Technische Universität Braunschweig

Institut für Betriebssysteme und Rechnerverbund

WSN Praktikum SS 2012

Tutorial zur Einführung

Robert Hartung

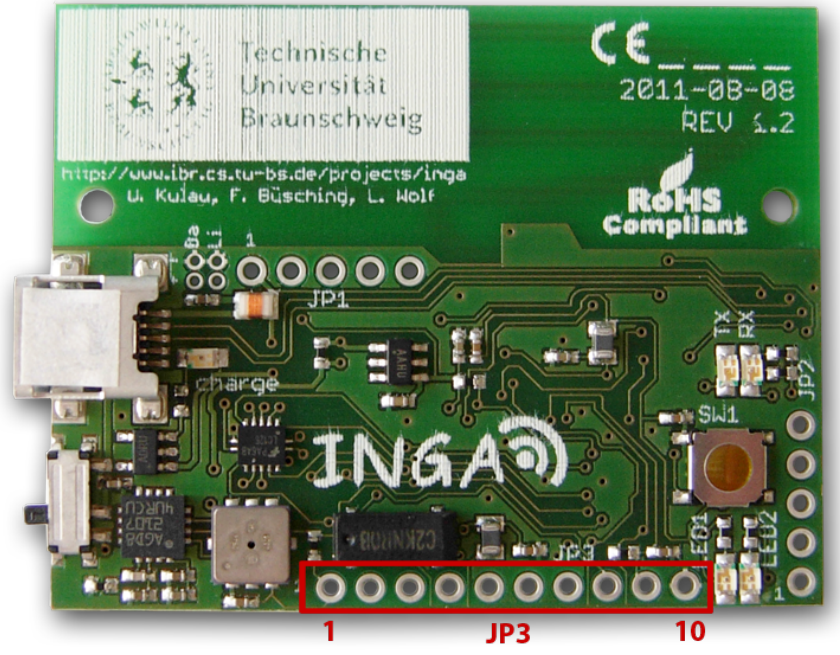
19.04.2012, Version 1.2

Outline

Worum geht es?

Was ist INGA?

- ein Sensor Knoten
- basiert auf dem AVR-Raven board
- cool
- eigentlich für alte Leute



