Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

BTnode Peripherals

Semester Thesis SA-2006-12
Summer Term 2006

July 7, 2006

Computer Engineering and Networks Laboratory (TIK)
ETH Zurich, D-ITET

Author:
Philipp Stadelmann, philista@ee.ethz.ch

Advisors:
Matthias Dyer, dyer@tik.ee.ethz.ch
Dr. Jan Beutel, beutel@tik.ee.ethz.ch

Professor:
Prof. Lothar Thiele, thiele@tik.ee.ethz.ch

Abstract

Today wireless ad-hoc networks have become a growing field of research. For this purpose, the Computer Engineering and Networks Laboratory (TIK) and the Research Group for Distributed Systems have developed a sensor node of their own at the ETH Zurich - the BTnode [5]. The BTnode is an autonomous wireless communication and computing platform based on a Bluetooth radio and a microcontroller. It serves as a demonstration platform for research in mobile and ad-hoc connected networks (MANETs) and distributed sensor networks. The BTnode runs on its own system software which is a SourceForge project [4]. In this work the BTnode's ability to connect to peripheral devices is shown considering two examples. In part one the node dispatches an SMS message via a cell phone working as gateway. In part two the node controls a CMOS VGA camera and receives images.

Preface

This semester thesis is part of my graduate study at the Department of Information Technology and Electrical Engineering (D-ITET) at the Swiss Federal Institute of Technology (ETH).

I would like to address my sincere thanks to my advisor Matthias Dyer for his support and guidance as well as for the valuable feedback and tips during this project. Many thanks also to my co-advisor Jan Beutel for his support. Special thanks go to Mustafa Yuecel who was always helpful and gave me important hints on the BTnut system software.

My gratitudes go to the Computer Engineering and Networks Laboratory, especially to Prof. Lothar Thiele for giving me the opportunity to do this thesis.

Zurich, July 7, 2006

Philipp Stadelmann

Contents

Chapter 1

Introduction

Today's world relies more and more on electronic devices. Since the boom of the computer industry many applications have been developed to support us in our everyday life. Furthermore, with the development of the Wireless Technology completely new implementations are provided. Together with consumer electronics such as phones and PDAs and many more also other applications make use of this technology.

For example networks of sensor nodes can be established quickly and easily. The sensor nodes collect data at regular intervals or at specific events and send them to a base station where they are evaluated. However some sensor nodes may not have direct contact to the base station. Therefore they have to send their data via other nodes to the base station. The nodes have to establish a so called mobile ad-hoc network which is a growing field of research.

For this purpose, the Computer Engineering and Networks Laboratory (TIK) and the Research Group for Distributed Systems have developed a sensor node of their own at the ETH Zurich - the BTnode [5]. The BTnode is an autonomous wireless communication and computing platform based on a Bluetooth radio and a microcontroller. It serves as a demonstration platform for research in mobile and ad-hoc connected networks (MANETs) and distributed sensor networks. The BTnode runs on its own system software which is a SourceForge project [4].

In this work the BTnode's ability to connect to external devices is shown. Therefore two sample applications are presented. Chapter 2 explains how the BTnode can establish a connection to a cell phone over a Bluetooth link. Chapter 3 describes how to control a serially connected CMOS VGA camera module with the BTnode. With these two applications the versatility of the BTnode is pointed out.

Chapter 2

Mobile Phone

2.1 Introduction

There are several computer programs such as floAt's Mobile Agent [14] allowing users to control their cell phone with the computer or vice versa on the basis of a serial connection. The whole functionality of the phone is at one's disposal: Address books can be synchronized, phone books exchanged, SMS messages sent, to name only a few. On the other hand, the phone can control the computer's mouse, adjust the speaker volume, and much more. The connection between the phone and the computer can be established via cable e.g. USB or via Bluetooth.

The BTnode is equipped with a Zeevo ZV4002 Bluetooth radio. Therefore it should also be possible to connect to a phone and make use of its functions. This could for example be useful in sensor network applications where some sensor nodes acquire data to which specific actions have to be taken. The sensor node would then be able to send an SMS message to inform about the event that just happened.

In this chapter a solution is presented how to make the BTnode create a connection to a cell phone over Bluetooth and have the phone send an SMS message. In section 2.2 the necessary basics are explained. In section 2.3 the approach and implementation is depicted. Last but not least, in section 2.4 the results are discussed.

2.2 Theory

In this section the protocols needed to connect to a cell phone and send an SMS message are described. Figure 2.1 shows the Bluetooth protocol stack. The relevant layers are the RFCOMM layer described in subsection 2.2.1, the AT Commands layer discussed in subsection 2.2.2 and the application layer depicted in subsection 2.2.3.

2.2.1 RFCOMM

RFCOMM stands for *Radio Frequency Communication*. It is a simple transport protocol that emulates a serial interface (RS232) over an L2CAP link. Up to 60 simultaneous connections between two Bluetooth devices are supported. If a connection is established a simple terminal application can be opened to that interface to send data. Everything typed into the terminal is transmitted to the other end of the RFCOMM link.

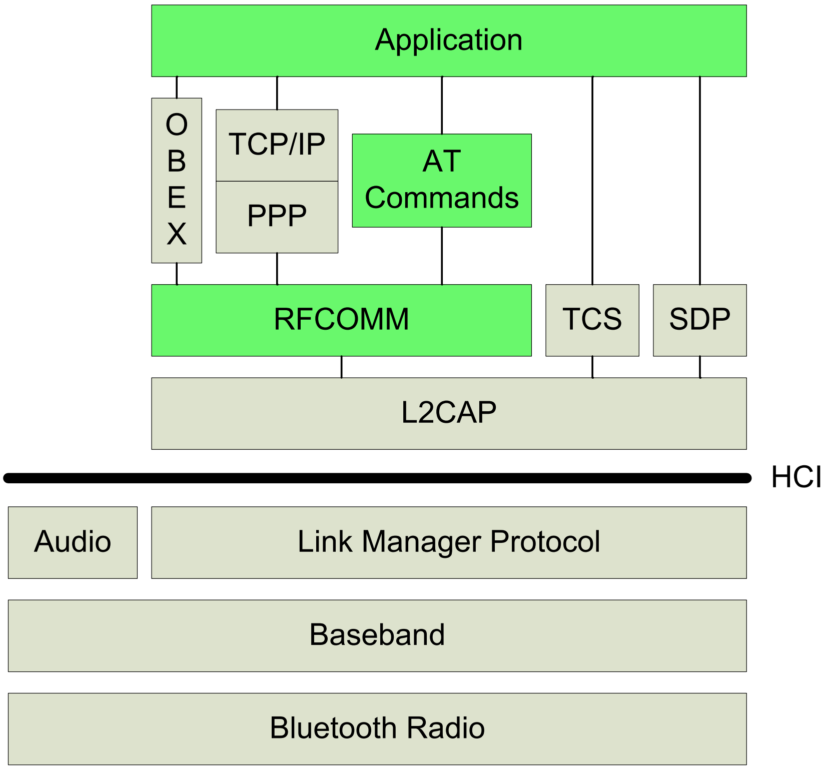


Figure 2.1: Bluetooth protocol stack

When establishing a connection between two devices for the first time, a secret key has to be exchanged. This procedure is called pairing. It is a security mechanism built into Bluetooth to prevent unauthorized connections. At the first connection attempt, connection keys of 128bit length are generated out of the Bluetooth addresses of the devices and some random number. These keys are then stored for further interaction. To safely transmit the connection keys at this very first connection set up, an initializing key has to be generated which in turn is calculated out of some random number, one of the Bluetooth addresses and the Bluetooth PIN-code.

If for example a connection is being established from cell phone A to B, then first user A is asked to enter a PIN-code (can be arbitrary) to its phone. This code is then sent to B's phone. B is asked to enter a PIN-code as well which is sent back to A's phone. If the two PIN-codes match, then a trusted pair is formed.

2.2.2 AT Commands

On top of the RFCOMM layer, the AT Commands are specified. These commands can be sent over an already established RFCOMM link. Originally, the AT Commands were developed as a specific programming language for dial-up modems. Back in the early days of Microprocessors when the Apple II was booming, users had to dial the phone manually and use an acoustic coupler for modem connection. Although internal modems did not have this shortage, they lacked the ability of being universal, since a different hardware design was needed for every computer bus. A more modular approach was an external modem connected to the widely available RS232 interface. It was then, when Dale Heatherington came up with the trailblazing idea to develop an external modem that was able to receive commands over the RS232 data line. Hence the Hayes Command Set or AT Commands were created.

Mobile phone manufacturers in one way or another have adopted this command set for the built-in modems in cell phones. Those modems can be accessed via Bluetooth, Infrared, USB cable or RS232 cable connection. Most of a cell phone's basic functionality AT Commands e.g. sending an SMS message are specified in [7] and standards referenced in there. However there are also vendor specific commands. A complete command set can usually be found in the developers guidelines of the manufacturer e.g. [9], [10] or [12].

The standard AT Command format consists of the Command itself followed by a carriage return. However some commands such as the SMS send command require a special line delimiter. Four different types of commands exist [13]:

The Set Format It is used to change settings of the mobile phone.

AT<command>=<parameters>**<CR>**

where	AT	notifies the built-in modem that a command is being entered
	<command>	the name of the command being entered
	<parameters>	the values to be used by the command
	<CR>	the carriage return `"\r"`

The Execute Format It is similar to the Set Format but the Execute Format usually does not require any parameters and is used to obtain information about the mobile phone.

The Read Format It is used to read current settings.

AT<command>?<CR>

Command Help Checks whether the command is available and returns the range of the parameters.

AT<command>=?<CR>

As already mentioned, AT Commands can be treated just like data packets that have to be sent over an RFCOMM channel. Therefore control over the phone can be gained as follows:

1. Open an RFCOMM link to the phone
2. Start a terminal application on that interface
3. Type in the desired AT Commands to control the phone

2.2.3 Application

As stated before, the target application is sending an SMS message. The AT Commands needed are specified in [6]. There are two ways to ways to send SMS messages using the AT Commands. On the one hand, there is the simple SMS text mode [11] where you can send the message as plain text:

1.	`AT+CMGF=1`	Set to text mode.
2.	`AT+CMGS="<phone number>"`	Send the recipient's phone number in international format i.e. +41…
3.	`<message>`	Send the message followed by the special line delimiter defined as `0x1A` in the ASCII code.

On the other hand, there is the more complicated protocol data unit (PDU) mode. The PDU for SMS messaging is assembled as follows.

