Tutorial at ASCI 2008

Fast-Prototyping of Wireless Sensor Networks

Jan Beutel, ETH Zurich
Today’s Objective

Learn how to build a typical wireless sensor network application

• Get our hands dirty with “real” sensor network implementations

• Learn about their challenges and caveats

• This tutorial does not require prior experience in embedded programming and thus is suitable for researchers from all areas, e.g. people that have not yet had (extensive) platform experience

• Have fun…
BTnode Programming – An Introduction to BTnut Applications

- In-depth BTnode Tutorial
  - Originating in lectures at ETH Zurich
  - Set up in separate sessions
  - Minimum prerequisites required
  - Available online

- This tutorial uses excerpts from
  - First Steps
  - Bluetooth Multihop
  - Interfacing to Handhelds
  - Sensors and Debugging

- [http://www.btnode.ethz.ch](http://www.btnode.ethz.ch)
Material

• What you should have received
  – BTnode tutorial manual
  – Slide copies
  – CDROM containing software tools and doc’s
  – BTnode developer kit
    • BTnode rev3
    • USBprog board
    • 2xAA rechargeable cells
    • ATAVRISP programmer
    • USB cable
    • Serial Cable
  – A Teco Particles SSMALL sensor board

• Optional
  – USB-UART transceiver + 2nd USB cable
Your Requirements as Participants

• The hands-on experience requires you to
  – Install the necessary tools on your laptop
    • root/admin access
    – Handle a build system (make)
    – (program in C)

• We suggest you
  – Work in groups of two
  – Use a Windows laptop
    • Non-windows see http://www.btnode.ethz.ch/Documentation/Installation

🌟 Hands-on exercises are marked in red on the slides
  – 1\textsuperscript{st} demonstration of the exercise – 2\textsuperscript{nd} time for participants
Outline

• Introduction
  – Basic concepts of embedded wireless sensor network platforms
  – Overview of the BTnode platform
    • Hardware architecture
    • BTnut system software

• Hands-on
  – Installing/getting to know the development tools
  – First steps in BTnode programming
    • Plugging things together
    • Basic communication, ISP programming
    • The `bt-cmd` application – simple Bluetooth networking
  – My first BTnut multi-hop application
    • Bluetooth networking basics
    • Multihop networking
    • Sensor interfaces, packets and payload
    • Bluetooth Scatternet topology visualization
Outline continued...

- **Demonstration**
  - Debugging and profiling of sensor network applications
    - Embedded debugging
    - Profiling using and OS tracer
    - Testbeds – The ETH Deployment-Support Network

- **Hands-on**
  - Interfacing to handheld devices
    - Bluetooth RFCOMM
    - AT commands
    - Sending an SMS message using AT commands

- **Question and answer**

- **Time to explore BTnodes...**
Basic concepts of embedded wireless sensor network platforms

- **“Mote class” devices**
  - Microcontroller + low-power radio
  - Battery powered
  - Many custom applications
  - Large design space, many variants
  - Most prominent examples: Mica2, Mica2Dot, Tmote Sky

- **Hardware is packaged with**
  - System software and apps (e.g. TinyOS, BTnut, Contiki, Mantis, ...)
  - Basestations, network access
  - Server-side solutions (backends)
  - Tools (e.g. simulators, ...)

---

The content is presented in a clear, logical manner, explaining the key components and features of embedded wireless sensor network platforms, focusing on the "Mote class" devices and the components that come with them.
The BTnode Platform

Communication

Bluetooth Module

ATmega128L Microcontroller

Generic Interfaces

Computation

IO/Peripherals

Prototype

2nd Generation

3rd Generation
The BTnode rev3 – Architecture Details

- **System core**
  - Atmel ATmega128
  - 256 kB SRAM
  - Generic IO/peripherals
  - Switchable power supplies
  - Extension connectors

- **Dual radio system**

- **Bluetooth radio**
  - 2.4 GHz Zeevo ZV4002

- **Low-power radio**
  - 433-915 MHz ISM
    - Chipcon CC1000

- **On-board antennas**
The BTnode rev3 – Atmel AVR Microcontroller Architecture

- **ATmega128l**
  - 8-bit AVR RISC @ 7.3 MHz
  - 64k address space
  - Integrated peripherals
  - Configurable using fuse bits

- Programs resident in flash memory
  - Max. size 128 Kbyte
  - One program at a time only

