
- cvs $CVSROOT update –Pd tos
  - Updates all files/subdirectories of tinyos-1.x/tos

Conclusion

- Installation Complete
- Review
  - Tool Sequence
  - Tool Installation for MS-Windows
  - Tool Description

Introduction to TinyOS and nesC Programming

- TinyOS Kernel Design and Implementation
- nesC Software Concepts and Basic Syntax
- nesC Code Lab
- TinyOS Packet Networking and PC Base Station Lab

Bill Maurer
maurer@dsplabs.com
DSP Labs, Livermore Ca., USA

The Advanced Signal Processing Company

TinyOS Design Goals

- Support Networked Embedded Systems
  - asleep but remain vigilant to stimuli
  - bursts of events and operations
- Support Mica Hardware
  - power, sensing, computation, communication
- Support Technological Advances
  - keep scaling down
  - smaller, cheaper, lower power

TinyOS Design Options

- Can't Use Existing RTOS's
  - Microkernel Architecture
    - VxWorks, QNX, WinCE, PalmOS Linux
  - Execution Similar to Desktop Systems
    - PDA's, Cell Phones, Embedded PC's
  - More Than a Order of Magnitude Too Heavy & Slow
  - Energy Hog

{ Network stack not suitable)

TinyOS Design Conclusion

- **Similar** to Building Networking Interfaces
  - Data Driven Execution
  - Manage Large # of Concurrent Data Flows
  - Manage Large # of Outstanding Events
- **Add:** Managing Application Data Processing
- Conclusion: Need a Multi Threading Engine
  - Extremely Efficient
  - Extremely Simple

The Advanced Signal Processing Company

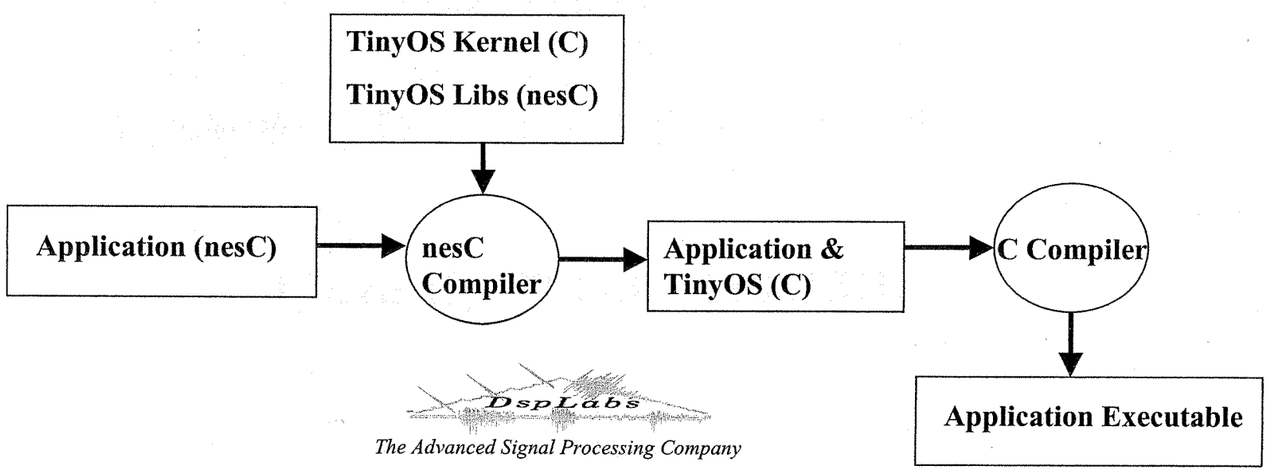
TinyOS Kernel Design

- TinyOS Kernel: 2 Level Scheduling Structure
  - Events
    - Small Amount of Processing
    - E.g. Timer, ADC Interrupts
    - Can Interrupt Longer Running Tasks
  - Tasks
    - Not Time Critical
    - Tasks - Larger Amount of Processing
    - E.g. Computing an Average on an Array
    - Run to Completion WRT other Tasks
      - Implies Only Need a Single Stack

The Advanced Signal Processing Company

TinyOS Applications Under The Hood

- Application is created in the nesC Language
  - Details of nesC Forthcoming
- nesC Programming Language Supports the TinyOS Kernel Design (Events and Tasks)



DspLabs
The Advanced Signal Processing Company

The TinyOS Kernel Under The Hood (nesC Compiler C Code Output)

```
int  main(void) {
  RealMain$hardwareInit();
  TOSH_sched_init();

  RealMain$StdControl$init();
  RealMain$StdControl$start();
  RealMain$Interrupt$enable();

  while (1) {
    TOSH_run_task();
  }
}

static inline void TOSH_run_task(void) {
  while (TOSH_run_next_task())
    ;
  TOSH_sleep();
  TOSH_wait();
}
```

Hardware and Kernel Initialization

Application Initialization

Infinite Loop

```
bool TOSH_run_next_task(void) {
  uint8_t old_full;
  void (*func)(void );
  if (TOSH_sched_full == TOSH_sched_free) {
    return 0;
  }
  else {
    old_full = TOSH_sched_full;
    TOSH_sched_full++;
    TOSH_sched_full &= TOSH_TASK_BITMASK;
    func = TOSH_queue[(int )old_full].tp;
    TOSH_queue[(int )old_full].tp = 0;
    func();
    return 1;
  }
}
```

1. First Run All Tasks in the Task Queue (Strictly a FIFO)

2. Then Sleep (In Low Power Mode)

3. And Wait for an Interrupt

Task Runs To Completion (But May Be Interrupted By An Event)

DspLabs
The Advanced Signal Processing Company

Overhead of TinyOS Primitive Operations

Operation	Cost(cycles)	Time(uSecs)	Normalized to Byte Copy
Byte Copy	8	2	1
Signal an Event	10	2.5	1.25
Call a Command	10	2.5	1.25
Schedule a Task	46	11.5	6
Context Switch	51	12.75	6
Hardware Interrupt (hw)	9	2.25	1
Hardware Interrupt (sw)	**71**	**17.75**	**9**

Code and Data Size of the TinyOS kernel

	Code Size(bytes)	Data Size(bytes)
Processor Init	172	30
Scheduler	178	16
C runtime	82	0
	432	**46**

TinyOS/nesC Application Notes

- Everything is Static
  - No Dynamic Memory (no malloc)
  - No Function Pointers
  - No Heap
- nesC Compiler Analysis
  - Data Race Conditions
  - Function Inlining
  - Development Made Easier
  - Robustness Improved

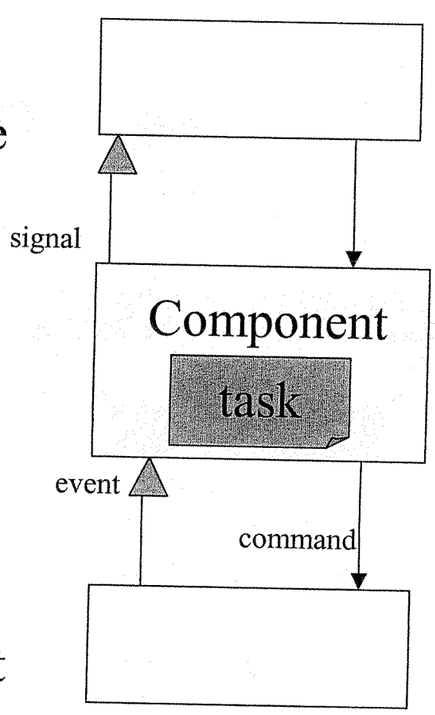
Application Memory Map

- Text/code - Executable Code
  - In the 128K Program Flash
- data – Program Constants
  - In the 128K Program Flash
- bss - Variables
  - In the 4K SRAM
- Free Space - Fixed (No Dynamic Memory)
- stack - Grows Down in the Free Space

DspLabs
The Advanced Signal Processing Company

TinyOS Concepts Embodied by nesC – Tasks, Events, Commands

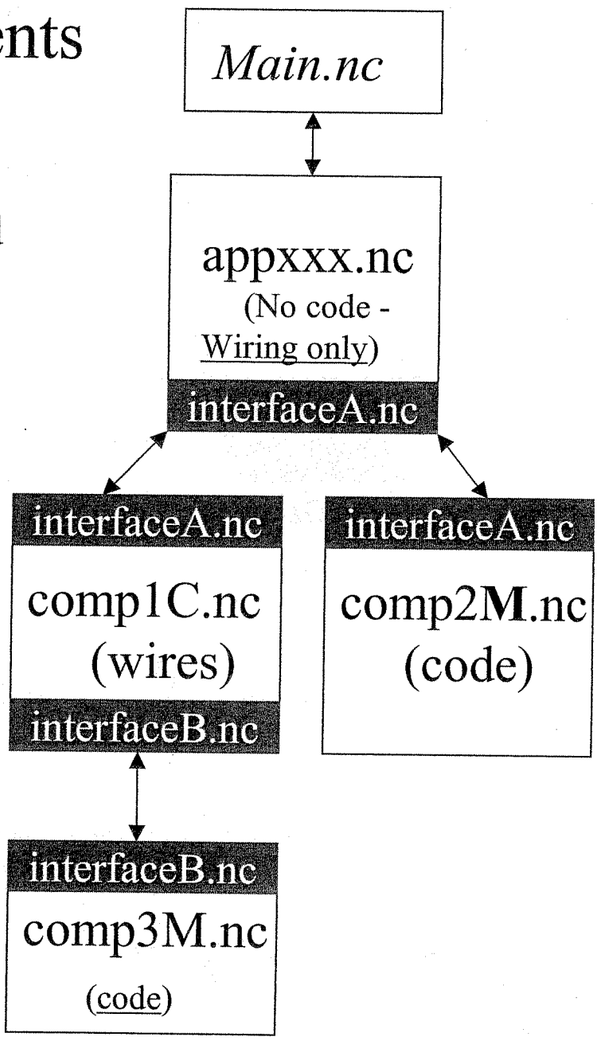
- **Tasks**
  - Background computation, non-time critical
- **Events**
  - Time critical
  - External Interrupts
  - Originator gives a '**Signal**'
  - Receiver gets/accepts an '**Event**'
- **Command**
  - Function call to another Component
  - Cannot **Signal**