`00 25 00 0B 91 14 77 14 36 21 F6 00 00 0B 42 AA FB 4D 2E 83 A6 CD A9 0B`

- length of the SMS-carrier address: `00` selects the number stored on the phone's SIM card.
- message flags: use `25`
- message reference number: `00` lets the phone set the reference number.
- length of the destination address (number of digits in hex format): `0B`
- format of the destination address: use `91` for international format i.e. `+41`…
- destination address: each digit of the phone number represents half of a Byte. Therefore if the length of the phone number is odd, a trailing `F` has to be added to complete the last Byte. The destination address is generated out of the phone number by flipping every Byte's lower and upper half. So the destination address from the example represents the phone number `41774163126`. The `+` sign is omitted.
- protocol identifier: use `00`
- data coding scheme: use `00`
- length of the original message: this is the number of characters (at most 160) of the message string including spaces in hex format.
- encoded message: the original message is coded using a 7bit ASCII character set. The stream of 7bit characters is then encoded into a Byte stream to form the encoded message. The coding scheme is depicted in the following formalism.

Definitions

length of message string	n
element of message string	$k, \quad 0 \le k \le n-1$
character k	$\mathbf{X} = X_6 \dots X_0$
character $k+1$	$\mathbf{Y} = Y_6 \dots Y_0$

Encoding

`if` $(k+1) \bmod 8 \neq 0$	$\underbrace{\underbrace{Y_{k \bmod 8} \dots Y_0}_{k \bmod 8+1[\text{bit}]}\underbrace{X_6 \dots X_{k \bmod 8}}_{7-k \bmod 8[\text{bit}]}}_{1\text{Byte}}$
`else`	`NULL`

Using this coding scheme on the message “BTnode SMS.” should result in the Byte stream shown in the example.

The PDU is of type string i.e. address lengths for example have to be converted to hex strings. The message is sent in PDU mode using the following commands:

1.	`AT+CMGF=0`	Set to PDU mode (set by default).
2.	`AT+CMGS=<PDU length>`	Number of Bytes of the PDU minus one since the leading `0x00` does not count. In the example it were `AT+CMGS=23`.
3.	`<PDU>`	Send the PDU followed by the special line delimiter defined as `0x1A` in the ASCII code.

2.3 Implementation

2.3.1 Approach

To find out how the different Bluetooth layers interact they were tested on a computer running Debian Linux. A state of the art USB Bluetooth dongle was used as interface together with BlueZ, the official Linux Bluetooth protocol stack. Useful commands are:

`hcitool` Provides the Host Controller Interface. With the command `hcitool scan` users can inquire for active Bluetooth devices and find out their names and Bluetooth addresses.

`hcidump` Creates debugging output. Use `hcidump -X` to display every packet’s data in hex and ASCII.

`rfcomm` Provides the functionality of the RFCOMM layer. With the command `rfcomm connect [Bluetooth Address] [Channel]` a connection to the device specified with the Bluetooth Address can be opened on channel `Channel`. It is closed with `CTRL-C`.

After successfully creating an RFCOMM connection to the cell phone the terminal application `minicom` was linked to this interface (`minicom /dev/rfcomm0`). It was then possible to control the phone by means of the AT Commands. They can simply be typed into the terminal concluded with a carriage return (`Enter`).

Results

During tests it was discovered that only RFCOMM channel 1 can be used to control the phone via the AT Commands.

As specified in [6], the SMS message in text mode or the PDU in PDU mode is concluded with a special line delimiter `CTRL-Z`. To find out the corresponding ASCII character, the packets generated during an SMS sending process were dumped using `hcidump`. The following is a short extract which shows the end of the sending process.

```
< ACL data: handle 40 flags 0x02 dlen 9
    L2CAP(d): cid 0x0044 len 5 [psm 3]
      RFCOMM(d): UIH: cr 1 dlci 2 pf 0 ilen 1 fcs 0x9a
      0000: 74                                                t
> HCI Event: Number of Completed Packets (0x13) plen 5
  0000: 01 28 00 01 00                                    .(...
> ACL data: handle 40 flags 0x02 dlen 10
    L2CAP(d): cid 0x0040 len 6 [psm 3]
      RFCOMM(d): UIH: cr 0 dlci 2 pf 1 ilen 1 fcs 0x5c credits 1
      0000: 74                                                t
< ACL data: handle 40 flags 0x02 dlen 9
    L2CAP(d): cid 0x0044 len 5 [psm 3]
      RFCOMM(d): UIH: cr 1 dlci 2 pf 0 ilen 1 fcs 0x9a
      0000: 1a                                                .
> HCI Event: Number of Completed Packets (0x13) plen 5
  0000: 01 28 00 01 00                                    .(...
> ACL data: handle 40 flags 0x02 dlen 10
    L2CAP(d): cid 0x0040 len 6 [psm 3]
      RFCOMM(d): UIH: cr 0 dlci 2 pf 1 ilen 1 fcs 0x5c credits 1
      0000: 1a                                                .
> ACL data: handle 40 flags 0x02 dlen 12
    L2CAP(d): cid 0x0040 len 8 [psm 3]
      RFCOMM(s): MSC CMD: cr 0 dlci 0 pf 0 ilen 4 fcs 0xaa mcc_len 2
      dlci 2 fc 0 rtc 1 rtr 1 ic 0 dv 0 b1 0 b2 0 b3 0 len 0
< ACL data: handle 40 flags 0x02 dlen 12
    L2CAP(d): cid 0x0044 len 8 [psm 3]
      RFCOMM(s): MSC RSP: cr 1 dlci 0 pf 0 ilen 4 fcs 0x70 mcc_len 2
      dlci 2 fc 0 rtc 1 rtr 1 ic 0 dv 0 b1 0 b2 0 b3 0 len 0
> HCI Event: Number of Completed Packets (0x13) plen 5
  0000: 01 28 00 01 00                                    .(...
> ACL data: handle 40 flags 0x02 dlen 26
    L2CAP(d): cid 0x0040 len 22 [psm 3]
      RFCOMM(d): UIH: cr 0 dlci 2 pf 0 ilen 18 fcs 0x40
      0000: 0d 0a 2b 43 4d 47 53 3a  20 34 0d 0a 0d 0a 4f 4b  ..+CMGS: 4....OK
      0010: 0d 0a
```

Note that each character sent is echoed. The first character sent is a `t` which is the last character of the message. The second character transmitted is the special line delimiter we were looking for. Now we know its hex value `0x1A`. At the end an `OK` message is is received from the phone, which notifies the success of the message sending process.

As the connection and message sending process has been fully analyzed and understood it can now be implemented in c-code.

2.3.2 RFCOMM Layer

The RFCOMM layer is already implemented in the BTnut system software (BTnut API [5]). However the current version 1.6 produces a bug while trying to create a connection between the BTnode and the cell phone. Therefore a patch has to be applied to the sources:

1. *Open the file* `bt_rfcomm.c` *located in the folder* `btnut/btnode/bt/`
2. *Search for* `case BT_RFCOMM_MSC_CMD` *and insert the instruction* `NutSleep(500);` *on the following line.*
3. *Compile the sources (*`make install`*).*

After that, connection establishment should work just fine. However pairing has to be carried out at every connection attempt. This is due to the BTnode starting the pairing procedure every time. The BTnode uses the default PIN code `1234` therefore this code has to be entered on the phone as well.

2.3.3 AT Commands Layer

On the AT Commands layer a protocol is implemented that allows opening and closing RFCOMM connections to cell phones as well as sending AT Commands. The available functions are declared in `at_phone.h` and defined in `at_phone.c`.

2.3.4 Application Layer

Since the SMS text mode is not supported by every phone, the more complicated PDU mode has to be implemented. An SMS sending interface is provided at this layer. The available functions are declared in `at_sms.h` and defined in `at_sms.c`.

2.4 Results

To test the protocol and interface the demo application `sms.c` was written. It reads the Bluetooth address or the name of the sending phone, the phone number of the receiver and the message to be transmitted. If all goes well, the output on the terminal should look similar to the following.

```
Enter the Bluetooth address or the name of the sending phone: philista

Enter the phone number of the recipient: 41774163126

Enter the message: Greetings from the BTnode!

connecting to phone... RFCOMM connect to 00:0a:28:ee:61:3d Channel 1
RFCOMM Connect on dlci 2...
rfsession: success

----------------
a t + c m g f = 0
 O K
----------------

----------------
a t + c m g s = 3 6
```

```
 >
----------------


----------------
0 0 2 5 0 0 0 b 9 1 1 4 7 7 1 4 3 6 2 1 f 6 0 0 0 0 1 a 4 7 7 9 b 9 4 c 4
f b b c f 7 3 9 0 5 9 f e 6 e 8 3 e 8 e 8 3 2 4 8 4 8 7 5 b f c 9 e 5 1 0
----------------


----------------
 + C M G S :   8 1
 O K
----------------
RFCOMM Disconnect on dlci 2...
```

2.4.1 BTnode Tutorial

From this work, the new chapter *Interfacing to Handheld Devices* evolved for the BTnode tutorial [2]. It was successfully held in the last practical exercise of the lecture *Embedded Systems* as can be seen in figure 2.2.

Figure 2.2: picture taken during the BTnode practical exercise

Chapter 3

VGA Camera Module

3.1 Introduction

With the further development of the CMOS technology more and more complex circuits can be manufactured on the same chip area. Single transistors become smaller and smaller, which results in a higher transistor density. For this reason cheap CMOS imaging sensors have become more attractive. With the smaller technology they do not have the disadvantage in photosensitivity any more compared to the more expensive CCD sensors.

Moreover, because of the lower price CMOS imaging sensors have out-scored CCD sensors in consumer electronics. They are included in many applications even if there is no need just as a toy. For instance it is hardly possible to buy a new cell phone that does not have a CMOS camera included. However the quality of those cell phone cameras varies a lot.

Since these CMOS cameras are popular, it is desired to create an implementation for the BTnode. The VGA camera module used here is provided by COMedia Ltd. This type, C328-7640 is especially intended to be used with PDAs. It basically consists of the OmniVision CMOS image sensor OV7640 the OmniVision JPEG compression chip OV528 and a program memory.

The image sensor provides frames of size 640x480 pixels at a rate of 30 frames per second. In addition it does some preprocessing of the image such as canceling Fixed Pattern Noise (FPN), eliminating smearing, and reducing blooming.

The image is then passed on to the JPEG compression chip which preprocesses the image to the desired pixel format and compresses the data. It has a built in 8051 microcontroller to control the program. Furthermore the OV528 is equipped with an integrated memory buffer for temporary storage of compressed images. The chip provides a four pin RS232 interface for the communication with the host.

The program memory provides the command set for communicating with the host.

This chapter explains how the camera is controlled by the BTnode. It is only a proof of concept, hence just basic functionality is implemented. In section 3.2 communication between the BTnode (host) and the camera is explained. Section 3.3 describes how this communication is realized. Some results are presented in section 3.4. Last but not least section 3.5 discusses the intended use of the camera module with the BTnode.

