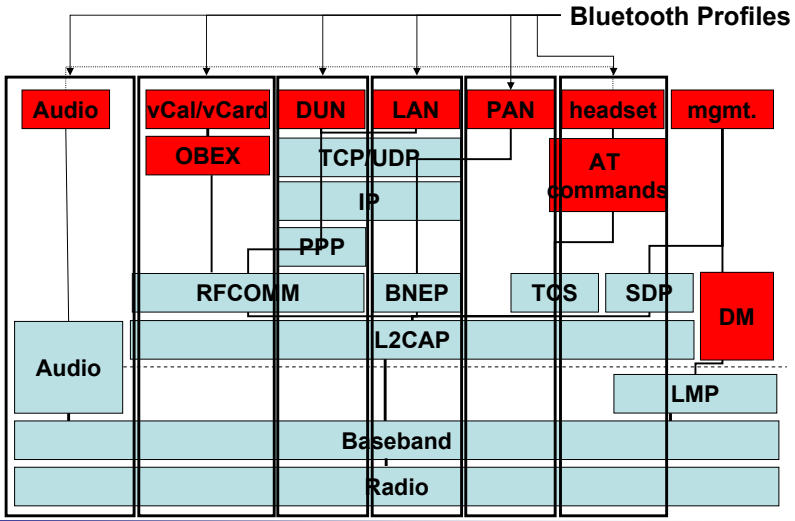
Embedded Internet
WS 02/03

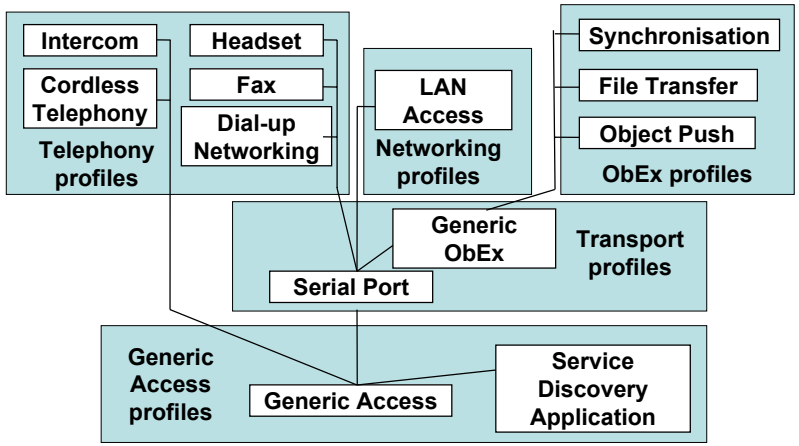
Bluetooth:
Application View

Application Framework



Profile Dependencies

- Profile Dependency Structure (incomplete list):



Generic Access Profile (GAP)

- Basis for all higher profiles
- Connectivity policies
  - Discoverability modes (“when to enter what inquiry scan mode”)
    - General discoverable mode:
      Inquiry scans at the General Inquiry Access (GIAC) code
      ⇒ discoverable for all
    - Limited discoverable mode:
      Inquiry scans at the Limited Inquiry Access (LIAC) code
      ⇒ future use: look only for headsets, …
    - Nondiscoverable mode:
      No Inquiry scans (does NOT mean: no page scans!)
      ⇒ not discoverable

Generic Access Profile (II)

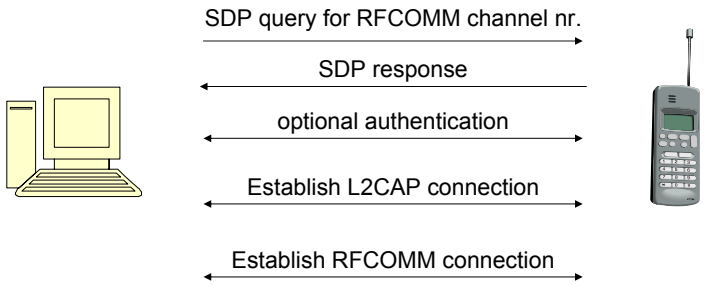
- Security modes:
  - Security mode 1:
    “open door”, no access control
  - Security mode 2:
    L2CAP connection from outside possible,
    afterwards authentication (using PINs etc.) necessary
  - Security mode 3:
    Link Manager handles security checks before L2CAP connection is set up completely