- System core – bus systems
  - UART0: Bluetooth
  - UART1: Ext. terminal, programming
  - SPI: Low-power radio, sensors
The BTnode rev3 – In-situ Power Profiling Function

Negri2005/2006
The BTnut System Software

• Versatile and flexible fast-prototyping
  – Lightweight operating system support in plain C
  – Standard GNU tools, avr-libc
  – Simple demo applications and tutorial

• Built on top of multi-threaded Nut/OS framework
  – Oriented towards networking applications
  – Non-preemptive, cooperative multi-threading
  – Events, timers
  – Priorities for threads
  – Dynamic heap allocation
  – POSIX style device drivers
  – OS tracer (μsec resolution)
• TinyOS is the de-facto standard for WSN software

• BTnut is plain vanilla C using the GNU toolchain, avr-libc
  – Less dependencies, no need for extra tools
  – No need to learn new languages/abstractions (nesC)

• BTnut offers support for concurrency through threads
  – Intuitive to program
  – No need to express all system functions using state-machines

• BTnut offers a clear and simple structure
  – Suitable for a quick jump-start and fast learning curve
  – Many features and tools target fast-prototyping
BTnodes are not targeted at ultra low-power...
... but target versatile and flexible fast-prototyping.

- Multi-threaded OS frame in C
  - Standard open-source tools
  - Lightweight software distribution (12.3 MB binary, 32.5 MB source)

- Rapid prototyping
  - HW emulation on Linux PC

- Demo applications and tutorial
  - Different labs for graduate lectures
  - (Multi-)day tutorials
The BTnode Platform – Beyond Hard- and Software...

• Hardware and system software don’t yet make a platform...

• Documentation resources
  – Hardware: Datasheets, schematics, parts, design specifications
  – Software: API, libraries, compilers, demo apps
  – Many documented projects and applications

• The BTnode community
  – Development hosted on sourceforge.net (version control, tracker)
  – Wiki based web pages
  – Mailing list
  – Continuous integration using CruiseControl
BTnode Platform Success

- **Industrial technology transfer**
  - Commercialization with ETH spin-off “Art of Technology”
  - Commercial replicas resulting from open source policy

- **BTnodes in education**
  - Different labs and demos
  - Graduate lab in embedded systems (120 participants)
  - 50+ successful completed student projects

- **BTnodes in research domains**
  - 35+ wearable and ubiquitous computing applications and demos
  - Wireless (sensor) network research
  - 50+ scientific publications based on or related to BTnodes

---

**BTnode dev kit € 500**

**Vitronics Cobalt Blue™ Bluetooth Board**
Outline

• Introduction
  – Basic concepts of embedded wireless sensor network platforms
  – Overview of the BTnode platform
    • Hardware architecture
    • BTnut system software

• Hands-on
  – Installing/getting to know the development tools
  – First steps in BTnode Programming
    • Plugging things together
    • Basic communication, ISP programming
    • The bt-cmd application – simple Bluetooth networking
  – My first BTnut multi-hop application
    • Bluetooth networking basics
    • Multihop networking
    • Sensor interfaces, packets and payload
    • Bluetooth Scatternet topology visualization
Hands-on: Installation of the Development Tools

• Development tools
  – AVR toolchain with avr-gcc, make, avrdude, ...
  – Terminal application (Hyperterm, Minicom, ZOC, ...)

🌟 AVR toolchain installation
  – Open the directory inss2007 on the BTnode CDROM
  – Install WinAVR-20060125-install.exe into C:\WinAVR
  – Alternate instruction http://www.btnode.ethz.ch/Documentation/WinInstall

🌟 Terminal application installation
  – Install zoc507_win_english.exe

🌟 Testing the installation
  – Open a cmd shell and execute avr-gcc --version
Hands-on: Installation of the BTnut Software

- **BTnut software**
  - Precompiled libraries
  - Demo examples
  - Documentation

★ Installation
  - Unpack `btnut_snap_btnode3_binary_1.8.tar.gz` into `C:\btnut_snap`

- **Directory structure**
  - Demo applications
  - Documentation
  - Precompiled libraries/API
Hands-on: Browsing the BTnut API Documentation

- BTnut API documentation
  - Available in `doc/html/index.html`
  - Inline docs generated from source code using doxygen
  - Also available online
    - [http://www.btnode.ethz.ch/static_docs/doxygen/btnut](http://www.btnode.ethz.ch/static_docs/doxygen/btnut)