3.2 Host Communication

3.2.1 Camera Packet Formats

The following explanation about the packet formats is based on [3].

Command Packet

A command packet is 6 Bytes long. It is either sent by the host i.e. the BTnode or the camera module.

0					5
Flag	Command	Parameter1	Parameter2	Parameter3	Parameter4

- Flag is always `0xAA`.
- Command specifies the command sent or received.
- Parameter1 . . . Parameter4 are individually defined for each command

Data Packet

The default size of data packets is 64 Bytes, the maximum is 512 Bytes. Data packets are solely sent by the camera module. However only compressed images are transmitted in packets. Uncompressed images are sent as stream.

0			N-1
Number (2 Bytes)	DataSize (2 Bytes)	ImageData ($N - 6$ Bytes)	Checksum (2 Bytes)

- Number is the packet number starting from 0. Byte 0 of the data packet is the lower Number Byte, Byte 1 the higher one.
- DataSize is the size in Bytes of ImageData. DataSize is $N - 6$ Bytes for all packets except for the last one. Byte 2 of the data packet is the lower DataSize Byte, Byte 3 the higher one.
- ImageData is the actual payload of the packet. Byte 4 of the data packet is the first ImageData Byte, Byte $N - 3$ the last one.
- Checksum is equal to the sum of all Bytes of the data packet except the Checksum. Only the lower Byte of the Checksum (Byte $N - 2$ of the data packet) is valid. The higher Byte (Byte $N - 1$ of the data packet) is always `0x00`.

3.2.2 Camera Command Set

The following explanation about the command set is based on [3].

Command Initial 0x01 Initializes the color depth, the preview resolution and the JPEG resolution. An ACK command is issued by the camera if initialization was successful, otherwise a NACK will be sent back.

Parameter1	**Parameter2**	**Parameter3**	**Parameter4**
	color depth	preview resolution	JPEG resolution
0x00	2-bit Gray Scale 0x01	80x60 0x01	80x64 0x01
	4-bit Gray Scale 0x02	160x120 0x03	160x128 0x03
	8-bit Gray Scale 0x03		320x240 0x05
	12-bit Color 0x05		640x480 0x07
	16-bit Color 0x06		
	JPEG 0x07		

Command Get Picture 0x04 Signals the camera to send a snapshot or video frame. The command is acknowledged with an ACK command if successful, otherwise a NACK is sent. If the command was acknowledged, a Data command is transmitted to inform the host about the image length i.e. the number of Bytes to be transmitted. If a compressed image is issued i.e. data packets are transmitted, then each data packet has to be requested separately with an ACK command. Else the camera starts sending the image stream.

Parameter1	**Parameter2**	**Parameter3**	**Parameter4**
Picture Type			
Snapshot Picture 0x01	0x00	0x00	0x00
Preview Picture 0x02			
JPEG Preview Picture 0x05			

Command Snapshot 0x05 Makes the camera store a JPEG snapshot in the local buffer. An ACK command is issued by the camera if taking the snapshot was successful, otherwise a NACK will be sent back. The number of frames to be dropped before the snapshot is taken can be specified in the two Bytes Parameter2 and Parameter3. Therefore values from 0 to 65535 are allowed.

Parameter1	**Parameter2**	**Parameter3**	**Parameter4**
Snapshot Type	Skip Frame Low Byte	Skip Frame High Byte	
Compressed Picture 0x00	0xXX	0xXX	0x00
Uncompressed Picture 0x01			

Command Set Package Size 0x06 Sets the data packet size for transmission of compressed images. Uncompressed images are not sent in packet format but as data stream. This command has to be issued by the host before a Snapshot or Get Picture command is sent. The command is acknowledged with an ACK command if successful, otherwise a NACK is sent.

Parameter1	**Parameter2**	**Parameter3**	**Parameter4**
0x08	Packet Size Low Byte	Packet Size High Byte	0x00

Command Set Baudrate 0x07 Changes the baud rate. The camera answers with an ACK command if the change was successful, otherwise with a NACK command. After receiving ACK from the camera, the host has to continue transmission with the new baud rate.

Baudrate	**Parameter1**	**Parameter2**	**Parameter3**	**Parameter4**
7200 bps	`0xFF`	`0x01`	`0x00`	`0x00`
9600 bps	`0xBF`	`0x01`	`0x00`	`0x00`
14400 bps	`0x7F`	`0x01`	`0x00`	`0x00`
19200 bps	`0x5F`	`0x01`	`0x00`	`0x00`
28800 bps	`0x3F`	`0x01`	`0x00`	`0x00`
38400 bps	`0x2F`	`0x01`	`0x00`	`0x00`
57600 bps	`0x1F`	`0x01`	`0x00`	`0x00`
115200 bps	`0x0F`	`0x01`	`0x00`	`0x00`

Command Reset 0x08 Resets the camera module. An ACK command signals the success, a NACK command the failure of the reset.

Parameter1		**Parameter2**	**Parameter3**	**Parameter4**
Reset Type				
complete reset	`0x00`	`0x00`	`0x00`	`0x00`
reset FSMs only	`0x01`			

Command Power Off 0x09 Sets the camera into sleep mode. An ACK command signals the success, a NACK command the failure of the command. To wake up the camera again, SYNC command has to be sent for a certain period until receiving an ACK from the camera.

Parameter1	**Parameter2**	**Parameter3**	**Parameter4**
`0x00`	`0x00`	`0x00`	`0x00`

Command Data 0x0A The camera issues this command after the host has initiated a Get Picture to tell the host the type and length (number of Bytes) of the image data that is ready for transmission to the host. If the image data is compressed, the host has to send an ACK command for each data packet to be received. However, if the image data is uncompressed, the image data stream transmission starts right after the Data command.

Parameter1		**Parameter2**	**Parameter3**	**Parameter4**
Data Type		Length Byte 0	Length Byte 1	Length Byte 2
Snapshot Picture	`0x01`	`0xXX`	`0xXX`	`0xXX`
Preview Picture	`0x02`			
JPEG Preview Picture	`0x05`			

Command SYNC 0x0D Creates a connection to the camera. The command can be issued at an arbitrary baud rate. However, it has to be sent for a certain period, at most 60 times until receiving an ACK from the camera. This usually happens after 25 SYNC commands. The camera then in turn sends a SYNC command which has to be acknowledged by the host.

Parameter1	Parameter2	Parameter3	Parameter4
0x00	0x00	0x00	0x00

Command ACK 0x0E This command is used in two different situations. On the one hand, it indicates the success of the operation specified by Command in Parameter1. For each successful operation the ACK counter is incremented by one. On the other hand, the host has to issue this command to request the image data packet with the desired packet number after receiving Data command from the camera. To end the packet transfer, the host should issue the ACK command with packet number `0xF0F0`.

Parameter1	Parameter2	Parameter3	Parameter4
Command	ACK counter	0x00	0x00

Parameter1	Parameter2	Parameter3	Parameter4
0x00	0x00	Packet Number Byte 0	Packet Number Byte 1

Command NACK 0x0F Indicates corrupt transmission or unsupported features.

Parameter1	Parameter2	Parameter3	Parameter4
0x00	NACK counter	Error Number	0x00

3.2.3 Receiving an Image

Table 3.1 shows the commands and data exchanged between host and camera when an image is received. It is based on [3]. A JPEG compressed snapshot of resolution 640x480 pixels is requested.

3.3 Implementation

3.3.1 Hardware

As already mentioned, the camera module provides a four pin RS232 interface. Therefore, the BTnode can simply be connected to the camera via the UART interface. Since the camera produces quite a lot of data it is best to connect it to the application UART because this interface supports higher data rates than the software UART. The debug connector J2 on the BTnode is used in order to keep the extension connector J1 free for other applications. The following pins are needed:

GND	Pin 1 Ground
UART0_TXD	Pin 4 Transmit Exchange Data
UART0_RXD	Pin 5 Receive Exchange Data
VCC	Pin 14 Supply Voltage

Figure 3.1 shows a picture of the BTnode - camera setup.

Host	**Transfer**	**Camera**
SYNC	`AA 0D 00 00 00 00`	max. 60 times
SYNC	`AA 0D 00 00 00 00`	
SYNC	`AA 0D 00 00 00 00`	
	⋮	
SYNC	`AA 0D 00 00 00 00`	
	`AA 0E 0D 00 00 00`	ACK
	`AA 0D 00 00 00 00`	SYNC
ACK	`AA 0E 0D 01 00 00`	
Initial	`AA 01 00 07 00 07`	
JPEG Preview, 640x480		
	`AA 0E 01 01 00 00`	ACK
Snapshot	`AA 05 00 00 00 00`	
compressed image		
	`AA 0E 05 02 00 00`	ACK
Get Picture	`AA 04 01 00 00 00`	
snapshot image		
	`AA 0E 04 03 00 00`	ACK
	`AA 0A 01 XX XX XX`	Data
		snapshot image
ACK	`AA 0E 00 00 00 00`	
packet number 0		
	64 Bytes	Data Packet
		packet number 0
ACK	`AA 0E 00 00 01 00`	
packet number 1		
	64 Bytes	Data Packet
	⋮	packet number 1
	⋮	
	7-64 Bytes	Data Packet
		last data packet
ACK	`AA 0E 00 00 F0 F0`	
end packet transfer		
Power Off	`AA 09 00 00 00 00`	
	`AA 0E 09 04 00 00`	ACK