DspLabs
The Advanced Signal Processing Company

Concepts of SW Components

- **Interfaces** (xxx.nc)
  - Specifies functionality to outside world
  - Tell outside world
    - what commands can be called
    - what events need handling
- Software Components
  - **Module** (xxxM.nc)
    - Code file, code implementation
    - It codes the **Interface**
  - **Configuration** (xxxC.nc)
    - Linking/wiring of components
    - When top level app, drop C from filename xxx.nc
    - optional **Module**



TinyOS app Blink – Blinks the Red LED

BlinkM.nc
Blink.nc

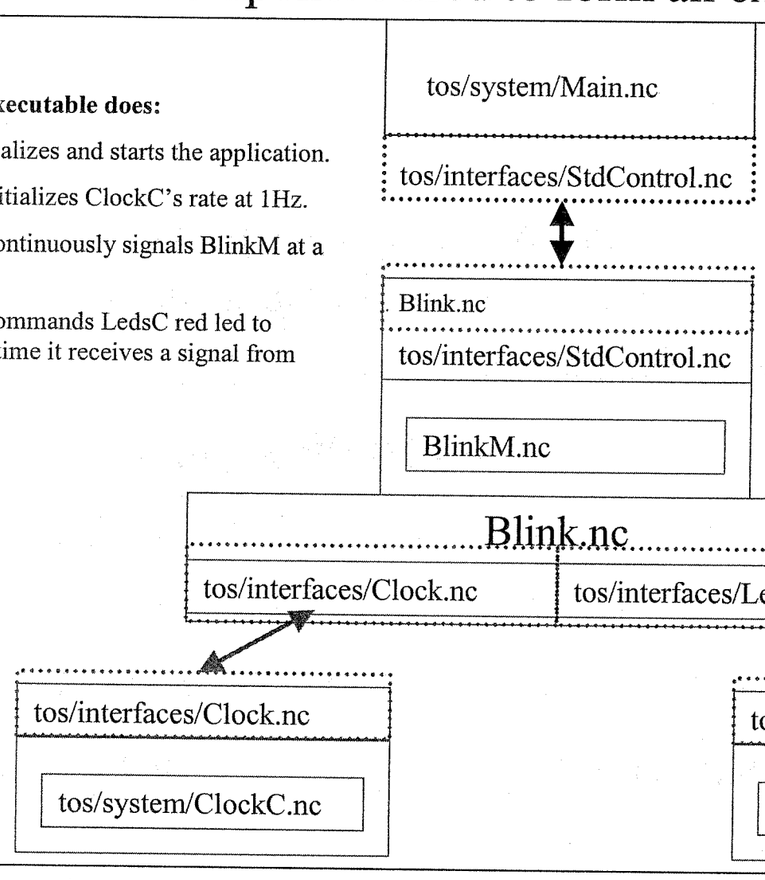
DspLabs
The Advanced Signal Processing Company

Ad infinitum...

Blink.nc Application - A top level **configuration** SW component used to form an executable

What the executable does:

1. Main initializes and starts the application.
2. BlinkM initializes ClockC's rate at 1Hz.
3. ClockC continuously signals BlinkM at a rate of 1 Hz.
4. BlinkM commands LedsC red led to toggle each time it receives a signal from ClockC.



Note: The StdControl interface is similar to state machines (init, start, stop); used extensively throughout TinyOS apps & libs

DspLabs
The Advanced Signal Processing Company

Blink Configuration

- Blink.nc Wires Together **Components**
  - Unused System **Components** Excluded
  - A Directed Wire (Arrow '->') Connects Components
    - Only 2 **Component**s at a time
    - Connection is Across an **Interface**
    - Component that uses the interface '->' component that provides the interface

The Blink Configuration - Our First Application

A nesC **application** consists of one or more **components**, **wired together** to form a runnable executable

```
configuration Blink {
}
implementation {
components Main, BlinkM, ClockC, LedsC;

Main.StdControl -> BlinkM.StdControl;
BlinkM.Clock -> ClockC.Clock;
BlinkM.Leds -> LedsC.Leds;
}
```

A **configuration** is a component that "wires" other components together. The idea here is that one can build an application as a set of components, wiring together those components by providing a configuration. Every nesC application has a single **top-level configuration** that specifies the set of components in the application and how they connect one another.

Why isn't this BlinkC.nc ?
- At the top level, it is implied

tos/interfaces/Clock.nc Interface

```
interface Clock {
command result_t setRate(char interval, char scale);
event result_t fire();
}
```

An **interface** is used to provide an abstract definition of the interaction of two **components** (the **user** and the **provider**). Similar to languages such as Java, an interface contains no actual code or wiring; it simply declares a set of functions:

the **provider** must implement all **commands** and signal all **events**

the **user** must implement all **events**

In this way it is different than Java interfaces. A Java interface specifies one direction of call, while a nesC interface specify two directions. For a component to **call** the **commands** in an interface, it must implement the **events** of that interface A single component may use or provide multiple interfaces and multiple instances of the same interface.

BlinkM.nc Implementation

A **module** is a nesC component consisting of application code in a C-like syntax.

```
module BlinkM {
 provides {
  interface StdControl;
 }
 uses {
  interface Clock;
  interface Leds;
 }
}
implementation {
command result_t StdControl.init() {
  call Leds.init();
  return SUCCESS;
 }
command result_t StdControl.start() {
  return call Clock.setRate(TOS_I1PS, TOS_S1PS);
 }
command result_t StdControl.stop() {
  return call Clock.setRate(TOS_I0PS, TOS_S0PS);
 }
```

The nesC compiler generates C code that consists of the hardware clock interrupt handler invoking a function called: BlinkM$Clock$fire():

```
result_t
BlinkM$Clock$fire(void)
{
 BlinkM$Leds$redToggle();
 return SUCCESS;
}
```

```
event result_t Clock.fire()
 {
  call Leds.redToggle();
  return SUCCESS;
 }
}
```

Component Implementation

- **Modules**
  - Contain "C" Like Code
  - Must implement the all the *'provides'* interfaces
    - Commands
    - Signals

```
provides {
  interface StdControl;
}
```

    - Module must implement StdControl's commands & (any signals, note: StdControl has no Events to be signal'ed)
  - Must deal with '*uses*' interfaces
    - Events must be handled
    - Commands invoked as required

```
uses {
 interface Clock;
 interface Leds;
}
```

    - Module must implement Clock's fire event, note: Leds has no Event's

The Advanced Signal Processing Company

nesC Command and Event Syntax

- For a **Command** Or **Event** *F* for **Interface** *I*
  - Definition and Invocation Name is *I.F*
- **Command** Calls prefixed with "**call**"
- **Event** signal prefixed with "**signal**"
- Definition of a **command** or **event** prefixed with "**command**" or "**event**" and return type is **result_t**

The Advanced Signal Processing Company

BlinkM.nc Implementation

A **module** is a nesC component consisting of application code in a C-like syntax.

```
module BlinkM {
 provides {
   interface StdControl;
 }
 uses {
   interface Clock;
   interface Leds;
 }
}
implementation {
command result_t StdControl.init() {
   call Leds.init();
   return SUCCESS;
 }
command result_t StdControl.start() {
   return call Clock.setRate(TOS_I1PS, TOS_S1PS);
 }
command result_t StdControl.stop() {
   return call Clock.setRate(TOS_I0PS, TOS_S0PS);
 }
```

The nesC compiler generates C code that consists of the hardware clock interrupt handler invoking a function called: BlinkM$Clock$fire():

```
result_t
BlinkM$Clock$fire(void)
{
 BlinkM$Leds$redToggle();
 return SUCCESS;
}
```

```
event result_t Clock.fire()
 {
   call Leds.redToggle();
   return SUCCESS;
 }
}
```

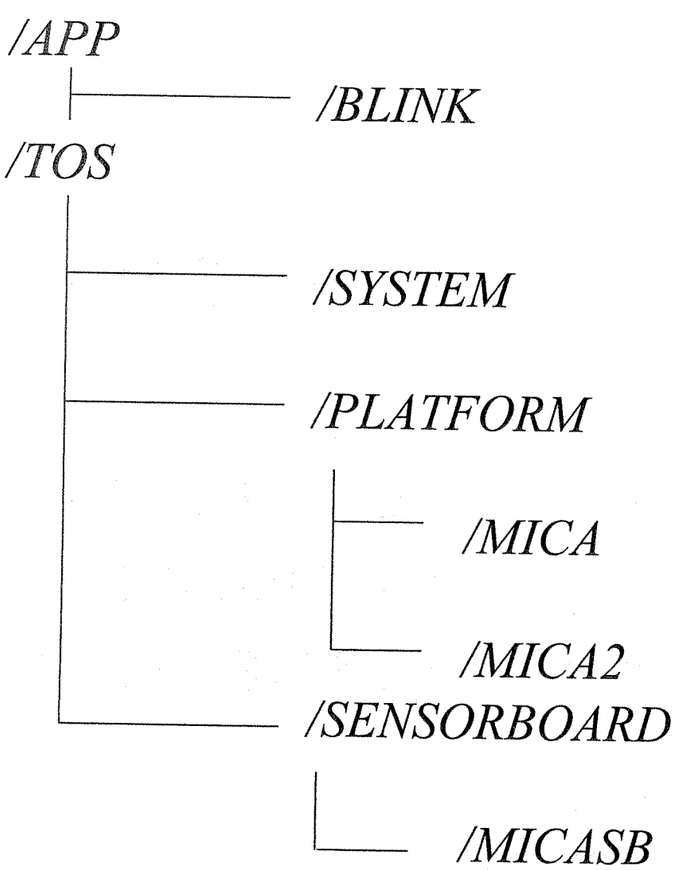

TinyOS/nesC General Notes

- Equivalent Wiring Syntax
  - BlinkM.Clock -> ClockC.Clock;
  - BlinkM.Clock -> ClockC;
  - BlinkM -> ClockC.Clock;
  - Clock.Clock <- BlinkM.Clock;
  - Clock <- BlinkM.Clock;
  - Clock.Clock <- BlinkM;
- Most Components Are Software Implementations
  - Some are Thin Wrappers Around Hardware
  - Distinction is Transparent