Service Discovery Application Profile (SDAP)

- Service Discovery Protocol defines primitives like
  - SDP_ServiceSearchRequest
  - SDP_ServiceAttributeRequest
  - …
- Does NOT define usage of higher layer service requests
- SDAP provides usage model
  - serviceBrowse
  - serviceSearch
  - enumerateRemDev
  - terminatePrimitives
    (SDP is connectionless, SDAP provides “session”

Serial Port Profile (SPP)

- Establish RFCOMM communication between devices, emulate serial port (e.g. “COM 3”)



SDP – human readable

```
ServiceClassIDList
{
  uuid SerialPort
}

ProtocolDescriptorList
{
  {
    uuid L2CAP
  }
  {
    uuid RFCOMM
    uint8 0x12
  }
}

ServiceName "Cable test"
```

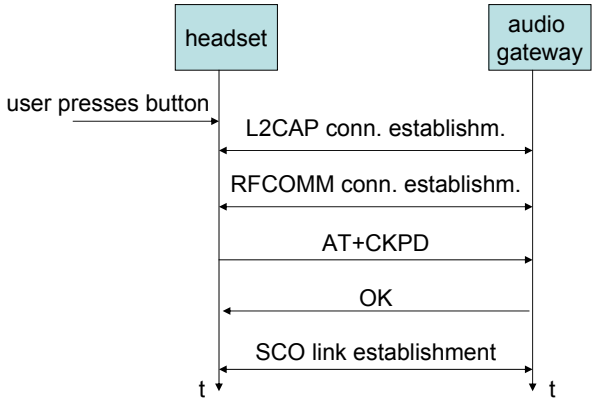
SDP – machine readable

```
0x09, /* ServiceClassIDList(0x0001) */
  0x00,
  0x01,
0x35, /* DataElSeq 3 bytes */
0x03,
  0x19, /* uuid SerialPort(0x1101) */
  0x11,
  0x01,
0x09, /*
    ProtocolDescriptorList(0x0004) */
  0x00,
  0x04,
0x35, /* DataElSeq 12 bytes */
0x0c,
  0x35, /* DataElSeq 3 bytes */
  0x03,
    0x19, /* uuid L2CAP(0x0100) */
    0x01,
    0x00,
```

```
0x35, /* DataElSeq 5 bytes */
  0x05,
    0x19, /* uuid RFCOMM(0x0003) */
    0x00,
    0x03,
    0x08, /* uint8 0x12 */
      0x12,
0x09, /* ServiceName(0x0100) =
    "Cable test" */
  0x01,
  0x00,
0x25, /* String length 10 */
0x0a,
  'C','a','b','l','e',' ','t','e','s','t',
```


Headset profile

- Handle audio connection setup and management between headset and “audio gateway”