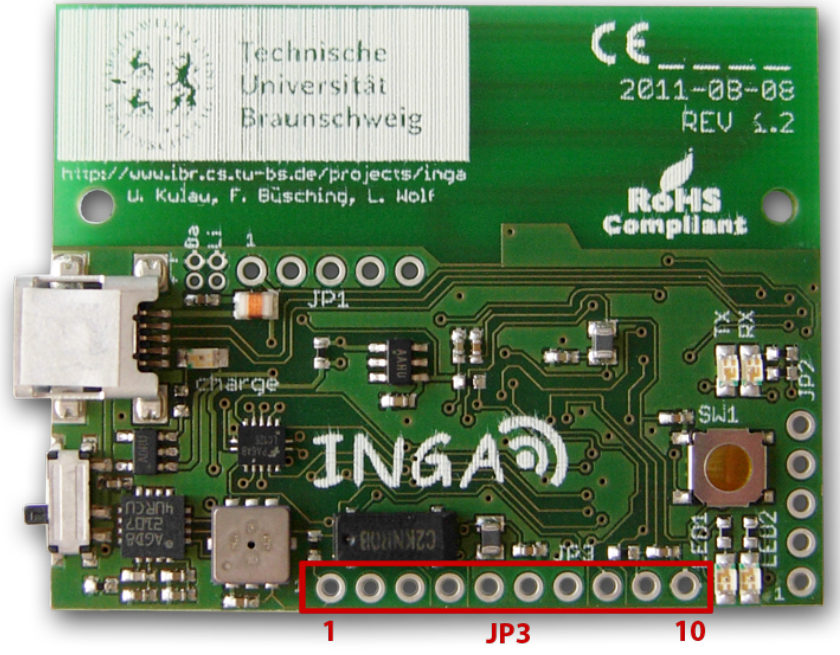
Hardware

- Atmega1284u
- Radiochipset
- diverse Sensoren

Inga

Features

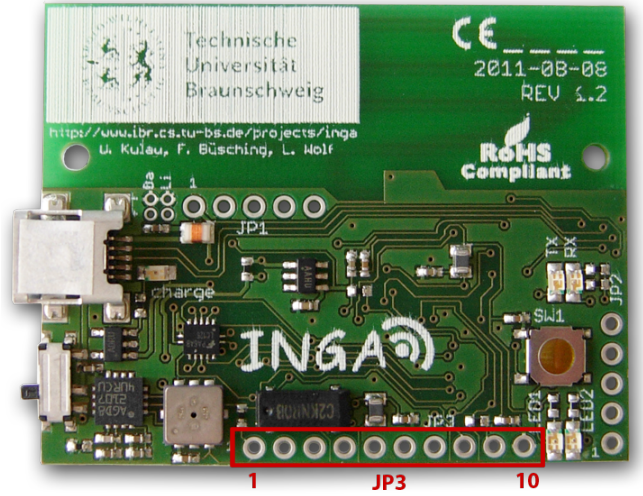
- kompatibel mit Contiki
- über USB programmierbar
- SD-Karten Slot
- rausgeführte Schnittstellen
  - I^2C
  - SPI
  - AD Eingang
  - Uart: RX, TX
- viele Sensoren
- Aktoren (Leds)

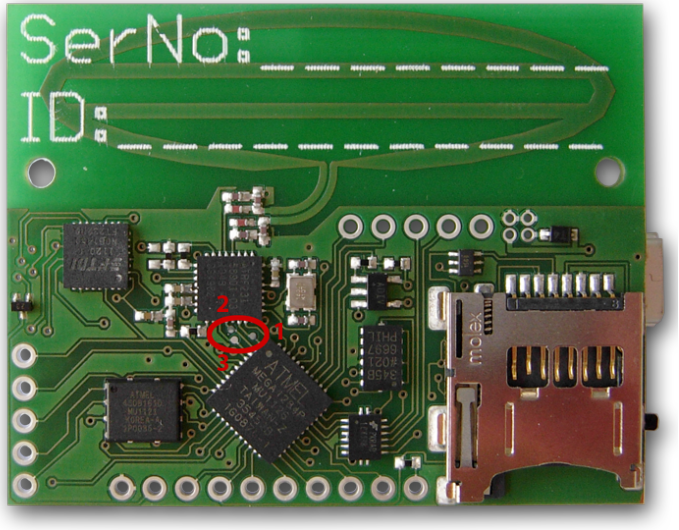


Inga - Sensoren

Sensoren:

- Button
- Temperatur-Sensor (2x)
- Accelerometer
- Gyroskop
- Luftdruck-Sensor





Entwicklungsumgebung

Windows:

- VirtualBox mit Instant Contiki

Linux:

- VirtualBox mit Contiki
- Ubuntu und Pakete selber installieren (empfohlen)
  - gcc-avr
  - avrdude
  - avr-libc
  - git

Entwicklungsumgebung - G40

Übersicht:

Pakete sind installiert

INGA branch herunterladen

Einführung in git

Git

Warum git?

- Contiki wird mit Hilfe von git entwickelt
- IBR erweitert das Repository für INGA weiter

Checkout

Anonymes Checkout

```
git clone git://git.ibr.cs.tu-bs.de/contiki-inga.git
```

Commiten

lokal:

```
git add bla.c
git ci
```

oder

```
git ci -a
```

zum remote Reprository:

```
git push origin master
```

Branches

Zum Ausprobieren von (unabhängigen) Features sind Branches sehr hilfreich.

Liste aller Branches (* = aktuelles)

```
git branch -a
```

Branch wechseln

```
git co <branchname>
```

Branches zusammenführen

```
git merge <branchname>
```

http://book.git-scm.com/3_basic_branching_and_merging.html

Nützliches

weitere Befehle:

```
git diff
git diff -cached
git log
git status
```

Links:

http://byte.kde.org/~zrusin/git/git-cheat-sheet-medium.png

C Kurzeinführung

Programmiersprache C

- Header-Datei (.h)
- C-Datei (.c)
- Externe Dateien einbinden

```
#include "file.h"
```

Beispiel H-Datei (my.h)

```
/**
 * Tolle Hilfsfunktionen
 *
 * @author Robert
 */

/**
 * A >= B
 * @param a Variable 1
 * @param b Variable 2
 */

int agteb(int a, int b);
```