Building Applications: Blink

- Blink.nc
- BlinkM.nc
- Makefile – (local to Blink) application specific build instructions
  - COMPONENT=Appname (Blink)
  - Radio frequencies
  - Debug options
- Makerules – in apps directory, global to all apps
  - System level build instructions
    - Group ID, Code Optimization
  - Platform Specific Build Instructions
    - Target hardware and sensor board information
  - Install Specific Instructions
    - How to program (i.e., which uisp and uisp flags)
  - Other
    - E.g., JTAG Debug

Resolving Components / Interfaces



Exercise 1

- Change Blink Rate of Blink to 2 blinks per second
  - Edit BlinkM.nc
  - Look for 1PS
    - Change to TOS_I2PS, TOS_S2PS

defined at ~~tos~~/interfaces/Clock.h

- Make install.1 mica2

Exercise 2

- `$ make docs mica2`
- File is at doc/nesdoc/mica2/index.htm

apps/Blink Application Notes

Blink consists of two **components**: a **module**, called "BlinkM.nc", and a **configuration**, called "Blink.nc". Remember that all applications require a single top-level configuration, which is typically named after the application itself. In this case Blink.nc is the configuration for the Blink application and the source file that the nesC compiler uses to generate the executable for the mote. BlinkM.nc, on the other hand, actually provides the *implementation* of the Blink application. As you might guess, Blink.nc is used to wire the BlinkM.nc module to other components that the Blink application requires.

The reason for the distinction between modules and configurations is to allow a system designer to quickly "snap together" applications. For example, a designer could provide a configuration that simply wires together one or more modules, none of which she actually designed. Likewise, another developer can provide a new set of "library" modules that can be used in a range of applications.

Sometimes (as is the case with Blink and BlinkM) you will have a configuration and a module that go together. When this is the case, the convention used in the TinyOS tree is that Foo.nc represents a configuration and FooM.nc represents the corresponding module. While you could name an application's implementation module and associated top-level configuration anything (ncc uses the 'COMPONENT' definition in the application's Makefile to find the top-level configuration), to keep things simple we suggest that you adopt this convention in your own code. There are several other naming conventions used in TinyOS code. The TinyOS coding and naming conventions is described in **doc/tutorial/naming.html**.

nesC Terminology Review

- Application
- Component
  - Configuration
  - Module
- Interface
  - Command
  - Event
  - provides
  - uses

What's Left?

- Tasks
- Split Phase Operations
- Parameterized Interfaces
- Fan In & Fan Out Component Wiring

The Advanced Signal Processing Company

Review of the TinyOS Kernel

- TinyOS Kernel: 2 Level Scheduling Structure
  - Events - Small Amount of Processing
    - Can Interrupt Longer Running Tasks
  - Tasks - Larger Amount of Processing
    - Run to Completion WRT Other Tasks
      - Implies Only Need a Single Stack

The Advanced Signal Processing Company

Blink Again

- There were no Tasks in Blink

- Sleeps
- Event – Clock fire, toggle led,
- Back to Sleep

- So let's look at an example Task

DspLabs
The Advanced Signal Processing Company

Snippets From apps/SenseTask/SenseTaskM.nc

```
task void processData() {
  int16_t i, sum=0;

  for (i=0; i<maxdata; i++)
    sum += (rdata[i] >> 7);
  display(sum >> shiftdata);
}

inline void putdata(int16_t val) {
  int16_t p = (int16_t)head;
  head = (p+1) & maskdata;
  rdata[p] = val;
  }

event result_t   ADC.dataReady(uint16_t data)
{
  putdata(data);
  post processData();
  return SUCCESS;
}
```

errors

TinyOS provides a two-level scheduling hierarchy consisting of **tasks** and **events**. As we have seen, an event is a generalization of an interrupt handler, a call that is made from a lower-level component to a higher-level component in response to some event. Events must only do a small amount of work. ***before completing, because events always run to completion and cannot be preempted.*** **Tasks** are used to perform longer processing operations, such as background data processing, that can be preempted by events. A task is declared in your implementation module using the syntax

```
task void taskname() { ... }
```

where `taskname()` is whatever symbolic name you want to assign to the task. Tasks must return void and may not take any arguments.

To dispatch a task for (later) execution, use the syntax

```
post taskname();
```

A task can be posted from within a command, an event, or even another task.

DspLabs
The Advanced Signal Processing Company

Tasks Characteristics

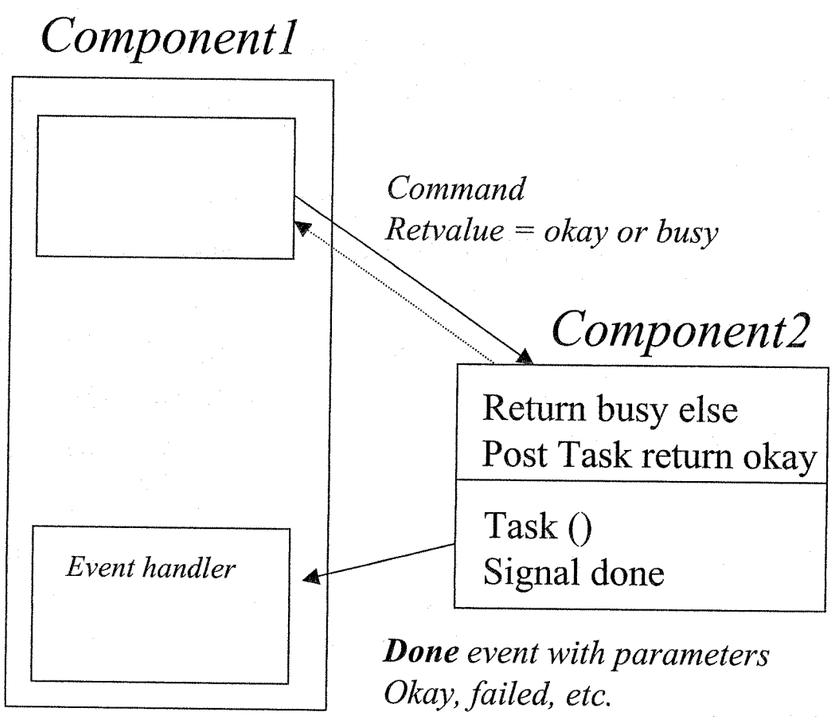
- Tasks do not Preempt Other Tasks
- Short Latency Tasks Are Nice
- Long Latency Tasks prevent other Tasks from executing
  - Break It Up into a Cascade of Tasks

task 1{ …. post task2}

task 2{…. post task3}

…

Split Phase Operations (aka Handshaking between Components)



Split Phase Defined

- 2 Components (*C1,C2)* Wired Together
- Upper Component *C1*
  - Uses the Interface
- Lower Component *C2*
  - Provides the Interface
- *C1* initiates the Split Phase Operation
  - call *C2.COMMAND(data1,data2,...)*
- *C2.COMMAND* returns FAILURE immediately
  - *C2* isn't Accepting Requests
  - *C1* Error Handling
- Else *C2.COMMAND* Posts Task and returns OKAY immediately
- *C2* Task Signals Event at Completion
- *C1* must handle this event

The Advanced Signal Processing Company

Parameterized Interfaces

- Multiple Instantiations of a component
  - E.g., ADC (Analog-Digital Converters)
  - One ADC component, but 8 ADC channels
  - How does TinyOS handle this ?

ADC

```
configuration ADCC
  {
    provides {
      interface ADC[uint8_t port];
      interface ADCControl;
    }
  }
  implementation
  {
    components ADCM, HPLADCC;

    ADC = ADCM;
    ADCControl = ADCM;
    ADCM.HPLADC -> HPLADCC;
  }
```

Uses ADC

```
includes sensorboard;
configuration AccelC
{
  provides interface ADC as AccelX;
  provides interface ADC as AccelY;
  provides interface StdControl;
}
implementation
{
  components AccelM, ADCC;

  StdControl = AccelM;
  AccelX = ADCC.ADC[TOS_ADC_ACCEL_X_PORT];
  AccelY = ADCC.ADC[TOS_ADC_ACCEL_Y_PORT];
  AccelM.ADCControl -> ADCC;
}
```

The Advanced Signal Processing Company

nesC Code
(Grok'ing) Lab

- Configuration Only Apps (No Module)
  - CntToLeds
    - Counter
    - IntToLeds
    - TimerC[ID]
  - CntToLedsAndRfm
    - Fan Out
  - RfmToLeds

Sniffer Pair Lab

Configuration Only Application

- CntToLeds.nc – Display Count on LEDs
  - Wires Up
    - A Counter : tos/lib/Counter.nc
    - The LED Display: tos/lib/IntToLeds.nc
    - The Timer: tos/system/TimerC.nc

CntToLeds
Configured from Existing Components

```
/** This application is built by wiring the TimerC, IntToLeds and
    Counter components together.
**/
configuration CntToLeds {
}
implementation {
  components Main, Counter, IntToLeds, TimerC;

  Main.StdControl -> IntToLeds.StdControl;
  Main.StdControl -> Counter.StdControl;
  Main.StdControl -> TimerC.StdControl;
  Counter.Timer -> TimerC.Timer[unique("Timer")];
  Counter.IntOutput -> IntToLeds.IntOutput;
}
```


Trigger the Counter
tos/system/TimerC.nc

```
configuration TimerC
{
  provides interface Timer[uint8_t id];
  provides interface StdControl;
}
implementation
{
  components LogicalTime;
  Timer = LogicalTime;
  StdControl = LogicalTime;
}
```

The Timer and StdControl interfaces we are providing is equivalent to the implementation in LogicalTime. Why can't we use an arrow?