★ Optional: Browsing the BTnut API – LED driver docs (Ex 2.4)
  - Find `btnode/include/led/btn-led.h`
  - Read and understand the documentation for `btn_led_init()` and `btn_led_add_pattern()`
Outline

• Introduction
  – Basic concepts of embedded wireless sensor network platforms
  – Overview of the BTnode platform
    • Hardware architecture
    • BTnut system software

• Hands-on
  – Installing/getting to know the development tools

  – First steps in BTnode Programming
    • Plugging things together
    • Basic communication, ISP programming
    • The bt-cmd application – simple Bluetooth networking

  – My first BTnut multi-hop application
    • Bluetooth networking basics
    • Multihop networking
    • Sensor interfaces, packets and payload
    • Bluetooth Scatternet topology visualization
Plugging Things Together – The BTnode Development Setup

- Developer Workstation
- In-System Programmer attached to COMy
- USB Terminal “Console” attached to COMx
- BTnode
Hands-on: Plugging Things Together – The BTnode Terminal

- USBprog adapter board
  - CP2101 USB-UART transceiver
  - Power via USB
  - Breakout connectors for prototyping
  - Sensor interface connector

⚠️ BTnode terminal connection
  - USB cable
  - USBprog adapter board
  - BTnode

  - Watch out for the correct orientation!

🌟 Optional: Installation of the CP2101 driver
Hands-on: BTnode Terminal – Basic Device Communication

★ BTnode terminal configuration
- Find the right COMx port
  • Start list_cp2101uart.vbs

  - Start a terminal application (e.g. minicom or ZOC) using 57600, 8N1, no handshake

  - Shortcut: Start BTnode_COM4.zoc to connect to COM4 (edit for other COMx ports)

★ BTnode terminal operation
- Press the reset button on the BTnode and observe the terminal
Hands-on: Plugging Things Together – The ISP Programmer

- AVR programming
  - Programs are resident in flash memory (ATmega128l = 128 Kbyte)
  - Different AVR programming variants
    - Serial using a hardware programmer
    - JTAG
    - Bootloader
    - (Parallel)

★ ATAVRISP programmer connection
  - Connect to J2 on USBprog and a serial port on the PC
  - No serial port available? Use the USB-UART transceiver

★ Testing the ISP programming tool installation (Ex 2.10)
  - Open a cmd shell and execute `avrdude -h`
Testing the ISP communication (Ex 2.12)
- Find the right COMy port
  - Start list_cp2101uart.vbs
  - Open cmd shell and execute
    `avrdude -pm128 -cavrispv2 -P//./COMy`

Programming a pre-compiled application (Ex 2.13)
- Open a cmd shell in inss2007 on the BTnode CDROM
- Erase the flash memory:
  - Execute
    `avrdude -pm128 -cavrispv2 -P//./COMy -e`
  - Program bt-cmd into flash
    - Execute
      `avrdude -pm128 -cavrispv2 -P//./COMy -D -V -s -U
       flash:w:bt-cmd.btnode3.hex:i`
**Hands-on: Building and Uploading BTnut Applications**

- **The BTnut build process**
  - Automated with GNU make
    - Toplevel `Makefile` in `btnut_snap/app`
    - Global `Makerules` and `Makedefs` in `btnut_snap`
    - Can be overridden using environment variables

- **Building the `bt-cmd` application (Ex 2.16)**
  - Open cmd shell in `btnut_snap/app/bt-cmd`
  - Execute
    ```
    make btnode3
    ```
  - Define the serial port for programming (default is `/dev/ttyS0`)
    ```
    set BURNPORT=localhost/./COMy
    ```
  - Execute
    ```
    make btnode3 upload
    ```
Hands-on: bt-cmd – Simple Bluetooth Networking

- Simple terminal commands
  - bt – Bluetooth radio commands
  - led – toggle LED patterns
  - bat – get the battery status
  - nut – show OS system information
  - log – BTnut logging features
  - More information on commands by pressing 2x tab

🌟 Try out these terminal commands with **bt-cmd**
  - led on 3
  - nut threads

  - bt inquiry sync
  - bt rname XX:XX:XX:XX:XX:XX
  - Try to connect to other Bluetooth devices
Hands-on: bt-cmd – A Look Under the Hood