Table 3.1: receiving an image

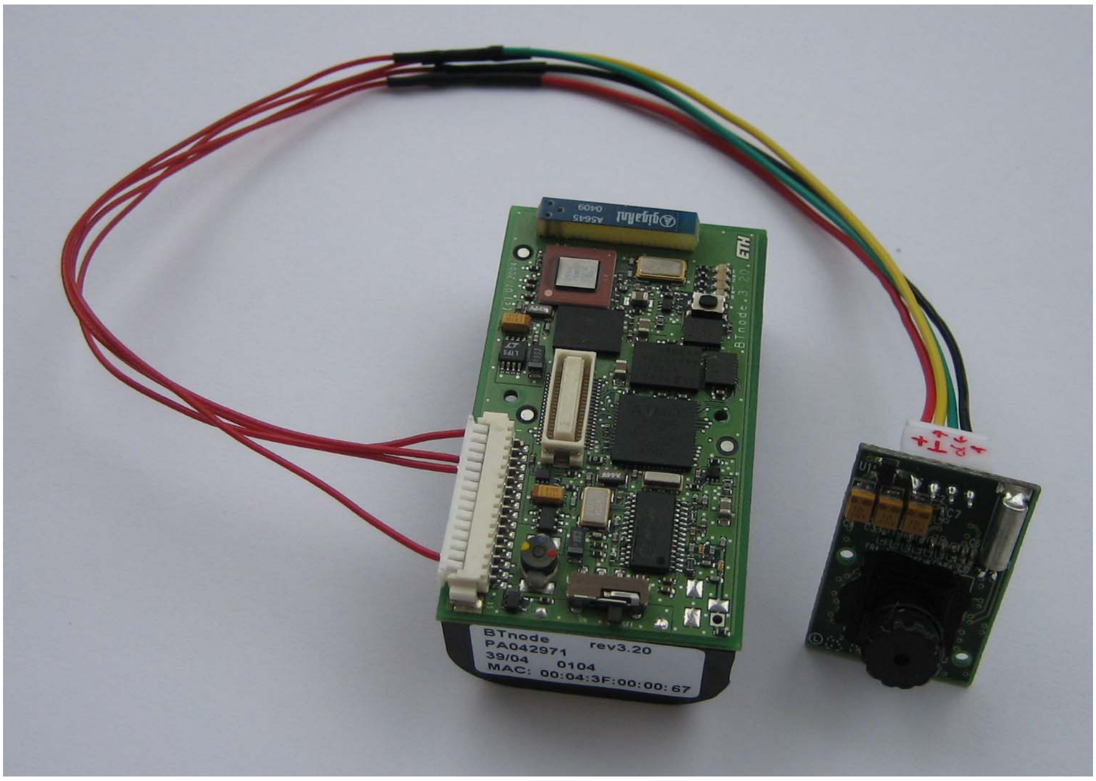


Figure 3.1: connecting the camera to the BTnode

3.3.2 Software

The basic functionality to communicate with the camera as described in section 3.2 is implemented as a device driver in software. The necessary functions are declared in the file `btnode_cam.h` and defined in the file `btnode_cam.c`.

Also an API is written that provides the terminal commands needed to receive an image from the camera module and to print it out on the terminal. The command registering functions are declared in the file `btnode_cam_cmds.h` and defined in the file `btnode_cam_cmds.c`.

A sample application is implemented in the file `cam.c`. The application initializes the terminal and registers the necessary terminal commands for receiving an image with the BTnode and printing it out to the terminal. The special terminal output format of the image is described in subsection 3.3.4.

3.3.3 Memory Management

To test the camera, the necessary functions to receive a snapshot image are executed on the BTnode. This snapshot image is buffered in the memory on the BTnode. For future applications this step may not be necessary, as the received packets from the camera can be directly forwarded to the intended destination.

Tests have shown that the JPEG compressed camera images normally take about 20-30 KBytes of memory. This figure varies because of the JPEG compression efficiency. To be on the safe side, 40 KBytes of memory are reserved for the image.

To comply with these memory requirements, the additional RAM banks of the BTnode have to be used. The first 129 KBytes of the additional RAM banks are reserved for the program. Additional 10 KBytes are taken by the event logger. Consequently, the following address space is reserved for the camera image:

Starting Address	139 KB
Size	40 KB

This address space can be accessed using the functions declared in the BTnut system file `btnut/btnode/include/hardware/ram_banks.h`.

3.3.4 Intel Hex Format

In a first step, the BTnode controlling the camera is directly connected to a computer. For debugging purposes the received image is transmitted to the terminal on the computer. Because of handling reasons the image data is printed out in the *Intel Hex Format*. This allows easy logging in the terminal. The monitored data can then be inserted in a `.hex` file. With simple command line tools such as the KEIL HEX2BIN converter, the `.hex` file can be transformed to a binary file. This file finally contains the original JPEG data. It can be displayed in a simple JPEG viewer.

Configuration

This explanation of the *Intel Hex Format* is based on [8] and [1].

Records All data lines are called records and each record contains the following fields:

`:llaaaatt[dd...]cc`

:	Every line starts with a colon. This is actually the only non-hexadecimal character in a record.
`ll`	record-length field (1 Byte). Represents the number of data Bytes `dd` in the record.
`aaaa`	address field (2 Bytes). Represents the first address to be used by this record.
`tt`	record type (1 Byte). `00` data record `01` end-of-file record
`[dd...]`	data field that represents one Byte of data. A record may have multiple data Bytes. The number of data Bytes in the record must match the number specified by the `ll` field.
`cc`	Checksum (1 Byte). The checksum is calculated by summing up the values of all hexadecimal digit pairs in the record modulo 256 and taking the two's complement.

Data Records An Intel HEX file consists of an arbitrary number of *Data Records*. In the implementation used in this work, continuous data is formated in *Intel Hex Format*. Hence, the address field always has to point to the next free memory space. For example if each record contains 16 Bytes of data, then the first address field has to point to address 0, the second to address 16, the third to address 32 and so on.

Figure 3.2: image taken with the camera

End-of-File (EOF) Records An Intel HEX file must end with an end-of-file (EOF) record. This record must have the value 01 in the record type field. An EOF record always appears as follows:

```
: 00 0000 01 FF
```

Example

```
: 10 0000 00 FFD8FFE000114A464946000102030405 FB
: 10 0010 00 060708090AFFDB004300100C0C0E0C0A 4F
: 10 0020 00 100E0E0E121210141828 1A1816161832 66
: 10 0030 00 24261E283A343E3C3A34383840485C4E 38
: 00 0000 01                                  FF
```

3.4 Results

The following output is produced at the terminal when an image is requested as depicted in section 3.2.3. With the `cam` terminal command, a snapshot is stored in the RAM. The `pic` terminal command then produces the image in *Intel Hex Format* to the terminal. The result can be seen in figure 3.2 which shows an image taken with the camera.

```
# ------------------------------------------------------
# Welcome to BTnut (c) 2006 ETH Zurich
# btnode_cam program version: 20060627-2012
debug READTIMEOUT: 100
debug READBUFFERSIZE: 256
[philista]$cam
debug: connecting to camera
debug: ACK received: aa:e:d:0:0:0
debug: SYNC received: aa:d:0:0:0:0
debug: camera set up: successful
-------------------------------------------
debug: changing the baud rate
```

```
debug: ACK received: aa:e:7:1:0:0
debug: baud rate change: successful
-------------------------------------------
debug: initial
debug: Command sent: aa:1:0:7:3:7
debug: ACK received: aa:e:1:2:0:0
debug: initial: successful
-------------------------------------------
debug: snapshot
debug: Command sent: aa:5:0:0:0:0
debug: ACK received: aa:e:5:3:0:0
debug: snapshot: successful
-------------------------------------------
debug: get_picture
debug: Command sent: aa:4:1:0:0:0
debug: ACK received: aa:e:4:4:0:0
debug: data: aa:a:1:c:69:0
debug: img_size: 26892
debug: request packet: aa:e:0:0:0:0
debug: receive packet (header[data]checksum): 0:0:58:0:...:16:0
debug: request packet: aa:e:0:0:1:0
debug: receive packet (header[data]checksum): 1:0:58:0:...:41:0
debug: request packet: aa:e:0:0:2:0
debug: receive packet (header[data]checksum): 2:0:58:0:...:246:0
debug: request packet: aa:e:0:0:3:0
debug: receive packet (header[data]checksum): 3:0:58:0:...:231:0
```

⋮

```
debug: request packet: aa:e:0:0:cd:1
debug: receive packet (header[data]checksum): cd:1:58:0:...:254:0
debug: request packet: aa:e:0:0:ce:1
debug: receive packet (header[data]checksum): ce:1:58:0:...:247:0
debug: request packet: aa:e:0:0:cf:1
debug: receive packet (header[data]checksum): cf:1:38:0:...:14:0
debug: terminate transfer aa:e:0:0:f0:f0
get_picture: successful
-------------------------------------------
debug: power off
debug: Command sent: aa:9:0:0:0:0
debug: ACK received: aa:e:9:5:0:0
power_off: successful
[philista]$pic
```

```
-----------------------------------
```

```
:10000000FFD8FFE000114A464946000102030405FB
:1000100000060708090AFFDB004300100C0C0E0C0A4F
:10002000100E0E0E121210141828 1A181616183266
:1000300024261E283A343E3C3A34383840485C4E38
```

```
⋮

:100890005794632531FEEE3FAD64EE99A277443F55
:1008A00067279DB4D7B6F97938AB5E69EF8A63C81C
:0808B00008E9FAD3D40FFFD9C7
:00000001FF

----------------------------------
[philista]$
```

3.5 Application

The sample application used in this work is sufficient for concept demonstration. However as a real application it does not make much sense to connect the camera to a BTnode which in turn is connected to a computer. The camera would rather be connected directly to the computer.

The idea though is to integrate the camera connected BTnode in a network of BTnodes. In this setup other nodes should be able to communicate with the camera node over Bluetooth. They should be able to access the camera node directly or via other nodes (multihop). In order to realize this integration, two protocols are needed:

- A routing protocol has to run on the nodes of the network to setup the forwarding paths for image request packets and image data packets. Furthermore each node has to keep a routing table in the local memory where these forwarding rules are stored.
- A higher layer protocol has to define the packets that are sent over Bluetooth from remote nodes to control the camera and from the camera node to send data.

Possible applications for the camera in the ETZ building are monitoring the cafeteria entry to see if there is a line or the foosball table to see if it is occupied. Figure 3.3 shows some devices such as PDAs, computers and other BTnodes that could connect to the camera controlling BTnode.

Implementation and testing of the described protocols could be an excellent assignment for a new Semester or Master Thesis. Moreover, a java applet would be nice for viewing the image.