In file tos/system/LogicalTime.nc:

```
configuration LogicalTime {
    provides interface Time;
    provides interface TimeSet;
    provides interface TimeUtil;
    provides interface StdControl;
    provides interface AbsoluteTimer[uint8_t id];
    provides interface Timer[uint8_t id];
}
```

```
implementation {
    components ClockC, LogicalTimeM, NoLeds,
TimeUtilC;
    Time = LogicalTimeM;
    TimeSet = LogicalTimeM;
    TimeUtil = TimeUtilC;
    StdControl = LogicalTimeM;
    Timer = LogicalTimeM;
    AbsoluteTimer = LogicalTimeM;
    LogicalTimeM.Clock ->ClockC;
    LogicalTimeM.ClockControl ->ClockC.StdControl;
    LogicalTimeM.Leds ->NoLeds;
    LogicalTimeM.TimeUtil -> TimeUtilC;
}
```

In file tos/system/LogicalTimeM.nc
module LogicalTimeM (lots of code...)

The Counter
tos/lib/Counter.nc

```
module Counter {
 provides {
  interface StdControl;
 }
 uses {
  interface Timer;
  interface IntOutput;
 }
}
implementation {
 int state;
command result_t StdControl.init()
 {
  state = 0;
  return SUCCESS;
 }
command result_t StdControl.start()
 {
  return call Timer.start(TIMER_REPEAT, 250);
 }
command result_t StdControl.stop()
 {
  return call Timer.stop();
 }

 event result_t Timer.fired()
 {
  state++;
  return call IntOutput.output(state);
 }

 event result_t IntOutput.outputComplete(result_t success)
 {
  if(success == 0) state --;
  return SUCCESS;
 }
}
```

The Advanced Signal Processing Company

IntOutput Implemented by IntToLEDS
tos/lib/IntToLeds.nc

```
configuration IntToLeds
{
 provides interface IntOutput;
 provides interface StdControl;
}
implementation
{
 components IntToLedsM, LedsC;
 IntOutput = IntToLedsM.IntOutput;
 StdControl = IntToLedsM.StdControl;
 IntToLedsM.Leds -> LedsC.Leds;
}
module IntToLedsM {
 uses interface Leds;
 provides interface IntOutput;
 provides interface StdControl;
}
implementation
{
 command result_t StdControl.init()
 {
  call Leds.init();
  call Leds.redOff();
  call Leds.yellowOff();
  call Leds.greenOff();
  return SUCCESS;
 }
 command result_t StdControl.start() {
  return SUCCESS;
 }
 command result_t StdControl.stop() {
  return SUCCESS;
 }
 task void outputDone()
 {
  signal IntOutput.outputComplete(1);
 }
 command result_t IntOutput.output(uint16_t value)
 {
  if (value & 1) call Leds.redOn();
  else call Leds.redOff();
  if (value & 2) call Leds.greenOn();
  else call Leds.greenOff();
  if (value & 4) call Leds.yellowOn();
  else call Leds.yellowOff();
  post outputDone();
  return SUCCESS;
 }
}
```

The Advanced Signal Processing Company

What is strange about the IntToLedsM component?

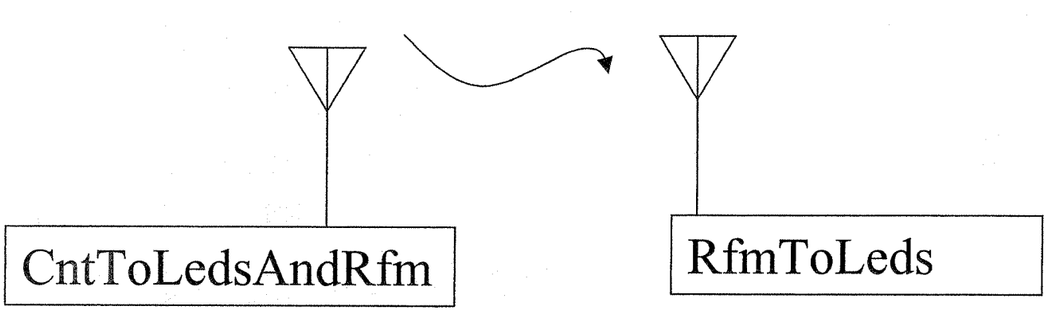
Fan Out
Count To LEDS and Radio

- Counter Output Fans Out
  - LEDS
  - Radio
- ```
  configuration CntToLedsAndRfm {
  }
  implementation {
    components Main, Counter, IntToLeds, IntToRfm, TimerC;

    Main.StdControl -> Counter.StdControl;
    Main.StdControl -> IntToLeds.StdControl;
    Main.StdControl -> IntToRfm.StdControl;
    Main.StdControl -> TimerC.StdControl;
    Counter.Timer -> TimerC.Timer[unique("Timer")];
   Counter.IntOutput -> IntToLeds;
   Counter.IntOutput -> IntToRfm;
  }
  ```


Sniffer Pair Lab
RfmToLeds
Display Radio Payload on LED

- Transmit Count to Receiver and Display


  ```

Build Transmitter Side
CntToLedsAndRFM

- Go to ../apps/CntToLedsAndRfm
- Set the Radio Frequency
  - Makefile
  - `CFLAGS -DCC1K_DEFAULT_FREQ=CC1K_433_002_MHZ`
- Check GROUPID
  - Makerules
  - `DEFAULT_LOCAL_GROUP := use # on your Name Tag`
- Plug-In MICA2 Unit #1 to Programmer
- make install.10 mica2
- Verify LEDS show count

The Advanced Signal Processing Company

Build Receiver Side
RfmToLeds

- Go to apps/RfmToLeds
- Set the Radio Frequency
  - Makefile
  - `CFLAGS -DCC1K_DEFAULT_FREQ=CC1K_433_002_MHZ`
- Check GROUPID
  - Makerules
  - `DEFAULT_LOCAL_GROUP := use # on your Name Tag`
- Plug-in MICA2 Unit #2 to Programmer
- make install.10 mica2

The Advanced Signal Processing Company

Things to Observe

- Turn Off Transmitter
  - LED counting stops on both Motes
- Move around and observe Receiver Count
- Sources of erratic Receiver Count
  - multipath, fade, collisions
- What is the dominant interference source here today?

TinyOS Packet Networking and PC Base Station Lab

- Active Messages
- Packet Structure and Networking
- Oscilloscope TinyOS Application
- ListenRaw Java Application
- Display Raw Packet Data Lab

"Active Messages" (AM) Messaging System Service

- Establishes a Connection between Sender and a Component in an Application
- e.g. Route Update messages vs Data Payloads
- tos/system/AMStandard.nc Delivers Message to Appropriate Component in the Application
- Inter-process Communication Mechanism
- Active Messages Similar to a Port On a Server
- AM Handler using Parameterized Interface

TinyOS Packet Structure
Addresses, AM#, GroupID and Payload
Public vs. Secure

```
typedef struct TOS_Msg //Public
{
  /* The following fields are transmitted/received on the
radio. */
  uint16_t addr;
  uint8_t type; //ACTIVE MESSAGE ID
  uint8_t group;
  uint8_t length;
  int8_t data[TOSH_DATA_LENGTH]; //system/AM.h
  uint16_t crc;
  /* The following fields are not actually transmitted */
  uint16_t strength;
  uint8_t ack;
  uint16_t time;
} TOS_Msg;
```

```
typedef struct TinySec_Msg //Secure
{
  uint16_t addr;
  uint8_t type;
  uint8_t length;
  // encryption iv
  uint8_t iv[TINYSEC_IV_LENGTH];
  // encrypted data
  uint8_t enc[TOSH_DATA_LENGTH];
  // message authentication code
  uint8_t mac[TINYSEC_MAC_LENGTH];

  // not transmitted - used only by MHSRTinySec
  uint8_t calc_mac[TINYSEC_MAC_LENGTH];
  uint8_t ack_byte;
  bool computeMACDone;
  bool validMAC;
} __attribute__((packed)) TinySec_Msg;
```

DspLabs
The Advanced Signal Processing Company

AM Routing
GroupID and ADDRESS Filter

```
// Handle the event of the reception of an incoming message
 TOS_MsgPtr received(TOS_MsgPtr packet)  __attribute__
((C, spontaneous)) {
  uint16_t addr = TOS_LOCAL_ADDRESS;
  atomic {
   counter++;
  }
  dbg(DBG_AM, "AM_address = %hx, %hhx;
counter:%i\n", packet->addr, packet->type, (int)counter);

  if (//packet->crc == 1 && // Uncomment this line to check
crcs
            packet->group == TOS_AM_GROUP &&
            (packet->addr == TOS_BCAST_ADDR ||
             packet->addr == addr))
    {
            uint8_t type = packet->type;
            TOS_MsgPtr tmp;

            // Debugging output
            dbg(DBG_AM, "Received message:\n\t");
            dbgPacket(packet);
            dbg(DBG_AM, "AM_type = %d\n", type);

            // dispatch message
            tmp = signal
ReceiveMsg.receive[type](packet);
            if (tmp)
              packet = tmp;
    }
  return packet;
 }