AT commands

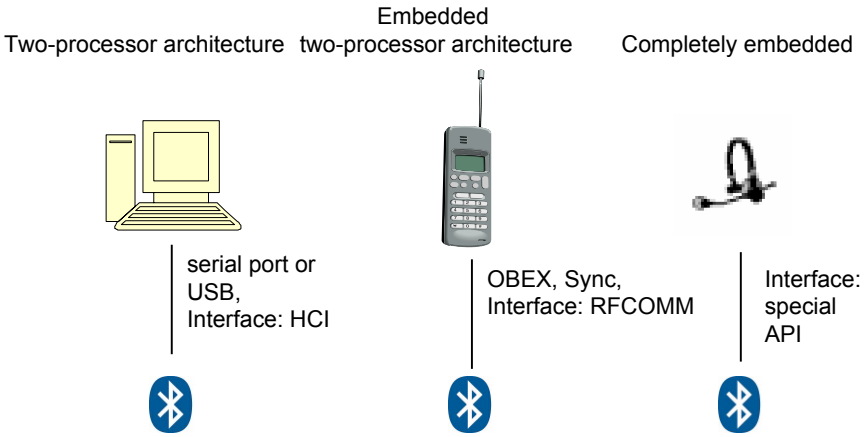
- AT: “Attention”
  - ASCII command sequences for communication modem <-> PC
  - Bluetooth extensions:
    - AT+VGM=<value>: microphone gain
    - AT+VGS=<value>: speaker gain
    - …
  - Proprietary extensions:
    - AT+xxx
    - Universal remote control (e.g. PcControl at www.christersson.org for Ericsson)

Profile List

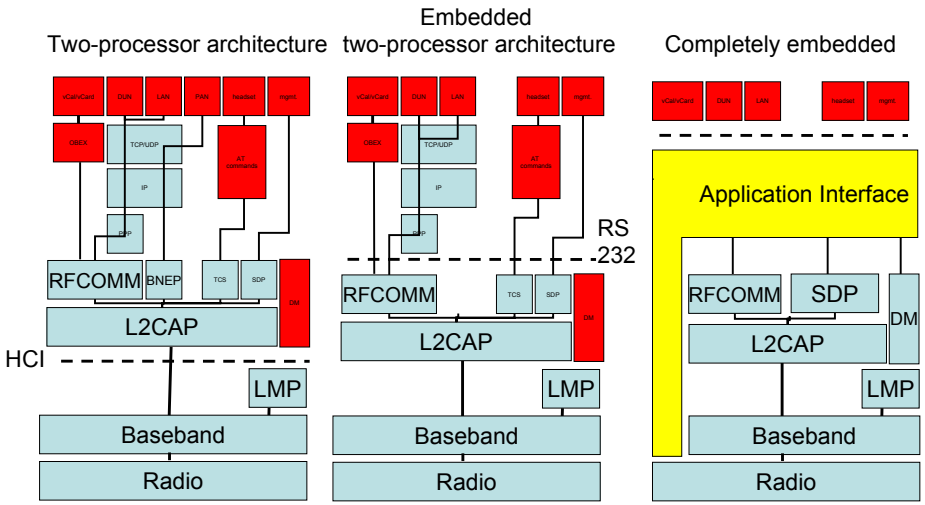
- Generic Access Profile
- Service Discovery Application Profile
- Cordless Telephony Profile
- Intercom Profile
- Serial Port Profile
- Headset Profile
- Dial-up Networking Profile
- Fax Profile
- LAN Access Profile
- Generic Object Exchange Profile
- Object Push Profile
- File Transfer Profile
- Synchronization Profile

New Profiles:
Advanced Audio Distribution
PAN (using BNEP)
Audio Video Remote Control
Basic Printing
Basic Imaging
Extended Service Discovery
Generic Audio Video Distribution
Hands Free
Hardcopy Cable Replacement

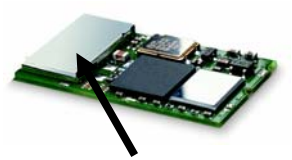
Deployment



Deployment



Two-processor architecture (ex.)