Beispiel C-Datei (my.c)

```
#include "my.h"

/**
 * A >= B
 * @param a Variable 1
 * @param b Variable 2
 */

int agteb(int a, int b)
{
 if(a >= b)
  return 1;

 return 0;
}
```

Variablen

- int8_t
- uint16_t
- double
- float

sprintf und printf

printf

```
printf("%d\n", 2);
printf("%x", 0x42);
printf("%s: %d", "String", 123);
```

sprintf

```
uint8_t buffer[6];
sprintf(buffer, "%d",5);
printf("%s\n",buffer);
```

Contiki

Themen Tag 1

Grundlagen:

- Leds
- Timer
- Events
- Button
- Sensoren auslesen
- HowTo: Projekt
- Beispiele

Themen Tag 2

weitergehende Grundlagen:

- Protothread
- Prozesse
- IP / UDP
- rime
- cooja
- Beispiele

LED Konstanten

Verfügbare Konstanten

- LEDS_ALL
- LEDS_GREEN
- LEDS_YELLOW

LEDs

LEDs initialisieren

```
leds_init();
```

LEDs ansteuern

```
leds_on(LEDS_ALL);
leds_off(LEDS_ALL);
leds_toggle(LEDS_ALL);
```

Events

Auf ein Ereignis warten

```
PROCESS_WAIT_EVENT();   PROCESS_YIELD();
```

Auf ein Ereignis unter einer Bedingung warten (z.B. Timer)

```
PROCESS_WAIT_EVENT_UNTIL(condition c);
```

Post (a)synchronous event to a process.

```
PROCESS_POST(...);   PROCESS_POST_SYNCH(..);
```

Einfacher Timer

Timer, der wartet, bis eine Zeit abgelaufen ist

Timer definieren

```
struct timer myTimer;
```

Timer starten (1/10 Sekunde)

```
timer_set(&myTimer, CLOCK_SECOND / 10);
```

Timer neustarten

```
timer_restart(&myTimer);
timer_reset(&myTimer);
```

Timer abgelaufen

```
timer_expired(&myTimer);
```

Event Timer

Timer, sendet nach Ablauf ein Event

Timer definieren

```
struct etimer myTimer;
```

Timer starten (1/10 Sekunde)

```
etimer_set(&myTimer, CLOCK_SECOND / 10);
```

Timer neustarten

```
etimer_restart(&myTimer);
etimer_reset(&myTimer);
```

Timer abgelaufen

```
etimer_expired(&myTimer);
```

Callback Timer

Nach Ablauf des Timers wird eine Callback-Funktion aufgerufen

CTimer Beispiel

```
static struct ctimer myTimer;
static void (*light_off) = leds_off;
static unsigned char led = LEDS_YELLOW;
leds_init();
ctimer_set(&myTimer,  CLOCK_SECOND*2, light_off, &led);
```

Echtzeit Timer

Timer, der ein Event zu einem exakten Zeitpunkt auslöst.
ACHTUNG: Dieser Timer funktioniert unter INGA bisher noch nicht

```
struct rtimer myTimer;
```

Buttons

Button aktivieren

```
#include "dev/button-sensor.h";
SENSORS_ACTIVATE(button_sensor);
```

Auf Button warten

```
PROCESS_WAIT_EVENT_UNTIL(ev == sensors_events &&
 data == &button_sensor);
```

Hinweis: ev und data sind Parameter eines Prozesses

Druckluft- & Temperatur-Sensor

Library einbinden

```
#include "pressure-sensor.h"
```

Sensor initialisieren

```
SENSORS_ACTIVATE(pressure_sensor);
```

Werte auslesen

```
value = pressure_sensor.value(PRESS); // Druck
value = pressure_sensor.value(TEMP); // Temperatur
```

hello_world.c

Beispiel Programm

```
PROCESS(hello_world_process, "Hello world process");
AUTOSTART_PROCESSES(&hello_world_process);
PROCESS_THREAD(hello_world_process, ev, data)
{
  PROCESS_BEGIN();
  printf("Hello, world\n");
  while(1) {
    PROCESS_WAIT_EVENT();
 }
  PROCESS_END();
}
```

vom Code zum Ergebnis

Wie geht das?