 // default do-nothing message receive handler
  default event TOS_MsgPtr ReceiveMsg.receive[uint8_t
id](TOS_MsgPtr msg) {
  return msg;
 }
```


Display Raw TinyOS Packet Data on PC Terminal via UART

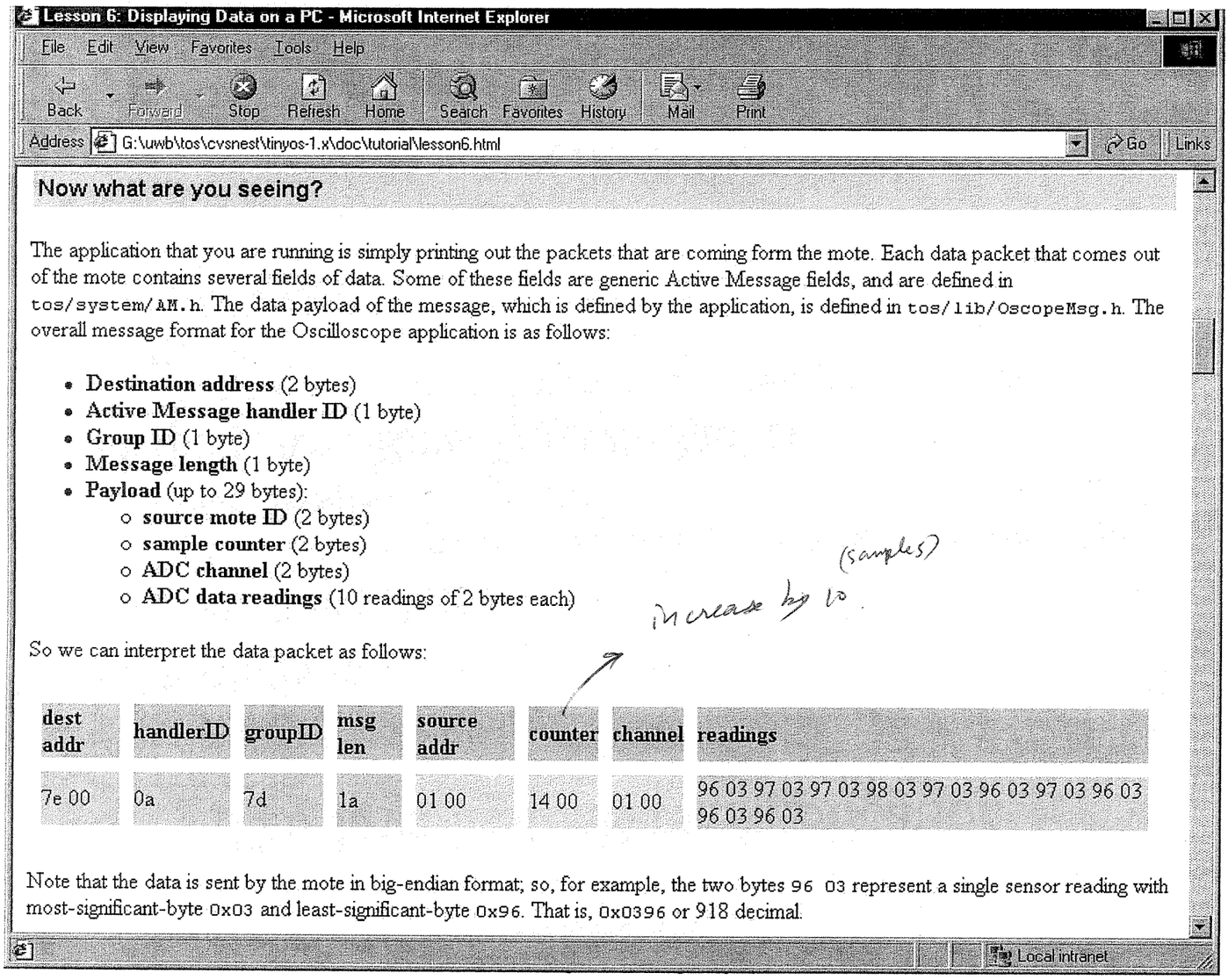
- Application sends/receives TinyOS Packets via UART
  - Oscilloscope
- Hardware Connections
  - Mote to MIB
  - MIB [to Serial Cable] to PC COM Port

Install TinyOS Application

- Oscilloscope
  - Plug-in Mica2 #1 into Programmer
  - Go to ../apps/Oscilloscope
  - make install.10 mica2

The Java ListenRaw Application

- cd tools/java
- List PC Com Ports
  - "`java net.tinyos.tools.ListenRaw -p`" will list the available ports
- Display Any Raw TinyOS Packets
  - "`java net.tinyos.tools.ListenRaw -mica2 COMn`"
  - `Note the -mica2 switch sets baud rate`



End of Morning Session

- Coming up this Afternoon
  - Hardware Overview
  - Radio Basics
  - TinyOS Programming Part II

nesC Programming II

Bill Maurer
maurer@dsplabs.com
DSP Labs, Livermore Ca., USA

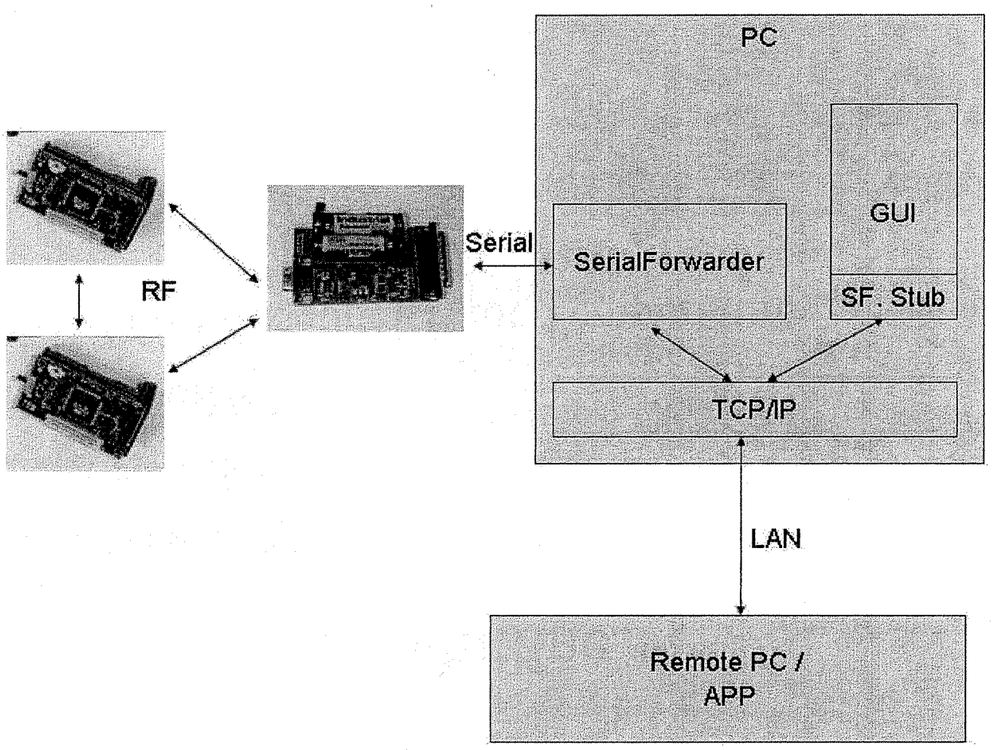
The Advanced Signal Processing Company

Outline

- Generic Base App
- Serial Forwarder
- OscilloscopeRF
- Modified OscilloscopeRF for Temperature Lab

The Advanced Signal Processing Company

Mote Network / "Active Messages" Server / Client Applications



"Active Messages" Server

- Serves up the "Active Messages" encapsulated in MOTE TinyOS Packets to Client Applications via TCP/IP
- Implemented using SerialForwarder
  - Java Server application running on PC
  - Client Interface via Sockets
  - Mote Network interface via Mote Basestation
    - Basestation Mote runs GenericBasic

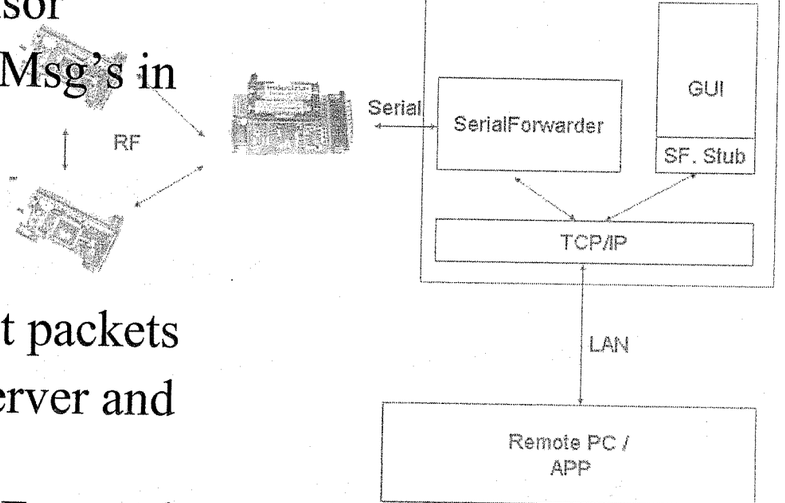
Mote Base Station
Generic Base Layer

- Radio-UART Bridge
  - apps/GenericBase.nc
    - Issues
      - No Packet Framing on UART side
        Fragile - Collisions between Tx and Rx Packets cause lockup.
  - apps/XGenericBase.nc
    - Packet Framing on UART
      - Robust
  - UART Baudrate
    - ../tos/platform/../hpluart0m.nc
      - ```
        outp(15, UBRR0L);//57.6Kbaud
        outp(47, UBRR0L);//19.2Kbaud
        ```

DspLabs
The Advanced Signal Processing Company

Building a Server Application
OscilloscopeRF

- OscilloscopeRF Application on Mote
  - Samples Photosensor
  - Broadcast OscopeMsg's in TinyOS Packets
- Base Station Mote
  - GenericBase
  - Receives broadcast packets
- "Active Messages" Server and Client GUI on PC
  - net.tinyos.sf.SerialForward
  - net.tinyos.oscope.Oscilloscope



DspLabs
The Advanced Signal Processing Company
        ```

Mote Network/"Active Messages" Server
How to Do it – Mote OscilloscopeRF

– Plug-in Mica2 Mote#1 in to Programmer
– Go to ../apps/OscilloscopeRF
– Make install.10 mica2

The Advanced Signal Processing Company

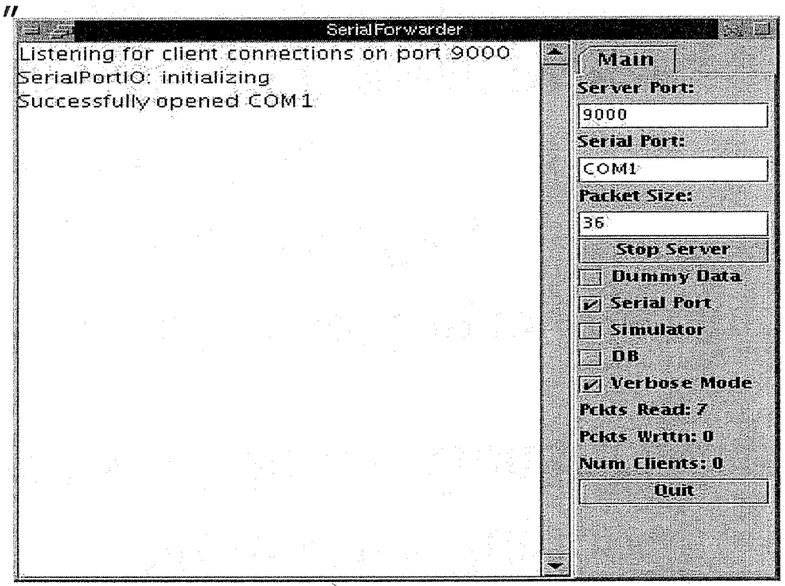
Mote Base Station
How to Do it – GenericBase

– Plug-in Mica2 Mote#2 in to Programmer
– Go to ../apps/GenericBase
– Make install mica2
– Connect MIB to PC Serial Port

The Advanced Signal Processing Company

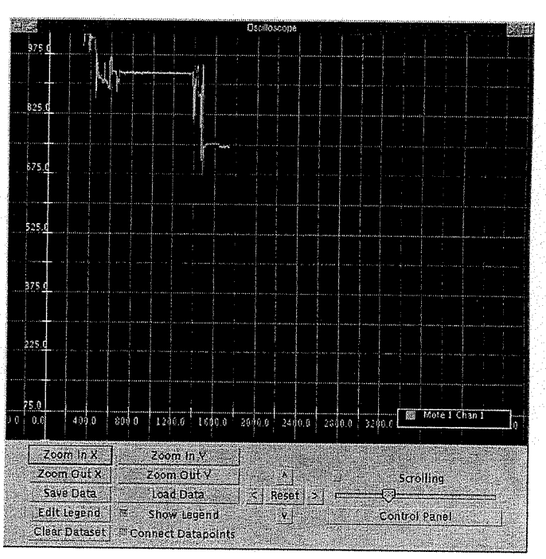
Mote Network/ "Active Messages" Server How to Do it –SerialForwarder

- Go to ../tools/java
  - "java net.tinyos.sf.SerialForward -baud 57600 [-debug]"



Mote Network/ "Active Messages" Server How to Do it – Oscilloscope Client

- Go to ../tools/java/net/tinyos/oscope
  - java net.tinyos.oscope.oscilloscope
  - If you don't see a signal (green lines)
    - Select the "Scrolling" Check Box
    - Click the Resest Button



A New Oscilloscope

- Sense the Temperature instead of PhotoSensor
- Backup ../apps/OscilloscopeRF
- Modify Configuration File
  - `configuration Oscilloscope { }`
    `implementation`
    `{`
    `components Main, OscilloscopeM, ClockC, LedsC, Temp,`
    `GenericComm as Comm;`
    `Main.StdControl -> OscilloscopeM;`
    `OscilloscopeM.Clock -> ClockC;`
    `OscilloscopeM.Leds -> LedsC;`
    `OscilloscopeM.SensorControl -> Temp;`
    `OscilloscopeM.ADC -> Temp;`
    `OscilloscopeM.CommControl -> Comm;`
    `OscilloscopeM.ResetCounterMsg ->`
    `Comm.ReceiveMsg[AM_OSCOPERESETMSG];`
    `OscilloscopeM.DataMsg -> Comm.SendMsg[AM_OSCOPEMSG];`
    `}`

FINISHED ☺

- End of First Day
- Sake .....

MultiHop Networking Session

Overview:

- Characteristics of wireless mesh networks
- TinyOS Status of MultiHop Protocols

Ad-Hoc Routing (Self configuring)

- Links are not reliable over the long term
- Links change dynamically
- Requires networking topology that also dynamically changes.
- Low energy requirements limit types of protocols. Powered networks can afford to expend a lot more energy to manage links.
- Broadcasting is energy and time inefficient
- Protocols where the motes dynamcially determine the best parent are attractive.

Mote Msgs Constrain Sleep Time

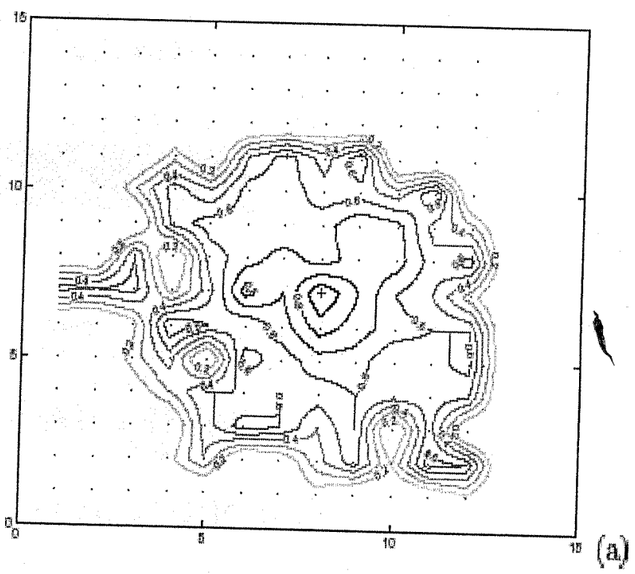
Computation of required sleep time to achieve duty cycle given number of TOS msgs to rcv/xmit while awake

	value	units
Specifications		
Msg Size	40	bytes
Msg Preamble	16	bytes
Baud Rate	38400	baud
Duty Cycle	0.5	%
# of msgs to rcv/re-xmit during wake time	5	msg
Computed Values		
Time to xmit/rcv 1 msg	11.7	msec
Time to rcv/xmit all msgs	116.7	msec
Required sleep time to maintain duty cycle	23.22	sec

Motes closer to base station get more messages

Radio Link Behavior #1

• Radio contour plot shows received radio strength varies significantly .



