BTnode System Development

Jan Beutel
Computer Engineering and Networks Lab
March 13, 2003
Hardware Requirements

Autonomous wireless communication and computing platform based on a Bluetooth radio module and a microcontroller.

Requirements:
- Small form factor, low component count
- Standardized wireless interface
- Flexible and cost effective deployment of large quantities of networking nodes
**System software architecture**

Lightweight OS made up of **device drivers** and **dispatcher for scheduling**

- Low level interrupt driven drivers and libraries for peripherals and interfaces
- Event driven application model facilitates coarse grained cooperative multithreading
- 30 kB code size in ROM, 1-2 kB in RAM (with 128 byte UART buffers)
System software

• **Sort of OS**
  – access to hw resources (drivers)
  – cooperative multitasking

• **but**
  – is a library completely (libBTNsys.BTNODE.a)
    • linked with application code into one executable
  – one program per BTnode

• **no sequential programming**
  – wait(100);
  – while(1) { ... };

• **strictly event driven!**
  – register interest in events
  – provide an event handler (i.e., a function)
Read 100 bytes: any good?

```c
int num = 0;
while( num < 100 ) {
    num += btn_uart1_read( app_buf); }
// your business
```

Energy?

Other drivers?

Bluetooth

UART1

I2C
void handler(/* ... */) {}
void main() {
    btn_disp_ev_reg(RECEIVE_EV, handler, 0);
    btn_disp_run();
}
Dispatcher (2)

```c
void handler(/* ... */) {}
void main() {
    btn_disp_ev_reg(RECEIVE_EV, handler, 0);
    btn_disp_run();
}
```

**Diagram:**
- **Application:** `btn_disp_ev_reg`, `btn_disp_run`
- **Driver:** `read buffer`, `received_irq`, `EVENT, handler func RECEIVED_EV, handler() OTHER_EV, func2()`, `Dispatcher`
- **Hardware**
Dispatcher (3)

```c
void handler(/* ... */) {}

void main()
{
    btn_disp_ev_reg(RECEIVE_EV, handler, 0);
    btnDisp_run();
}
```

![Diagram of Dispatcher](image)
void \textbf{handler}( /* ... */ ) {}
void \textbf{main}() {
    \textbf{btn\_disp\_ev\_reg}( \textbf{RECEIVE\_EV}, \textbf{handler}, 0 );
    \textbf{btn\_disp\_run}();
}
Better than a sequential model but not yet perfect

void handler() {
    static int num = 0;
    num += btn_uart1_read(app_buf);
    if( num >= 100 ) {
        /* your business*/
    }
}

void main() {
    // ...
    btn_disp_ev_reg(RECEIVE_EV, handler, 0);
    btn_disp_run();
}
Getting going on BTnodes ...

• Documentation available on the Web
  – Design documents
  – API documentation
  – Links
  – Mailing list

• Software development tools
  – Gnu toolchain
  – avr-gcc, avr-libc, downloader, gdb
  – Available for Linux, Windows and Mac OS X

• BTnode System Software
  – Snapshot packages and CVS repository
Required hardware

- In system programmer
  - Atmel AVR ISP
  - or STK500

- RS-232 level converter

- Some cables
  - Programming cable
  - UART cable
  - 2 x serial 9 pin

- A BTnode
Hardware setup: ISP and UART

- ISP (J1)
- ISP 6 PIN
- UART
- UART cable
- Level converter
- serial cable
- serial cable
- Programmer
- STK 500/AVR ISP
- programming cable
Developer debugging support

- Serial programming on UART using ISP
- Reset button
- 4 LED’s
- JTAG interface
- Serial console for debugging output `printf()`
- Example programs in the BTnode system software

- Linux emulation of BTnode system

- Bootloader resident in Flash memory
  - Remote update of the bootcode via Bluetooth link
  - Use of extra SRAM blocks as cache
Linux Emulation mode

- Same software as on the embedded target
- Access to files
How fast is it to get going?

Smart Its Hackfest

SW install \( \frac{1}{2} \) h
First steps on `helloworld.c` \( \frac{1}{2} \) h
Own program \( \frac{1}{2} \) h

Under 2 hours for 15 newcomers this November in Zurich
BTnode hardware details

- Atmel ATmega 128l MCU
  8-Bit RISC
  (max. 8 MHz ~8MIPS)
- Real time clock
- 128 kB Flash ROM
  64 kB SRAM
  4 kB EEPROM
- Generic sensor interfaces
- UART and I2C data interface
- Power and frequency management
- Ericsson Bluetooth radio
- Integrated PIFA antenna