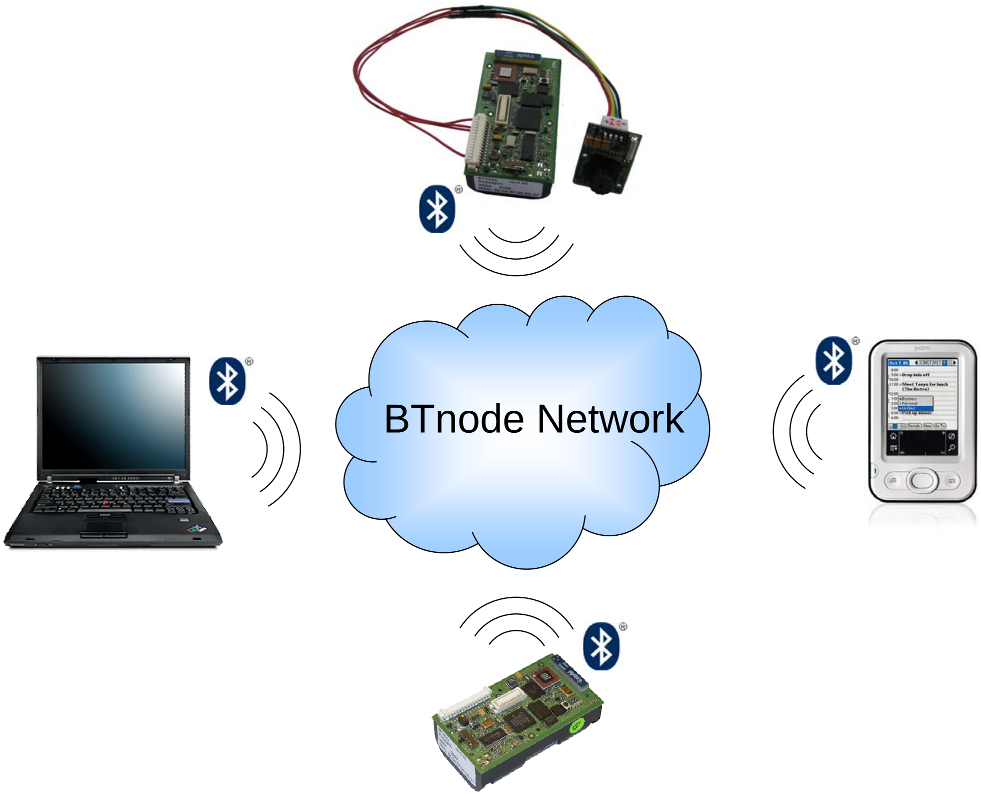


Figure 3.3: devices to interact with the BTnode controlling the camera

Chapter 4

Conclusion

In this assignment two demonstration applications were implemented for the BTnode. In a first part the BTnode connected to a cell phone to make the phone send an SMS message. In a second part the BTnode was able to control a CMOS VGA camera module.

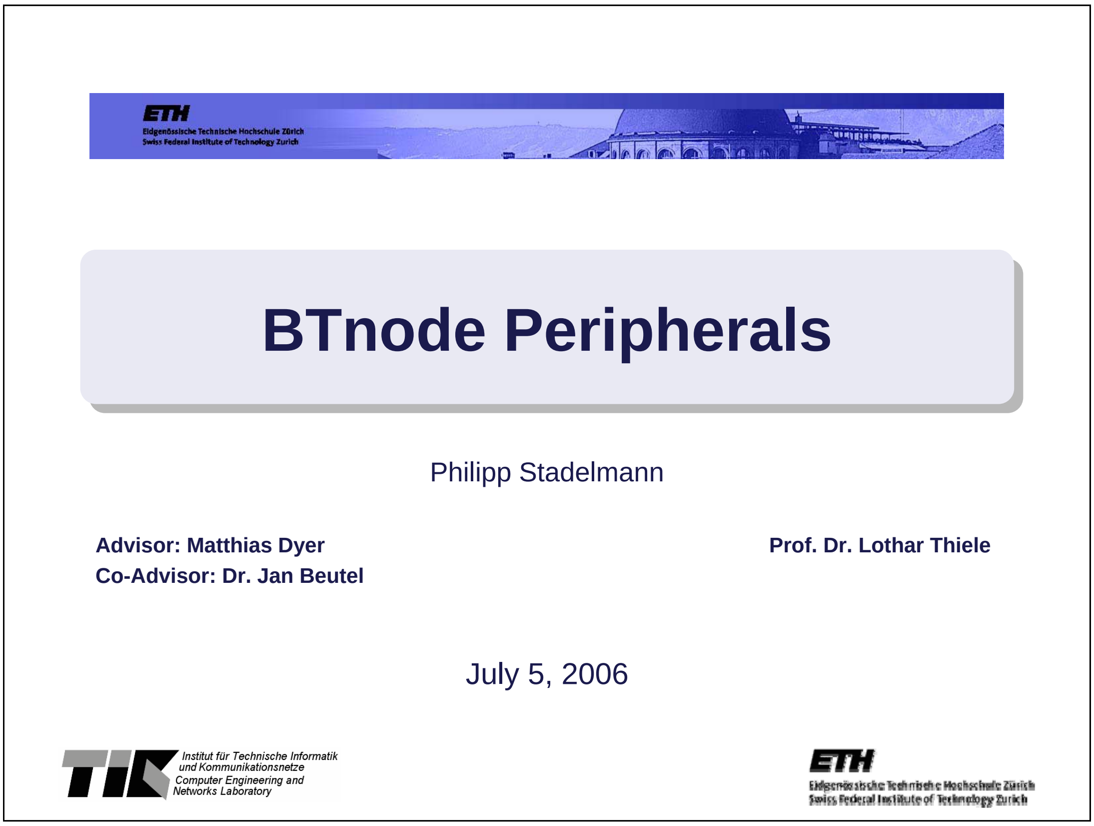
As can be seen from the two applications, the BTnode is very versatile. It has been shown that the BTnode can easily connect to external devices that are equipped with the necessary interface. It is imaginable to use the BTnode with even other devices.

The BTnode could for example also be used in conjunction with the LEGO Mindstorms since they have a Bluetooth module integrated in their controller. Among many others a remote control would be an obvious application.

To sum up also external devices can interact with a network of sensor nodes. This makes the opportunities of sensor networks even more numerous.

Appendix A

Slides of Presentation



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

BTnode Peripherals

Philipp Stadelmann

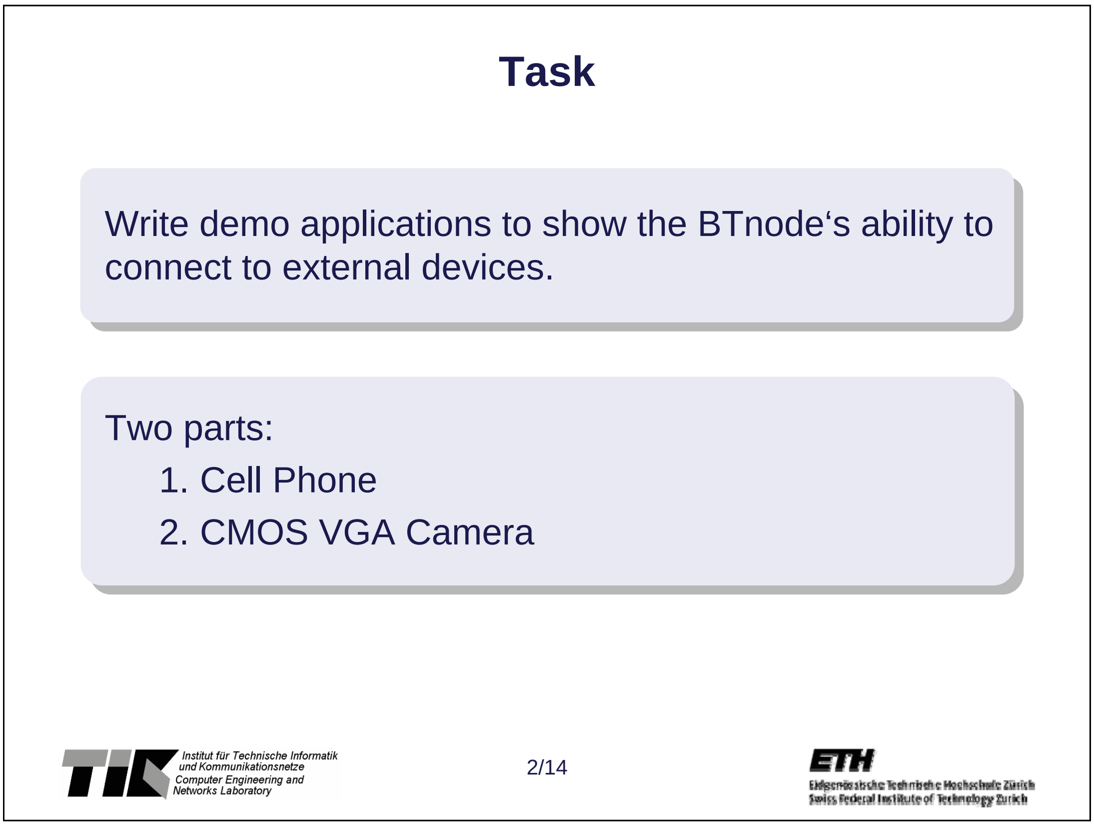
Advisor: Matthias Dyer
Co-Advisor: Dr. Jan Beutel

Prof. Dr. Lothar Thiele

July 5, 2006

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

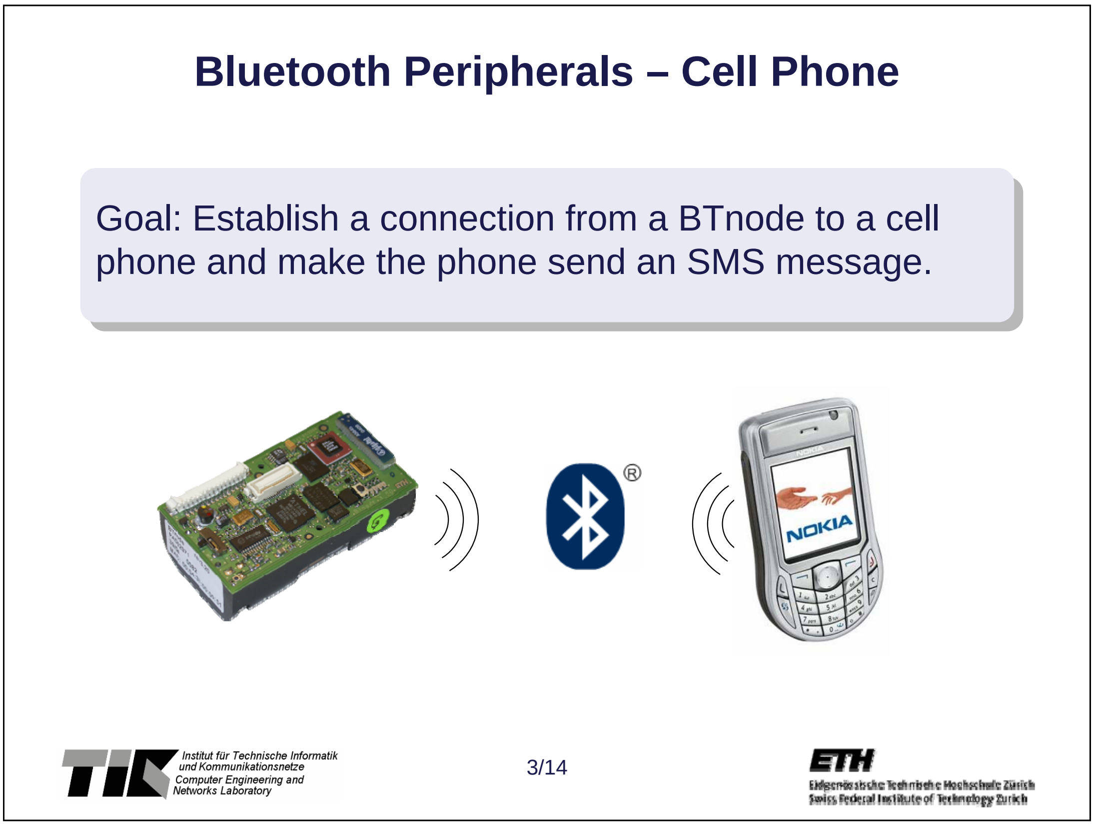
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich



Task

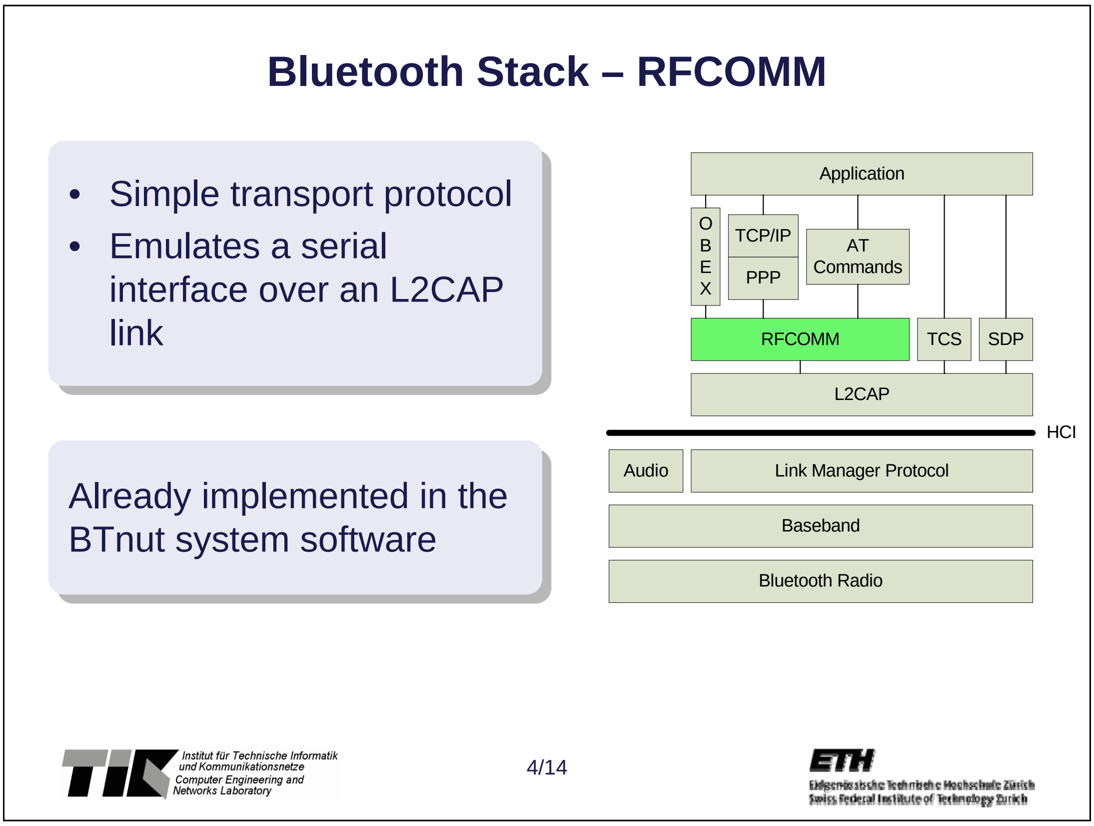
Write demo applications to show the BTnode‘s ability to connect to external devices.

Two parts:
1. Cell Phone
2. CMOS VGA Camera



Bluetooth Peripherals – Cell Phone

Goal: Establish a connection from a BTnode to a cell phone and make the phone send an SMS message.



Bluetooth Stack – RFCOMM

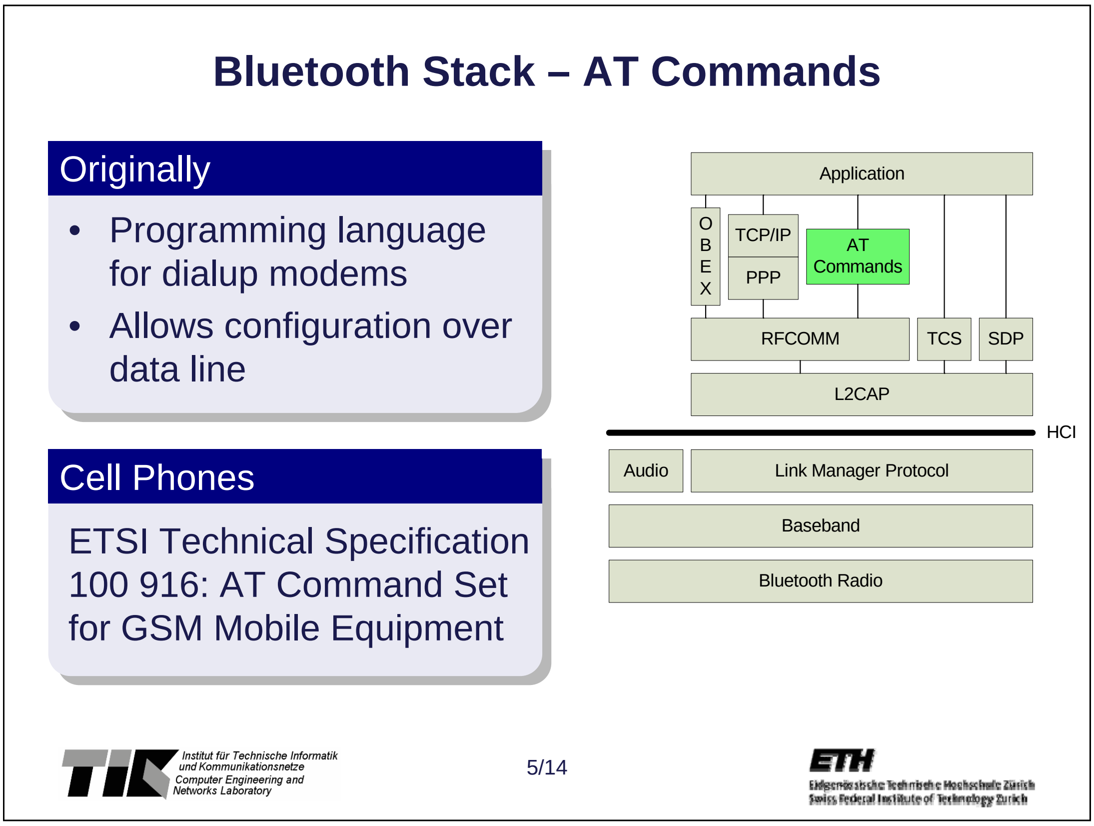
- Simple transport protocol
- Emulates a serial interface over an L2CAP link

Already implemented in the BTnut system software

Application
OBEX
TCP/IP
PPP
AT Commands
RFCOMM
TCS
SDP
L2CAP
HCI
Audio
Link Manager Protocol
Baseband
Bluetooth Radio

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich



Bluetooth Stack – AT Commands

Originally

- Programming language for dialup modems
- Allows configuration over data line

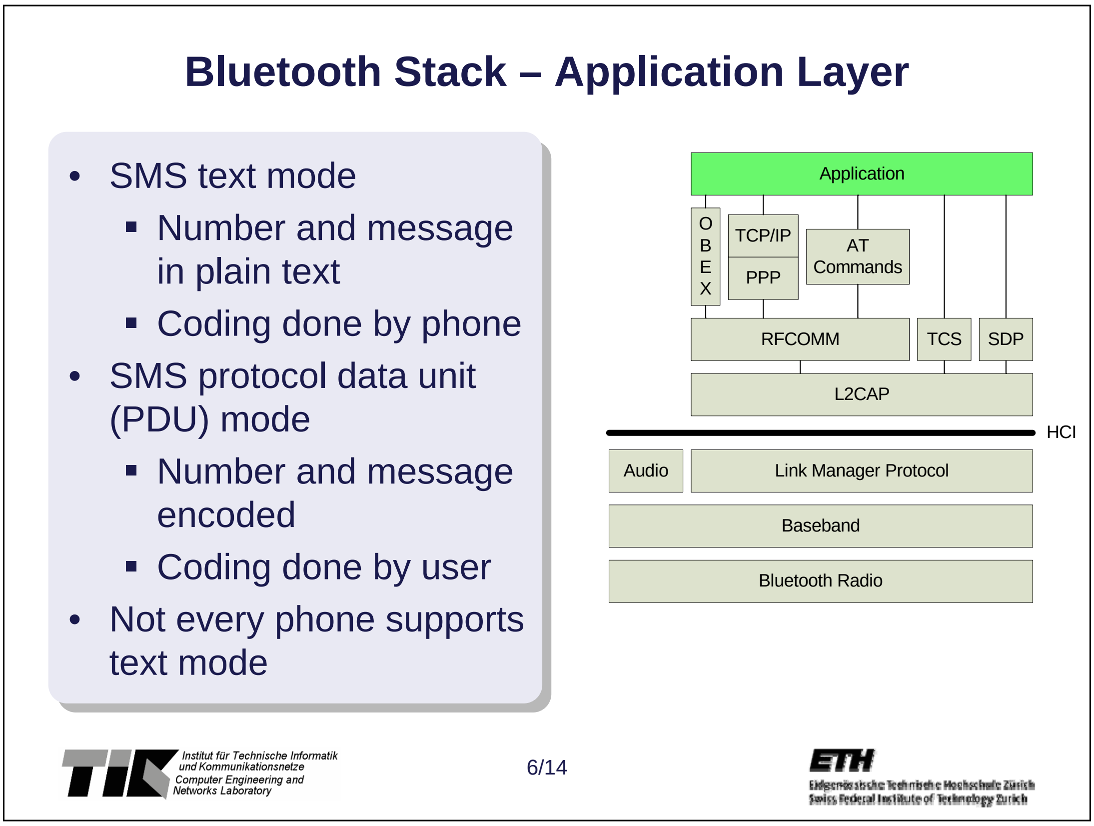
Cell Phones

ETSI Technical Specification 100 916: AT Command Set for GSM Mobile Equipment

Application
OBEX
TCP/IP
PPP
AT Commands
RFCOMM
TCS
SDP
L2CAP
HCI
Audio
Link Manager Protocol
Baseband
Bluetooth Radio