(a)

† mote

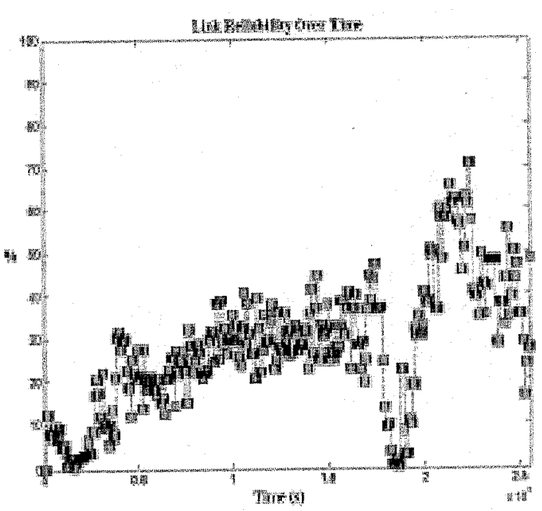
radio strength contour

From Alec Woo, To be Published

Radio Link Behavior #2

Static links show variability in receive strength over time

• Local null effects, people, … influence quality of link



From Alec Woo, To be Published

(c) Link reliability variations between two immobile nodes over a period of seven hours, with receiver placed at the cell edge of the transmitter.

Broadcasting (flooding)

- Backward links
- Longs links
- Stragglers (no reception)
- Clustering
- Asymmetric links

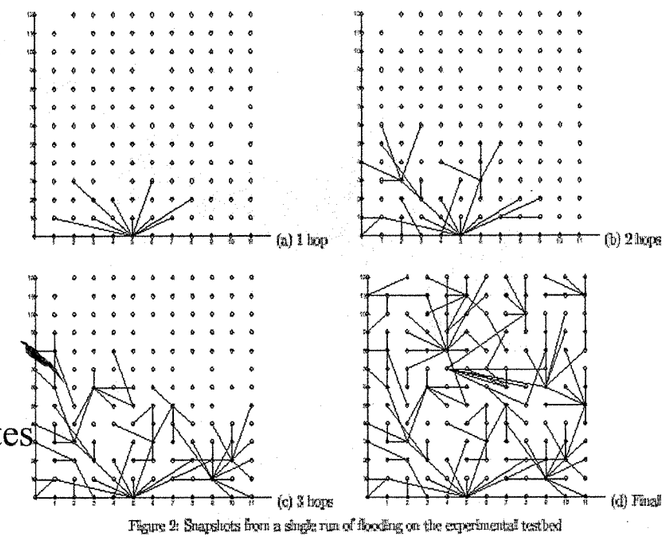


Figure 2: Snapshots from a single run of flooding on the experimental testbed

13x12 grid (156 motes)

Flooding at 19.2K baud Tos Packet, 156 motes

- 80msec to receive then transmit
- Each mote transmits once:
  - 156*40msec = 6.25 sec (no rndm delay)
  - Probably ~13 seconds with rndm delay
  - Large probability of pckt collision.

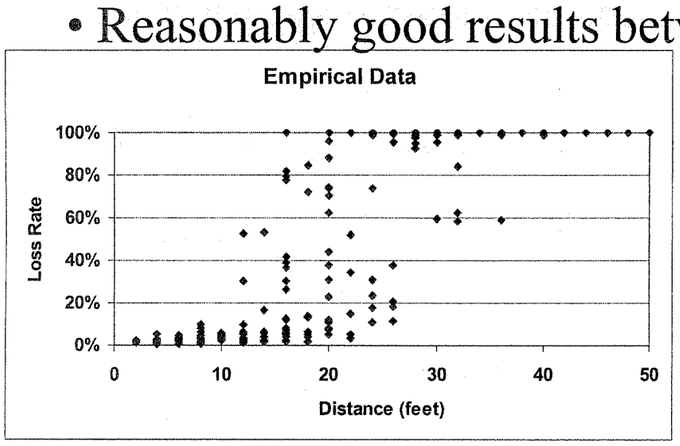
An Empirical Study of Epidemic Algorithms in Large Scale Multihop Wireless Networks

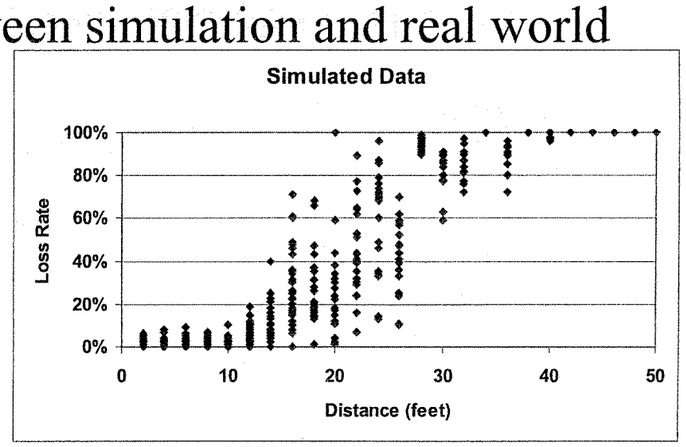
Deepak Ganesan, UCLA; Bhaskar Krishnamachari, Cornell; Alec Woo, UC Berkeley; David Culler, Intel Research and UC Berkeley; Deborah Estrin, UCLA; Stephen Wicker, Cornell.

Simulation

Nido

- TinyOS supports PC simulation of networks.
- Best way to start, prove out code before real implementation
- Reasonably good results between simulation and real world





TinyOS and Multihop

- Complex routing protocols
- Complex energy management
- Complex time management
- No single multihop stack; application dependant
- TinyOS will allow users to wire in different protocols with minimal effort.
- TinyOS will allow users to compare different protocols for reliability and efficiency.
- Time/Power management is being added to these protocol (3-4 months)

MultiHop Components

- Routing beacon from base station to establish route paths back.
- Find best parent to forward messages:
  - Routing Table: List of best neighbors and routing info. Sram constrained.
  - Table Management: Eviction/Insertion of neighbors into routing table
  - Estimator: Computation of neighbors link quality. Ex: # of hops, sequence numbers. Estimators differentiate strategies.
  - Parent Selection: Decide on parent to forward messages.
  - Cycle Detection: Avoid loops (I.e. forwarding msg to child instead of parent
- Timer : Period update of routing tables, messaging…
- Power management

From Alec Woo, To be Published

MultiHop Components

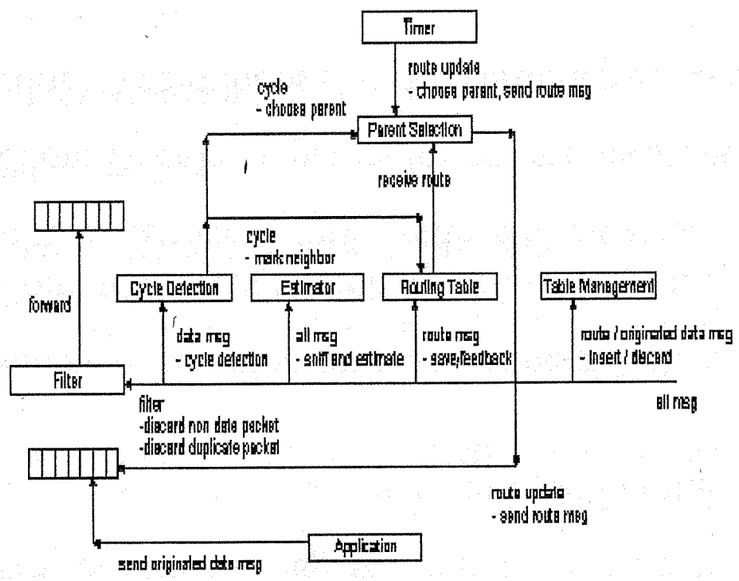
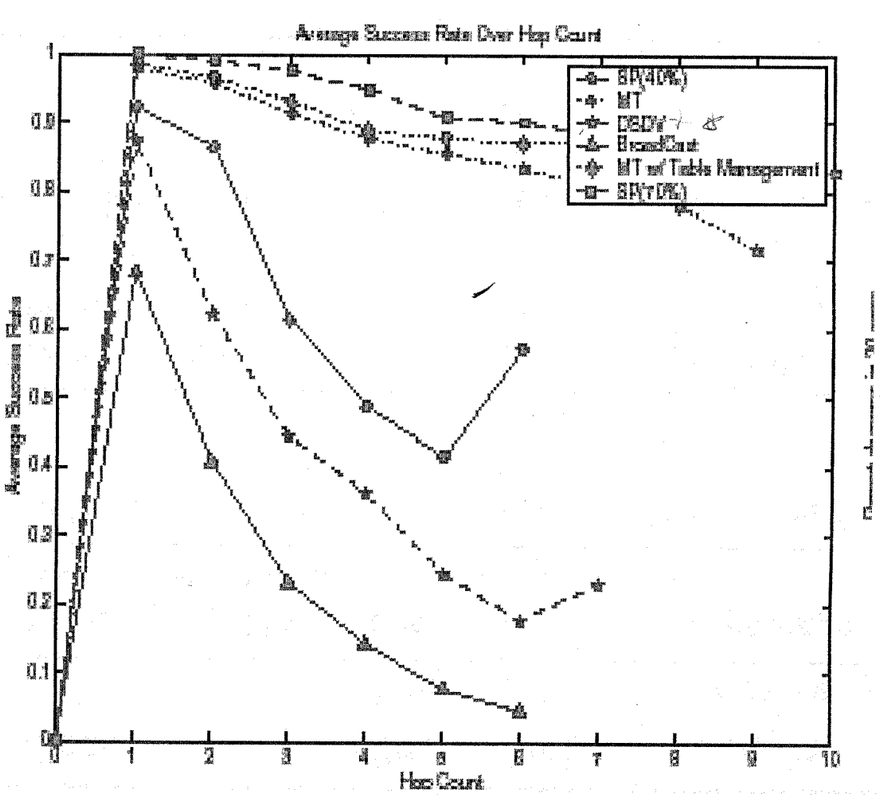


Figure 9: Message flow chart to illustration the different components for routing.

From Alec Woo, To be Published

Multihop Throughput

- Can achieve ~90% packet throughput over 5-6 hops with good protocols.
- Link estimators critical to good multihop. Need to change quickly as link quality changes.



Simulated Results;
From Alec Woo, To be Published

Some TinyOS Multihop Stacks

Protocol	Comments	Status	Pwr Mng	Authors	TinyOS Location
Surge	Used in TnyDB, GSK.	Being replaced by Alec Woos protocol	none	UCB	apps/surge
DSDV	Mote version of standard algorithm. Good results.	Released and stable	**	Intel Marsboroughl	contrib/hsn
TinyDiff	Now deployed in James Preserve.	Doesn't support mote base station (1-2 months)	*	UCLA-CENS	contrib/TinyDiff
Alec Woo	PhD research. A lot of work on estimators. May be best when done.	Should be released in next few months.	***	Alec Woo UCB	Not released yet

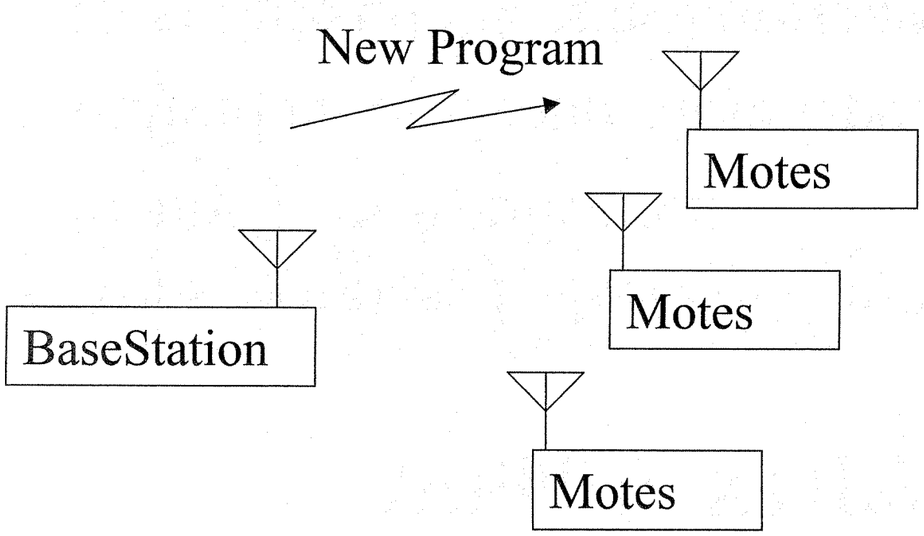
* Power management implementation in progress (in 1-2 months)

** May be released with power management (future version)

*** Currently using long radio preambles

XNP

- Crossbow In-Network Programming



What Is XNP?

- Adds Wireless In-Network re-programming to any TOS Application
- Individual or Group Mote Updates
- Host Side GUI Control program
- Compatible with Single and Multi-hop Routing (forward & reverse path)

How XNP Works

- Two Phases
  - Download Application's SREC file from Host to local FLASH Memory
  - Re-program Mote
- Implemented as a private Active Message Service – all XNP radio messages are processed independently of Application

Step #1 XNP Download

- Host Broadcasts Start Download message
- Host broadcasts srec file (2 TOS packets per srec record/capsule) to all Motes.
- Active Motes store capsules in FLASH
- Host Queries motes for any "missing/lost" capsules
- Motes request "missing" capsule id
- Host transmits lost capsule
- Repeat until all Active Motes have complete image

Step #2 XNP Re-Program

- Host Broadcasts Re-Program / Re-Boot command with Program ID
- Active Motes verify Program ID matches downloaded ID
- Motes store their current Mote ID in non-volatile memory
- Motes re-program themselves and re-boot
- Application fetches Mote ID from non-volatile memory and restores ID.

How to Use XNP

- Add XNP Event Support functions
  - XNP signals a request to start XNP download
  - Application must release resources (External FLASH)
  - Application responds with GRANT or DENY
  - XNP signals Done when XNP download ends
- Add `Xnp.NPX_SET_IDS` call in INIT to restore Mote and Group Ids.
- Wire xnpc.nc into application
- Install Application and XNP Bootloader in Motes

Example Interface to XNP