Open app/bt-cmd/bt-cmd.c in an editor

```c
/* \example bt-cmd/bt-cmd.c */

#include <hardware/btn-hardware.h>

int main(void) {

    btn_hardware_init();

    // hello world!
    printf("\n# ------------------------------------------------------\n");
    printf("\n# Welcome to BTnut (c) 2006 ETH Zurich\n");
    printf("\nbooting Bluetooth module...\n");

    // bluetooth module on (takes a while)
    btn_hardware_bt_on();

    // terminal init
    sprintf(prompt, "[bt-cmd@"SADDR_FMT"]$ ", SADDR(addr));
    btn_terminal_init(stdout, prompt);
    bt_cmds_init(stack);
    bt_cmds_register_cmds();
    btn_cmds_register_cmds();

    // terminal mode
    btn_terminal_run(BTN_TERMINAL_NOFORK, 0);

    return 0;
}
```
Outline

• Introduction
  – Basic concepts of embedded wireless sensor network platforms
  – Overview of the BTnode platform
    • Hardware architecture
    • BTnut system software

• Hands-on
  – Installing/getting to know the development tools
  – First steps in BTnode Programming
    • Plugging things together
    • Basic communication, ISP programming
    • The bt-cmd application – simple Bluetooth networking

  – My first BTnut multi-hop application
    • Bluetooth networking basics
    • Multihop networking
    • Sensor interfaces, packets and payload
    • Bluetooth Scatternet topology visualization
Now that you have an overview of BTnodes, the BTnut software and required tools we want to build an application

- Typical wireless sensor network application
  - Sampling on different sensors in a regular interval
  - Multi-hop networking
  - Debugging output
  - Data collection and visualization

- Hands-on exercises using an application template
• Bluetooth in a nutshell
  – Low-power, low-range personal communication
  – Frequency hopping spread spectrum
  – 2.4 GHz ISM band
  – 79 channels
  – 1 Mbit/sec data rate
  – 10-100 m range
  – Connection-oriented

• Many high-level features built in
  – Encryption, authentication
  – Error correction
  – Flow control

• Available on many consumer devices
Bluetooth Networking – Pico and Scatternets

- Communication organized in Piconets
  - Master-slave configurations
  - Up to 7 active slaves
  - 255 inactive (parked) slaves

- Piconets can be combined in Scatternets

- Four states
  - IDLE
  - MASTER
  - SLAVE
  - MASTERSLAVE

- Useful operations
  - inquiry() – find other nodes
  - connect() – open connection
  - roleSwitch() – change MS relation
  - sendData() – data transport

- Detailed capabilities/features vary across Bluetooth devices
Bluetooth Networking – Simple Scatternet Tree Construction

- A simple and effective approach
- Link layer connectivity
  - Random search and connect

```
loop {
  while (my_slaves < max_degree) do
    found_nodes = inquiry();
    forall nodes in found_nodes do
      connect();
  }
}
```

- Distributed coordination
  - Inquiry() and connect() operations can exhibit long delays
  - No a priori guarantee for success
  - Serialization of parallel processes

Bluetooth Networking – Host Controller Interface

- Standardized asynchronous, buffered packet interface
  - providing access to lower levels of the protocol stack

<table>
<thead>
<tr>
<th>HCI_COMMAND</th>
<th>HCI_EVENT</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>OGF</td>
<td>OCF</td>
<td>PARAMETERS</td>
</tr>
<tr>
<td>0</td>
<td>max. 64 Kbytes</td>
<td></td>
</tr>
</tbody>
</table>

Applications
- SDP
- RFCOMM
- L2CAP

Host Controller Interface

Audio
- Link Manager
- Baseband
- RF

Host processor

Physical interface

Bluetooth module
My First BTnut Application – The Bluetooth Protocol Stack
My First BTnut Application – The BTnut Protocol Stack

- Bluetooth Radio
- Baseband
- Link Manager Protocol
- Audio
- L2CAP connectionless
- L2CAP
- RFCOMM
- Connection Manager
- Multi-Hop
- RPC
- Code-Distribution
- Protocol Service Multiplexer
- Services with a PSM_ID
My First BTnut Application – The Host Controller Interface Layer

- Connection Manager
- Multi-Hop
- RPC

L2CAP connectionless

HCI - btstack

Manages connections to other Bluetooth devices
int main(void) {
    
}
Hands-on: Application Template Overview

A template `sensor-node.c` has been prepared for you

- Copy the folder `inss2007/mhop-example` into the folder `app`
- It contains `sensor-node.c`, `defs.h`, `Makefile` and solutions
- Open `app/mhop-example/sensor-node.c` in an editor
int main(void) {
    // hardware init
    btn_hardware_init();
    btn_led_init(1);