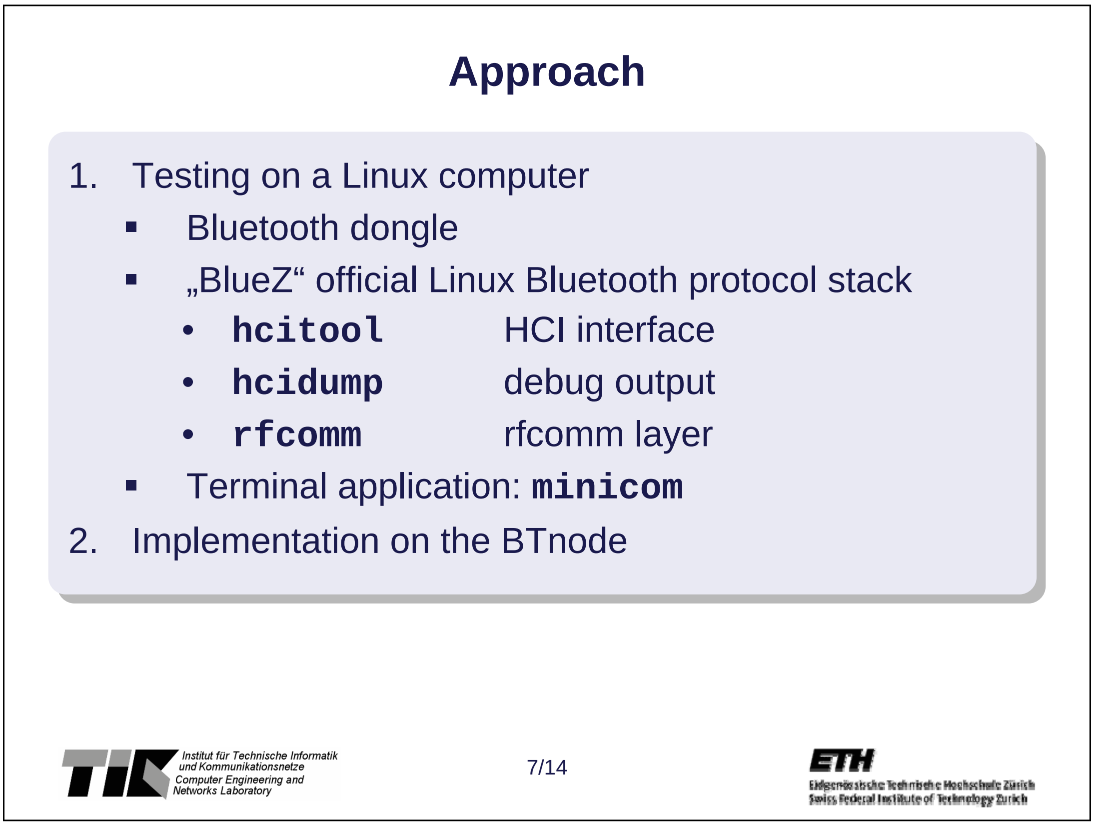
Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich



Bluetooth Stack – Application Layer

- SMS text mode
    - Number and message in plain text
    - Coding done by phone
- SMS protocol data unit (PDU) mode
    - Number and message encoded
    - Coding done by user
- Not every phone supports text mode



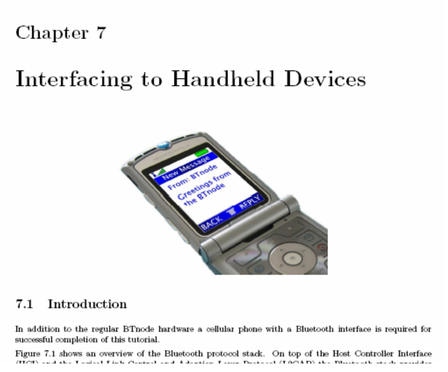
Approach

1. Testing on a Linux computer
    - Bluetooth dongle
    - „BlueZ“ official Linux Bluetooth protocol stack
        - **`hcitool`** HCI interface
        - **`hcidump`** debug output
        - **`rfcomm`** rfcomm layer
    - Terminal application: **`minicom`**
2. Implementation on the BTnode

Implementation

1. AT Commands Layer
    - Protocol
        - at_phone.h, at_phone.c
2. Application Layer
    - SMS sending interface
        - at_sms.h, at_sms.c
    - Demo Application
        - sms.c

Chapter for Btnode Tutorial



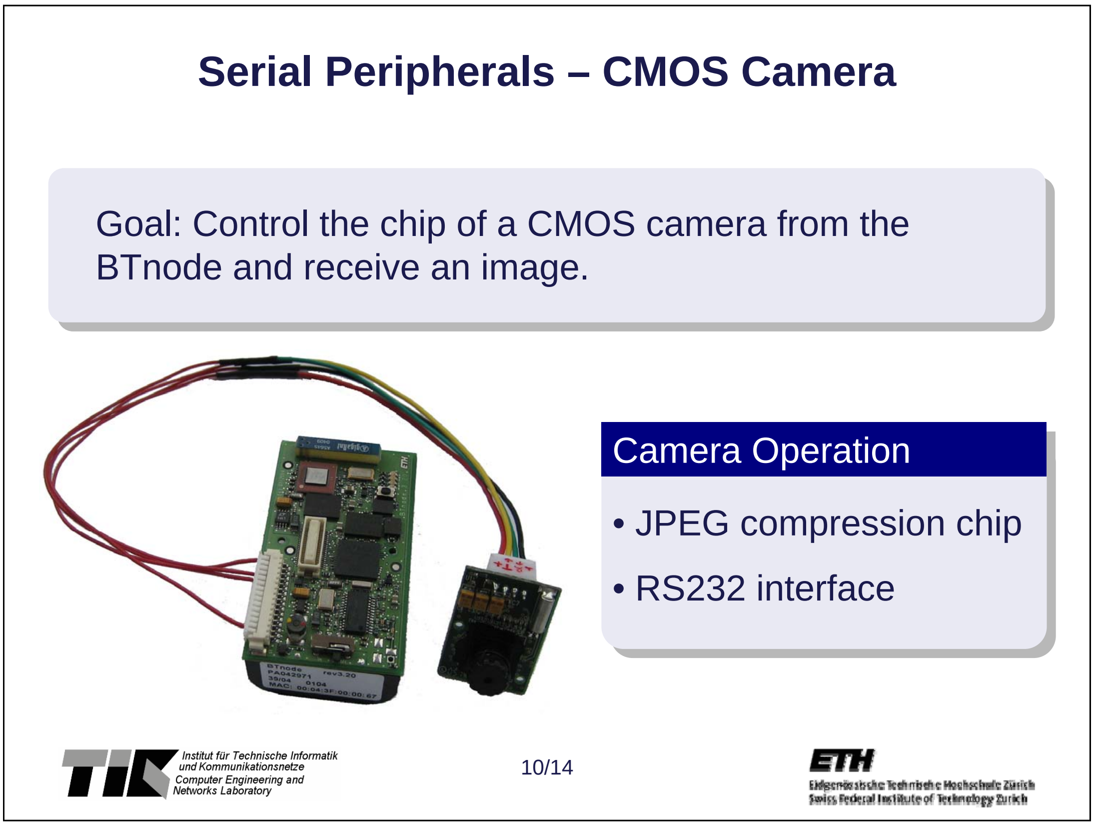
Chapter 7

Interfacing to Handheld Devices

7.1 Introduction

In addition to the regular BTnode hardware a cellular phone with a Bluetooth interface is required for successful completion of this tutorial.

Figure 7.1 shows an overview of the Bluetooth protocol stack. On top of the Host Controller Interface

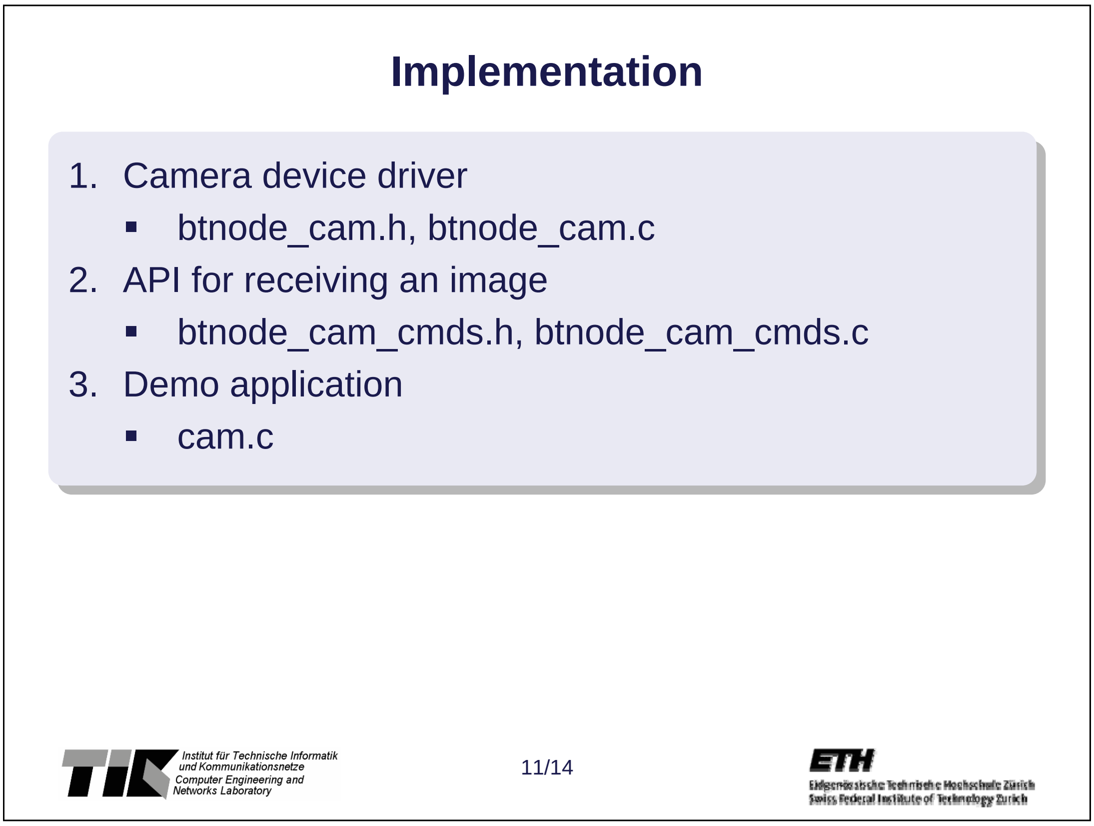


Serial Peripherals – CMOS Camera

Goal: Control the chip of a CMOS camera from the BTnode and receive an image.