```
• event result_t Xnp.NPX_DOWNLOAD_REQ(uint16_t
      wProgramID, uint16_t wEEStartP, uint16_t
      wEENofP)
      {//Acknowledge NPX
      call Xnp.NPX_DOWNLOAD_ACK(SUCCESS);
      return SUCCESS;
      }//event download_req
• event result_t Xnp.NPX_DOWNLOAD_DONE(uint16_t
      wProgramID, uint8_t bRet,uint16_t wEENofP){
      return SUCCESS;
      }//event download_done
• command result_t StdControl.init() {
      call Xnp.NPX_SET_IDS(); //restore id s
      … //standard init code
      }
```

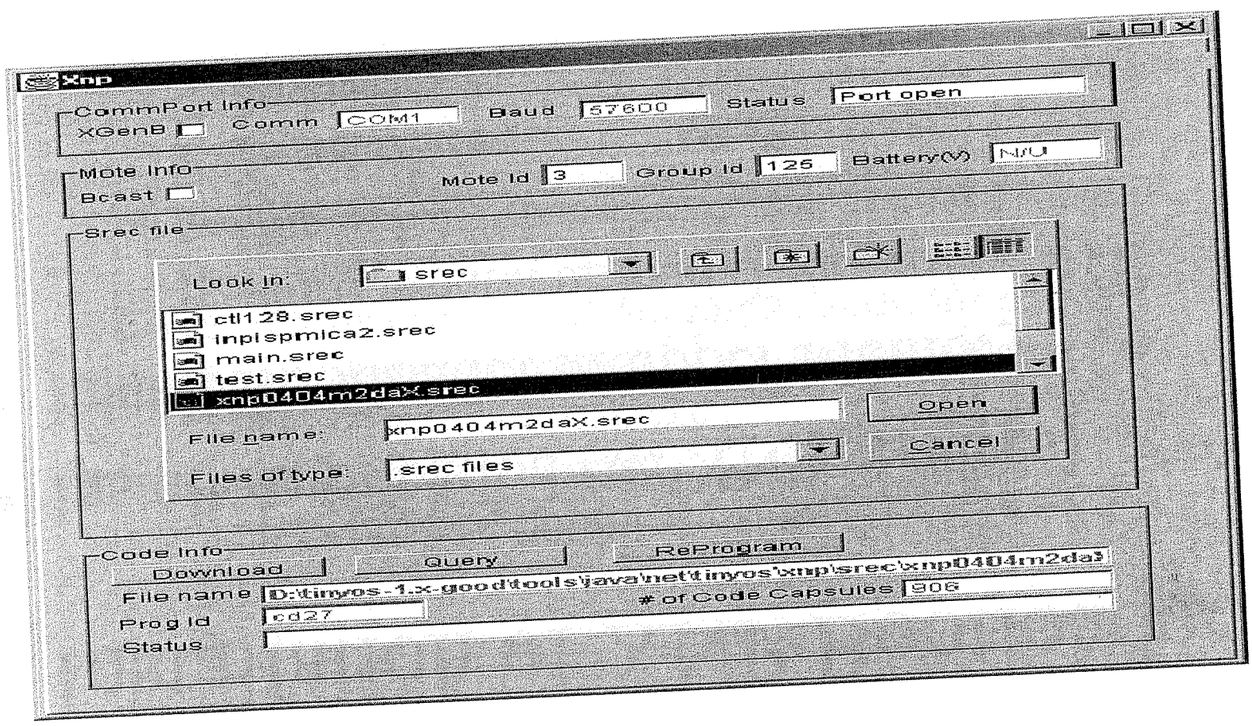
Example Component Wiring