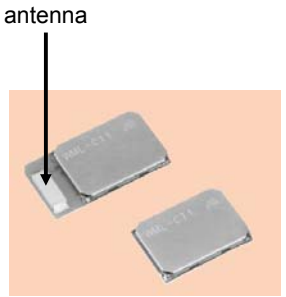
good old Ericsson module

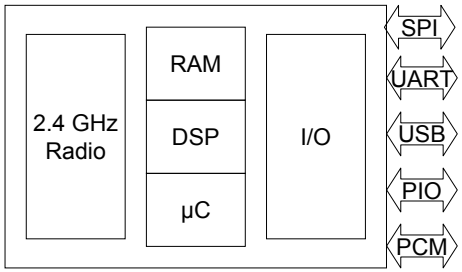
CSR-chip based Acer module

CSR: Cambridge Silicon Radio, http://www.csr.com

Completely Embedded Approach

- Example: CSR-chip based Mitsumi module





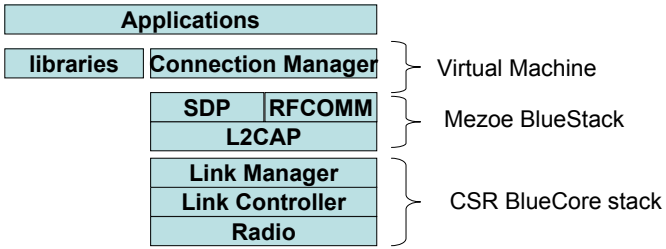
Overview

- Hardware architecture
- OS abstractions
  - Scheduling
  - Tasks
  - Messages
  - Events
- Development
  - Toolchain
  - PS-keys

Hardware Architecture

"Virtual Machine"

- Runs on the same processor as the Bluetooth protocol stack
- Provides runtime environment for user-defined application
- Protects protocol stack from malicious code
  ⇒ qualification issue!



OS abstractions: Scheduling

- Scheduling:
  - Virtual machine restricts processing time of user-defined applications
  - Stack priority, round-robin between user-defined apps
- Example:
  - Init variables, connection manager, callback functions, …
  - Either:
    Loop endlessly

    or:
    Call Sched();
    Wait for events

OS abstractions: Tasks, Messages

- Tasks with simple task id's:

```
DECLARE_TASK(1)
{}
```

- Predefined task ID's:
  - 0: Connection Manager (fixed)
  - 1: Application Framework (if used)

- Send messages between tasks

```
msg=MessageCreate(msg_type, length)
memcpy (msg, data, length)
MessagePut(ID,msg)
resp.  msg=MessageGet(ID, &type)
```

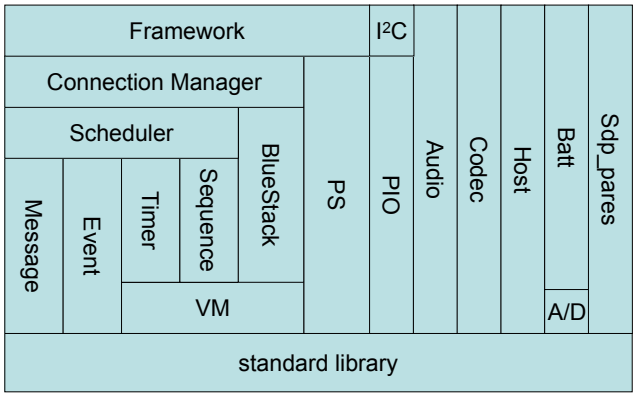

OS abstractions: Events

- Events for asynchronous notification

```
EventEnable(EVENT_TYPE);
for(;;)
{
VmWait();//wait for an event
if(EventCounter(EVENT_TYPE))
    {do_sthg();}
}
```

- I/O ports generate events (interrupts)
  - Set direction of I/O ports
  - Special library for PIO (programmable I/O) ports