    // init terminal app uart
    u_long baud = 57600;  // serial baud rate
    NutRegisterDevice(&APP_UART, 0, 0);
    freopen(APP_UART.dev_name, "r+", stdout);
    _ioctl(_fileno(stdout), UART_SETSPEED, &baud);
    btn_terminal_init(stdout, "[senso] $ ");

    // hello message
    printf("\n# ------------------------------------------");
    printf("\n# Welcome to ASCI 2008 (c) ETH Zurich\n");
    printf("# program version: %s\n", PROGRAM_VERSION);
    printf("# ------------------------------------------");

    printf("\nbooting bluetooth module.\n");
    btn_hardware_bt_on();
    ...
}
```c
int main(void){
    ...
    struct btstack* bt_stack;
    bt_stack = bt_hci_init(&BT_uart);

    bt_acl_init(bt_stack, BT_HCI_PACKET_TYPE_DM3);

    ...
}
```

`sensor-node.c`
int main(void){
    ...
    struct btstack* bt_stack;
    bt_stack = bt_hci_init(&BT_UART);
    bt_acl_init(bt_stack, BT_HCI_PACKET_TYPE_DM3);
    bt_psm_t* psmux;
    psmux = bt_psm_init(bt_stack, MAX_NR_SERVICES,
                        NR_BUFFERS);
    l2cap_cl_init(bt_stack, psmux);
    ...
}
```c
int main(void) {
    ...
    struct btstack* bt_stack;
    bt_stack = bt_hci_init(&BT_UART);

    bt_acl_init(bt_stack, BT_HCI_PACKET_TYPE_DM3);

    bt_psm_t* psmux;
    psmux = bt_psm_init(bt_stack, MAX_NR_SERVICES, NR_BUFFERS);
    l2cap_cl_init(bt_stack, psmux);

    con_mgr_init(bt_stack, psmux, CM_PSM,
                 bt_hci_register_con_table_cb, CM_COD);

    ...
}
```

This initializes a simple tree connection manager.

PSM_ID for con_mgr service

sensor-node.c

Connection Manager | Multi-Hop | RPC
---|---|---
L2CAP connectionless
HCI - btstack
int main(void) {
    ...
    struct btstack* bt_stack;
    bt_stack = bt_hci_init(&BT_UART);

    bt_acl_init(bt_stack, BT_HCI_PACKET_TYPE_DM3);

    bt_psm_t* psmux;
    psmux = bt_psm_init(bt_stack, MAX_NR_SERVICES, NR_BUFFERS);
    l2cap_cl_init(bt_stack, psmux);

    con_mgr_init(bt_stack, psmux, CM_PSM,
                 bt_hci_register_con_table_cb, CM_COD);

    mhop_cl_init(bt_stack, psmux, MHOP_PSM,NR_BUFFERS,
                 con_mgr_register_con_table_cb);

    ...
}
int main(void) {
    ...
    struct btstack* bt_stack;
    bt_stack = bt_hci_init(&BT_UART);

    bt_acl_init(bt_stack, BT_HCI_PACKET_TYPE_DM3);

    bt_psm_t* psmux;
    psmux = bt_psm_init(bt_stack, MAX_NR_SERVICES,
                        NR_BUFFERS);
    l2cap_cl_init(bt_stack, psmux);
    con_mgr_init(bt_stack, psmux, CM_PSM,
                 bt_hci_register_con_table_cb, CM_COD);
    mhop_cl_init(bt_stack, psmux, MHOP_PSM, NR_BUFFERS,
                 con_mgr_register_con_table_cb);
    rpc_init(psmux, 8, RPC_PROC_PSM, RPC_RESULT_PSM);
    ...
}
**Hands-on: Visual Control – Debugging with LEDs**

- First steps with **sensor-node.c**
  - Understand the basic structure of the source code files
  - Compile, upload and inspect the state on the LEDs