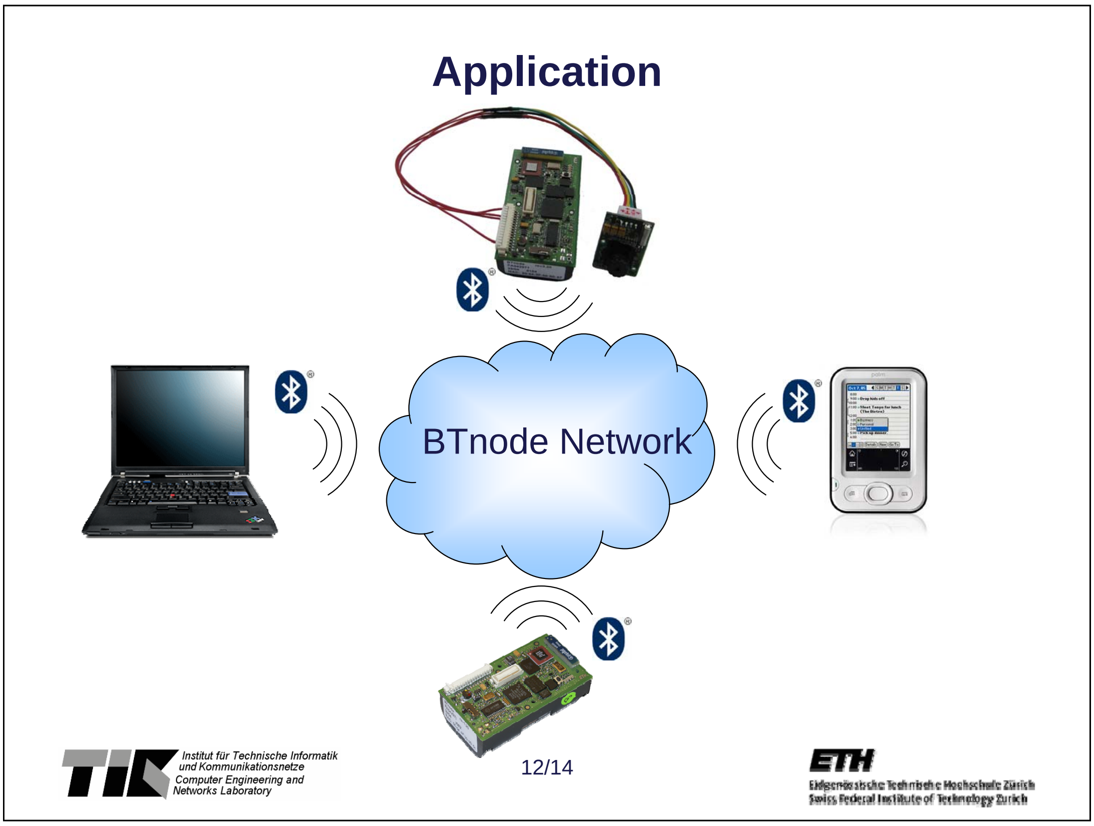
Camera Operation

- JPEG compression chip
- RS232 interface



Implementation

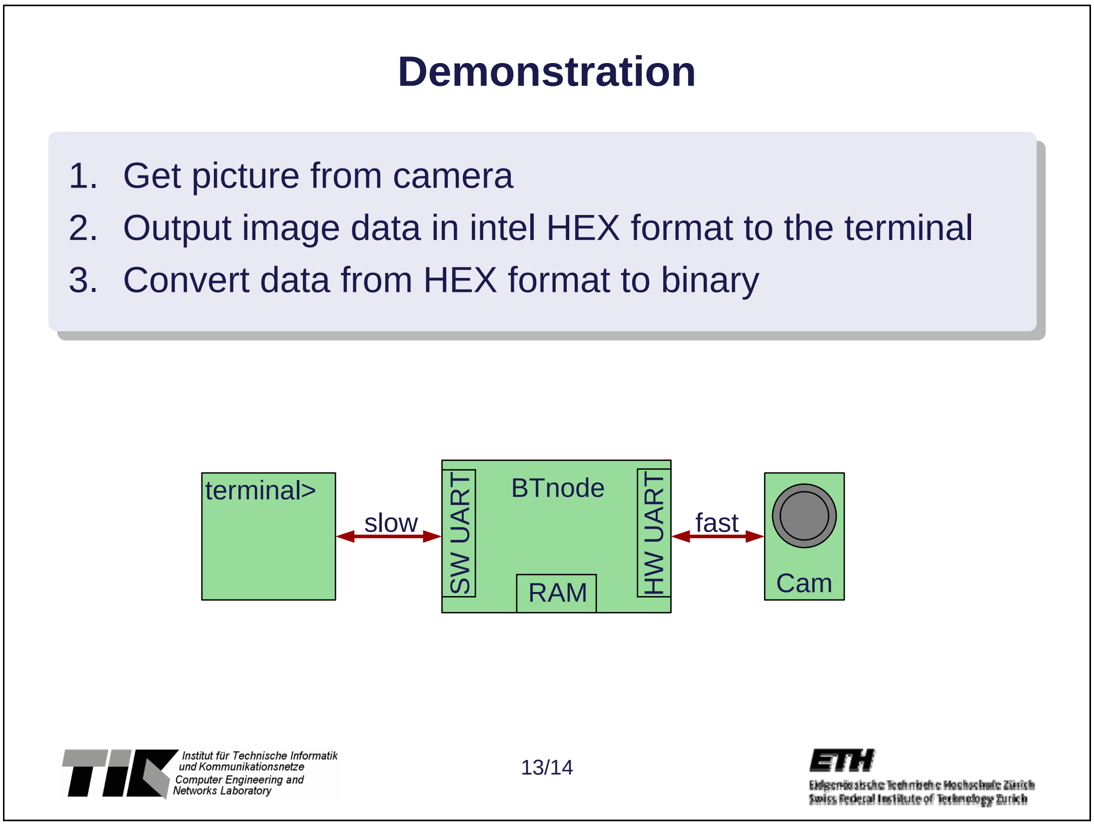
1. Camera device driver
   - btnode_cam.h, btnode_cam.c
2. API for receiving an image
   - btnode_cam_cmds.h, btnode_cam_cmds.c
3. Demo application
   - cam.c



Application

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

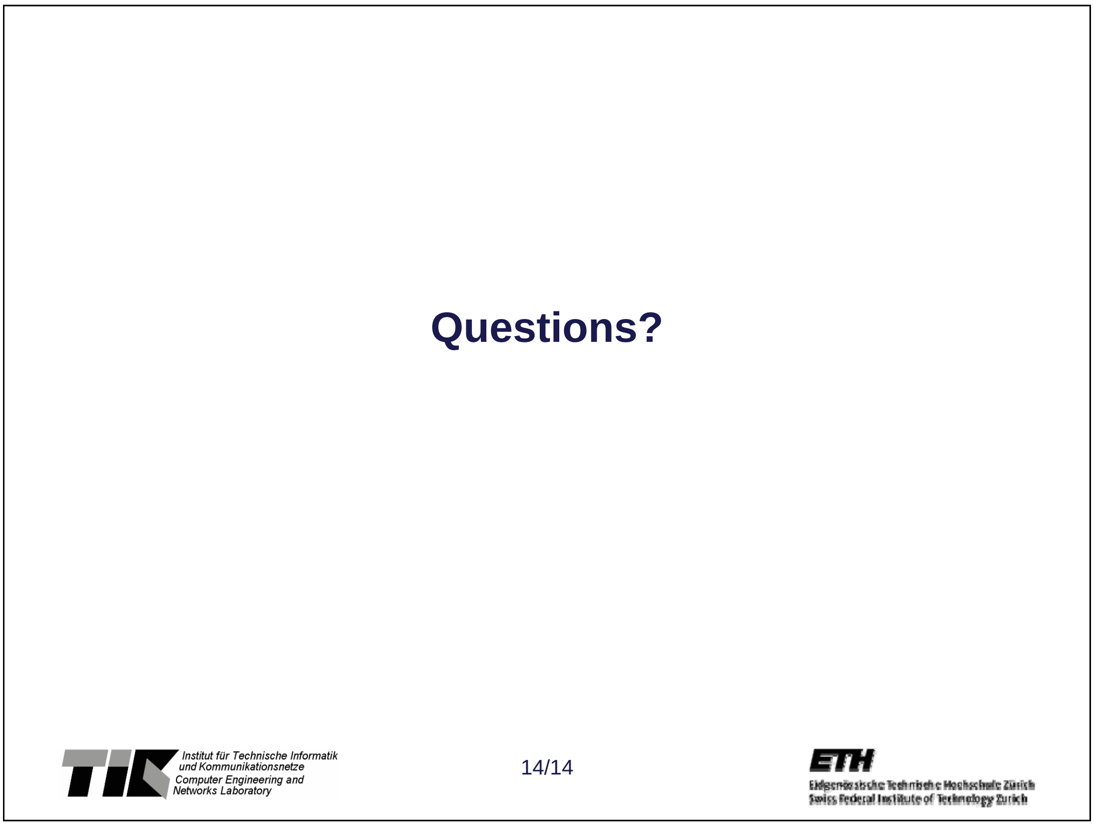


Demonstration

1. Get picture from camera
2. Output image data in intel HEX format to the terminal
3. Convert data from HEX format to binary

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich



Questions?

Institut für Technische Informatik
und Kommunikationsnetze
Computer Engineering and
Networks Laboratory

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Bibliography

[1] San Bergmans. *Intel HEX Format.* Knowledge Base, www.sbprojects.com, August 2005.

[2] J. Beutel, Ph. Blum, M. Dyer, C. Moser, and Ph. Stadelmann. *BTnode Programming - An Introduction to BTnut Applications.* Computer Engineering and Networks Laboratory, ETH Zurich, 8092 Zurich, Switzerland, 1.3 edition, June 2006.

[3] COMedia Ltd. *C328-7640 User Manual for CMOS VGA Camera Module, Version 3*, August 2005.

[4] Computer Engineering and Networks Laboratory, ETH Zurich, http://sourceforge.net/projects/btnode. *BTnode System Software.*

[5] Computer Engineering and Networks Laboratory, ETH Zurich, www.btnode.ethz.ch. *BTnodes - A Distributed Environment for Prototyping Ad Hoc Networks.*

[6] ETSI. *Technical Specification 100 585 - Equipment (DTE - DCE) interface for Short Message Service (SMS) and Cell Broadcast Service (CBS), Version 7.0.1*, July 1999.

[7] ETSI. *Technical Specification 100 916 - AT command set for GSM Mobile Equipment, Version 7.7.0*, December 2001.

[8] Keil - An ARM Company, www.keil.com. *Intel Hex File Format*, June 2004. Technical Support.

[9] Motorola, http://developer.motorola.com. *The Hayes AT Command Set With Motorola Handsets.*

[10] Nokia, www.forum.nokia.com. *AT Command Set For Nokia GSM And WCDMA Products.*

[11] Nokia. *Support Guide for the Nokia Phones and AT Commands*, May 2002.

[12] Sony Ericsson, http://developer.sonyericsson.com. *Developers Guidelines - AT Commands.*

[13] Sony Ericsson. *Developers Guidelines - AT Commands*, August 2005.

[14] SourceForge.net, http://fma.sourceforge.net/. *floAt's Mobile Agent.*