Libraries

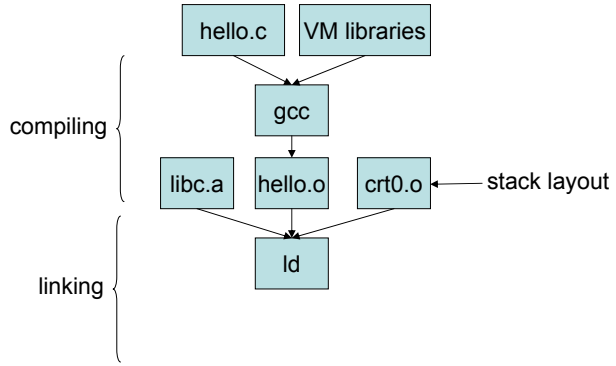


Connection Manager

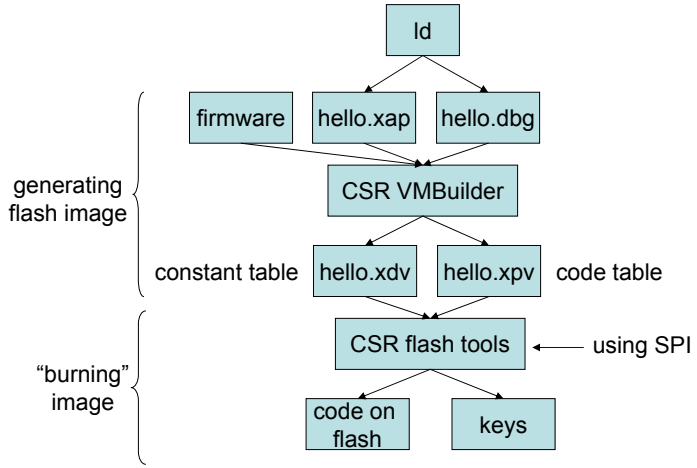
- High-level interface for connection setup, device inquiry, data transmission
- Steps:
  - Init
  - Open
  - Inquiry
  - Pair
  - Connect as master / as slave
- Data transfer
  - Message/Event-based: CM_DATA_REQ / CM_DATA_IND
  - Stream abstraction: StreamRfcommSource, StreamRfcommSink, …

Toolchain

- Chain of tools for compiling applications
  - Cross-compiling: compile on a PC, run on a different platform



Toolchain (cont'd)



Persistent Storage (PS keys)

- Set of keys for system settings (baud rate, BD_ADDR, …) (cf.: Windows registry)
- Windows tool: PStool.exe
- Ps library from the user-defined application

Further Reading

- D. Kammer, G. McNutt, B. Senese, J. Bray: Bluetooth Application Developer's Guide. Syngress Publishing, 2002
- Cambridge Silicon Radio: www.csr.com

Bluetooth under Linux

- Different stacks available, currently in the mainstream kernel: BlueZ implementation, available at http://bluez.sourceforge.net
- USB device:
  - *modprobe hci_usb*
  - *hciconfig hci0 up; hciconfig hci0*
- Tools:
  - *Hcitool, sdptool*
  - *l2ping*
  - *l2test, scotest*
  - *hcidump* (like *tcpdump*)
- Bluetooth socket extensions: L2CAP sockets, RFCOMM sockets

Assignment: Embedded part

- Go to C:\Program Files\BlueLab25\apps\test_cable (don't use BlueLab26; and don't use HWP3 ☺)
- The program is already running on the board: It accepts one RFCOMM connection and forwards incoming data to the serial port out of the board
- To compile and run, connect the parallel port cable, open a cygwin command window, go to the same directory, then type make bc01 (make bc02 will NOT work!)

Your job:

Try to understand what is going on in main.c, test the transmission range in connection with an iPAQ, try to connect as master, …

Assignment: Linux part

- Connect the serial port for debugging:
  - Use for example minicom
  - Settings: 115,2 kbit/s, 8N1, no flow control
- modprobe l2cap
- modprobe rfcomm
- sdptool search SP
  searches for available Serial Port Profiles
- Set up RFCOMM connection:
  go to bluez/bluez-pan-1.1pre4/dund (DUN daemon)
  try to understand main.c, compile it typing "make"
  start with ./dund -c "00:A0:96:1F:BC:50" -C 1 (example!!!)

Your job:

Transfer data from RFCOMM to the module and back over the serial port