<table>
<thead>
<tr>
<th>State</th>
<th>LED Pattern</th>
<th>LEDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>Heartbeat</td>
<td></td>
</tr>
<tr>
<td>No connection</td>
<td>4Bit-ID+Heartbeat</td>
<td></td>
</tr>
<tr>
<td>Inquiry</td>
<td>Knight-Rider</td>
<td></td>
</tr>
<tr>
<td>Connected</td>
<td>4Bit-Tree-ID</td>
<td></td>
</tr>
<tr>
<td>Connectivity with base station</td>
<td>Green LED</td>
<td></td>
</tr>
</tbody>
</table>
My First BTnut Application – Network Topology Visualization

Check the graph to see your BTnode pop up

# of Sensornodes: 3
Hands-on: Sampling a Single Sensor

- Attach the sensor board to the USBprog

- Extend `sensor-node.c` with a sensor
  - Sample one sensor (microphone) every 5 seconds
  - Print the value on the terminal

```c
#include <teco_ssmall/micsampler.h>
#include <dev/adc.h>

THREAD(sensorLoop, arg){
  // sensor thread
  ADCInit();
  mic_init();
  for(;;){
    NutSleep(5000);
    printf("mic value = \%u\n", mic_read());
  }
}

btn_hardware_io_power(1); // sensor board power
NutThreadCreate("T_sensor", sensorLoop, 0, 256)
```
Hands-on: Sampling Multiple Sensors

1. Sample all 3 sensors:
   - Microphone
   - Light
   - Temperature

2. Store the sensor values in a predefined data-structure

3. Print the sampled data on the terminal

★ Copy the code on the right into the template

```c
#include <dev/twif.h>
#include <teco_ssmall/tsl2550.h>
#include <teco_ssmall/tc74.h>

THREAD(sensorLoop, arg){
    TwInit(20); // init twi with slave addr

    while(tsl_init()) NutSleep(10000); // start light sensor

    ADCInit();
    mic_init(); // init mic

    sensor_data_t sensor_data; // see defs.h
    sensor_data_t* data = &sensor_data;
    u_char channel0, channel1;
    signed int temp;
    for(;;){
        NutSleep(5000);
        data->mic = mic_read(); // sample mic
        tsl_read(&channel0, &channel1); // sample light
        data->light = tsl_calculate_lux(channel0, channel1);
        tc_read(&temp); // read temperature
        data->temp = temp;
        printf("mic = %u, light = %d, temp = %u\n",
               data->mic,
               data->light,
               data->temp);
    }
}
```
**Hands-on: Multi-hop Data Transport**

Send the data-structure with the sensor values in a multi-hop packet to a base-station

- Address of the base station (note: big endian)
  ```c
  bt_addr_t sink_addr = {0x67, 0x00, 0x00, 0x3f, 0x04, 0x00};
  ```

- Send packet
  ```c
  mhop_cl_send_pkt((u_char*)data,
  sizeof(sensor_data_t),
  sink_addr,
  SENSO_PSM,
  MHOP_CL_UNICAST, MHOP_CL_TTL_INFINITE);
  ```

In **THREAD()** after `printf()`

Use the correct base-station address!
My First BTnut Application – Sensor Data Visualization

# of SensoNodes: 3

Sensor Values:

00:e5

00:6d

00:42
Data-callback prints sensor values on terminal:

```c
bt_acl_pkt_buf* sensor_data_cb(bt_acl_pkt_buf* pkt_buf,
    u_char* data, u_short data_len,
    u_short service_nr, void* cb_arg){
    u_char* source = mhop_cl_get_source_addr(pkt_buf->pkt);
    sensor_data_t* sensor_data = (sensor_data_t*) data;

    printf(":S "SADDR_FMT" %u %d %u\n", SADDR(source),
        sensor_data->mic,
        sensor_data->light,
        sensor_data->temp);

    return pkt_buf;
}
```

Data-callback is registered in main() with a PSM number:

```c
bt_psm_service_register(psmux, SENSO_PSM, sensor_data_cb, NULL);
```
Congratulations!
Outline continued...