```
configuration XTestXnp {
  }
  implementation {
    components Main, GenericComm, ClockC, LedsC,
  XTestXnpM, XnpC;
    Main.StdControl -> XTestXnpM.StdControl;
    XTestXnpM.GenericCommCtl -> GenericComm;
    XTestXnpM.Clock -> ClockC;
    XTestXnpM.Leds -> LedsC;
    XTestXnpM.Xnp -> XnpC;
  }
```

Installation of XNP Bootloader

- Install Application (w/ XNP services)
- Install Bootloader in upper Memory
  - uisp –dprog=dapa upload if=*bootfile*
    - *Bootfile*
      - Inpispm2.srec Mica2
      - Inpispm2d.srec Mica2Dot

XNP Host User Interface



Demonstration Example

- Afternoon Session

Mote Power Management Session

Overview:

- Battery Life Calculation
- Mote Sleep
- Radio Power Management

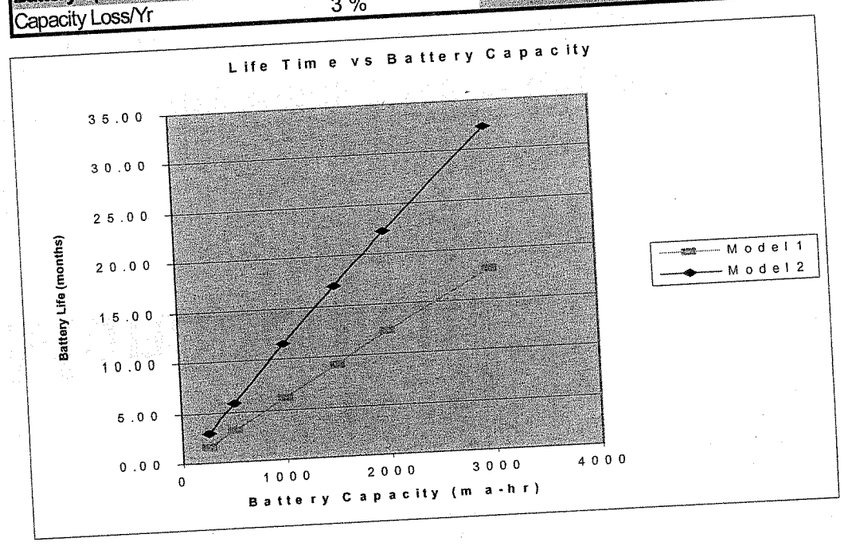
Power Management

- Not hard for single hop
  - Wake up and transmit only strategies have resulted in multiyear battery life (Axxon, 5yr). Prone to collisions, can't multihop for more distance.
  - Sleep and sniff mote strategies (~3-4 yr).
- Multiple hop becomes hard
  - Need to wake-up, send data, forward messages.
  - Bandwidth crunch (see slide from multiphop seesion)
- Must sleep 99% of the time or some equivalent strategy!

Battery Life Computation

- Two models:
  - 1% duty cycle
  - 0.5% duty cycle
- Conclusion:
  - To achieve multi-year battery life, must sleep most of the time.
- Spread Sheet in on CD

SYSTEM SPECIFICATIONS			Duty Cycles	
Currents	value	units	Model 1	Model 2 units
Micro Processor (Atmega128L)				
current (full operation)	6 ma		1	0.5 %
current sleep	8 ua		99	99.5 %
Radio (Chipconn 1000)				
current in receive	8 ma		0.75	0.4 %
current xmit	12 ma		0.25	0.1 %
current sleep	2 ua		99	99.5 %
Flash Serial Memory (AT45DB041)				
write	15 ma		0	0 %
read	4 ma		0	0 %
sleep	2 ua		100	100 %
Sensor Board				
current (full operation)	5 ma		1	0.5 %
current sleep	5 ua		99	99.5 %
Battery Specifcations				
Capacity Loss/Yr	3 %			



Mote Sleep

- Atmega128 will sleep at 10uA.
- All pins must be in correct state.
- Setting to any pin to incorrect state will typically add >100uA of current.
- Mica2 sleep example:

  contrib/xbow/apps/XTestSnooze

- UCB introducing more advance power management strategies.

Radio Power: Long Preambles

- Preamble is a series of 0x55 (alternating 0,1s) set of bytes.
- Radio receiver uses this to balance (zero) receiver.
- Strategy: User very long preamble so that sleeping motes can wake up and still catch the preamble.
- Example: If preamble is ~200msec then mote can wake up every 100msec for 10 msec to sample the preamble (10% duty cycle)
- New addition to TOS in mica2 platform. See CC1000const.h. Defines preamble up to ~1 second in length.
- Is this scalable for a large network?
- Penalty for transmitter power.

Radio Power: RSSI Sniff

- Chipconn radio can turn on quickly from sleep : ~200usec.
- Strategy: wake up Atmega128, sample radio's RSSI
- Either go to sleep or wake up to receive a radio packet.
- No implementation in TinyOS yet.