• Demonstration
  – Debugging and profiling of sensor network applications
    • Embedded debugging
    • Profiling using and OS tracer
    • Testbeds – The ETH Deployment-Support Network

• Hands-on
  – Interfacing to handheld devices
    • Bluetooth RFCOMM
    • AT commands
    • Sending an SMS message using AT commands

• Question and answer

• Time to explore BTnodes...
Beyond the Blinking LED – Embedded Debugging

• LEDs are nice but they
  – Offer a limited view inside only (e.g. Mica2Dot with one LED only)
  – Lack context/timing information (order of events?)
  – Consume lot’s of power

• Other debugging techniques
  – Instruction-code simulators (with debugging capabilities)
  – In-circuit emulators (ICE)
  – Breakpoints with JTAG
  – printf() statements
  – Levels of verbosity, memory consumption, timing

  – Operating system monitors
Beyond the Blinking LED – The BTnut OS Tracer

- The tracer tool is an extension to the BTnut software
  - Storage of information about important OS events
    - Thread switches, interrupts
    - Type of event, system time, additional information
  - Retrieval of information for offline analysis at a later time
  - Runtime configurable
  - Must be enabled at compile time using `-DNUTTRACER`

```
[btnode]$trace oneshot TRACE mode ONESHOT, restarted
[btnode]$trace TRACE STATUS
Mode is ONESHOT
Size is 500
contains 77 elements
[btnode]$trace print 10
TRACE contains 500 items, printing 10 items.
TAG PC/Info Time[s:ms:us]
---------------------------------------------------
Thread Yield idle 13:524:336
Thread Sleep LED 13:524:604
Thread Yield idle 13:581:857
Thread Sleep LED 13:582:125
Thread Yield idle 13:639:392
Thread Sleep LED 13:639:659
Thread Yield idle 13:696:909
```
The Deployment-Support Network

- Deployment-Support Network
  - Temporary, minimal invasive
  - Virtual connections to nodes
  - Reliable, wireless, scalable

Deployment-Support Network – Architecture

DSN - Layer

DSN - Server
- Command Thread
- SQL-Module
- Parser
- XMLRPC
- Rtx-Module

DSN - Node

Target - Layer

GUI’s & Analysis Tools

Target - Node

GUI
AT

Programming
Bit Control
Ascii String
• Characterization and validation of Smoke Detectors
  – Project in cooperation with Siemens Building Technologies
Deployment-Support Network – Analysis

Packet Reception Chart

Link Quality Chart

RSSI Histogram

Bit Errors / RSSI

NCCR MICS
National Competence Center in Research
Mobile Information and Communication Systems

Start
Outline continued...

- **Demonstration**
  - Debugging and profiling of sensor network applications
    - Embedded debugging
    - Profiling using and OS tracer
    - Testbeds – The ETH Deployment-Support Network

- **Hands-on**
  - Interfacing to handheld devices
    - Bluetooth RFCOMM
    - AT commands
    - Sending an SMS message using AT commands

- **Question and answer**

- **Time to explore BTnodes...**
After creating a wireless sensor network we want to connect to a cellular phone as gateway.

- Interfacing a sensor network to a mobile gateway
  - Internet connectivity
  - Alerting in case of events/failures
  - Independence of infrastructure

- Hands-on exercises using an application template
Interfacing to Handheld Devices – The Bluetooth Protocol Stack

- Bluetooth Radio
- Baseband
- Bluetooth Radio
- Link Manager Protocol
- RFCOMM
- TCP/IP
- OBE
- PPP
- AT Commands
- TCS
- L2CAP
- SDP
- Application
- HCI
- Audio
- Application
- HCI
Interfacing to Handheld Devices – The Bluetooth Protocol Stack

- Bluetooth Radio
- Link Manager Protocol
- Baseband
- Bluetooth Radio

- Modem Command Set
- AT Commands
- RFCOMM
- L2CAP

- Application
- Serial Port Emulation
- Protocol Service Multiplexer

- HCI
- Audio

NCCR MICS
National Competence Center In Research
Mobile Information and Communication Systems
Interfacing to Handheld Devices – The Bluetooth Protocol Stack

- AT Commands
  - RFCOMM
    - L2CAP
      - HCI - btstack
        - HCI
Interfacing to Handheld Devices – Template Overview

```c
int main(void) {
}
```
A template `send-sms.c` has been prepared for you

- Copy the folder `inss2007/send-sms` into the folder `app`
- It contains `send-sms.c`, Makefile and solutions
- Open `app/send-sms/send-sms.c` in an editor
int main(void) {
    // hardware init
    btn_hardware_init();
    btn_led_init(1);

    // init terminal app uart
    u_long baud = 57600;  // serial baud rate
    NutRegisterDevice(&APP_UART, 0, 0);
    freopen(APP_UART.dev_name, "r+", stdout);
    _ioctl(_fileno(stdout), UART_SETSPEED, &baud);
    btn_terminal_init(stdout, "[senso]$" cpp);

    // hello message
    printf("\n# -----------------------------------------

# Welcome to BTnut(c) 2007 ETH Zurich\n"
);  
    printf("# send-sms version: %s\n", PROGRAM_VERSION);
    printf("# configured for gateway BT MAC: ");
    printf("%.2x:%.2x:%.2x:%.2x:%.2x\n", Btaddr ... );
    printf("# channel %x\n", channel);
    printf("# sending SMS to phone number %.15s\n", number);

    ... }
```c
int main(void) {
    ...

    // bluetooth module on (takes a while)
    btn_hardware_bt_on();

    // Start the stack and let the initialization begin
    stack = bt_hci_init(&BT_UART);
    bt_hci_write_local_cod(stack, BT_HCI_SYNC, 200);
    printf_P(PSTR("ok.\n\r"));

    // give hint
    printf_P(PSTR("hit tab twice for a list of commands\n\r"));

    // terminal init
    btn_terminal_init(stdout, "[send-sms]$");
    bt_cmds_init(stack);

    ...
}
```
int main(void) {
    ...

    // Start L2CAP

    l2cap_stack = bt_l2cap_init(stack, 8, 8, BT_L2CAP_HCI_PACKET_TYPE);

    l2cap_cmds_init(l2cap_stack, 1, BT_L2CAP_MIN_MTU, BT_L2CAP_MTU_DEFAULT);

    ...
}

send-sms.c
```c
int main(void){
    ...

    // Start RFCOMM
    rfcomm_stack = bt_rfcomm_init(l2cap_stack, BT_RFCOMM_DEF_MFS, 4, 5);
    rfcomm_cmds_init();
    ...
}
```
```c
int main(void) {
    ...

    // Start RFCOMM

    rfcomm_stack = bt_rfcomm_init(l2cap_stack, BT_RFCOMM_DEF_MFS, 4, 5);

    rfcomm_cmds_init();

    bt_cmds_register_cmds();
    btn_cmds_register_cmds();
    nut_cmds_register_cmds();
    l2cap_cmds_register_cmds();
    rfcomm_cmds_register_cmds();

    ...
}
```
```c
int main(void) {
    ...

    // Start RFCOMM
    rfcomm_stack = bt_rfcomm_init(l2cap_stack,
                               BT_RFCOMM_DEF_MFS, 4, 5);

    rfcomm_cmds_init();

    bt_cmds_register_cmds();
    btn_cmds_register_cmds();
    nut_cmds_register_cmds();
    l2cap_cmds_register_cmds();
    rfcomm_cmds_register_cmds();

    btn_terminal_register_cmd("sendsms", send_sms);

    // terminal mode
    btn_terminal_run(BTN_TERMINAL_NOFORK, 0);
    return 0;
}
```
```c
void send_sms(char* arg) {

    // the message
    char* message = NutHeapAllocClear(161);
    strcpy(message, "Greetings from the BTnode.");

    // connect
    printf("connecting to ["ADDR_FMT"]\n", ADDR(BTaddr));
    bt_rfcomm_start_session(BTaddr, 0, 0)
    NutSleep(1000);  
    bt_rfcomm_connect(channel, con_cb, rcv_cb, line_cb, credit_cb, 10, NULL)
    NutSleep(1000);

    ...

}
```

RFCOMM session to the cell phone

RFCOMM channel setup: Callbacks necessary for the connection, reception, line status and RFCOMM credit
void send_sms(char* arg) {
    ...

    // general AT Commands
    bt_rfcomm_send(2*channel, "at&f\r", 5);
    NutSleep(2000);
    bt_rfcomm_send(2*channel, "at+cgmi\r", 8);
    NutSleep(1000);
    bt_rfcomm_send(2*channel, "at+cgmm\r", 8);
    NutSleep(1000);
    bt_rfcomm_send(2*channel, "at+cgsm\r", 8);
    NutSleep(1000);

    // send sms
    send_sms_pdu_mode(number, message);

    // disconnect
    bt_rfcomm_disconnect(2*channel);
}
Interfacing to Handheld Devices – Configuration

```c
// configure this!
// bt-address of the phone (reverse order!)
bt_addr_t BTaddr = {0x1d, 0x34, 0x05, 0x13, 0x18, 0x00};

// destination number to which sms is sent to
char number[17] = "+41774325509";

// destination channel number
u_char channel = 2;

void send_sms(char* arg){
    ...
}

int main(void){
    ...
}
```
Congratulations!