Makefile

Makefiles werden benötigt um die INGA-Knoten zu programmieren

Listing 1: Makefile

```
1 CONTIKI = Pfad_zum_Contiki_Verzeichnis
2 all: Projekt_Dateiname_ohne_Extention
3
4 include $(CONTIKI)/Makefile.include
```

Listing 2: Makefile example

```
1 CONTIKI = ../..
2 all: hello-world
3
4 include $(CONTIKI)/Makefile.include
```

mehr Makefile

Erweiterung:

- für zuladende Apps
  `APPS=servreg-hack`
- für IPv6
  `WITH_UIP6=1`
- für floats in printf
  `LDFLAGS+=-Wl,-u,vfprintf -lprintf_flt`
- einiges mehr - siehe examples (in Instant Contiki)

Makefile Tipp

Makefile.target

```
echo "TARGET=inga" > Makefile.target
```

oder

```
make TARGET=inga savetarget
```

aus

```
make TARGET=inga my-project.upload
```

wird

```
make my-project.upload
```

INGA Knoten Programmieren

angenommenen der Code der Datei project.c soll geflashed werden

Alle INGA-Knoten flashen

```
make project.upload
```

oder

```
make project.upload
  MOTES=/dev/ttyUSB0,/dev/ttyUSB1,...
```

Debug-Ausgaben ansehen

```
make login MOTES=/dev/ttyUSB0
```

oder

```
make login
```

INGA Tool

Dient zum flashen ohne manuellen Reset

Pakete installieren

- Libusb 0.1
- Libftdi >= 0.19
- Libpopt
- Libudev

```
$ sudo apt-get libusb-0.1-4 libusb-dev libftdi1
libftdi-dev libpopt0 libpopt-dev libudev0 libudev-dev
```

INGA Tool

Kompilieren

```
$ cd tools/inga/inga_tool
$ make
```

INGA Tool

EEPROM von INGA Updaten

```
$ sudo ./inga_tool -device=/dev/ttyUSBx -f
```

Beispiel

```
$ sudo ./inga_tool -device=/dev/ttyUSB0 -f
Reading out EEPROM image...done
Writing updated EEPROM image...done
```

INGA Tool

Rechte setzen (/etc/udev/rules.d/30-inga-usb.rules)

```
 SUBSYSTEM=='usb', ATTRS{idVendor}=='0403',
ATTRS{idProduct}=='6001', ATTR{product}=='INGA',
GROUP='plugdev', OPTIONS+='last_rule'
```

Reset durchführen

```
 $ ./inga_tool -device=/dev/ttyUSB0 -r
Resetting INGA node...done
```

Zusammenfassung

der Weg zum Hello World ...

- `git clone git://git.ibr.cs.tu-bs.de/contiki-inga.git`
- `cd examples/hello-world/`
- `make TARGET=inga savetarget`
- Bord mitels gedrückten button und während dessen einschalten in den Programmiermodus bringen
- `make hello-world.upload`
- `make login`
- Knoten mit Schalter neustarten

Einführung

- Contiki-Kernel ist Event basiert
- Prozesse werden aufgerufen, sobald ein Event eintritt
- Events: Weniger Speicherverbrauch, da nur ein Stack für jeden Event Handler
- Threads: Ein Stack pro Thread

Protothreads - Einführung

- Mischung aus Events und Threads
- Ein Stack

Protothreads - Beispiel

Beispiel Programm

```
int a_prototheread(struct pt *pt) {
 PT_BEGIN(pt);
 PT_WAIT_UNTIL(pt, condition1);
 if(something) {
  PT_WAIT_UNTIL(pt, condition2);
 }
 PT_END(pt);
}
```

Prozesse

- Prozesse sind Protothreads

Beispiel

```
PROCESS_THREAD(hello_world_process, ev, data) {
 PROCESS_BEGIN();
 printf("Hello, world!\n");
 while(1) {
  PROCESS_WAIT_EVENT();
 }
 PROCESS_END();
}
```

Prozesse

Vorsicht

- Variablen werden nicht gespeichert
  Lösung: Lokale, statische Variablen
- Verwendung von switch vermeiden!

weiteres zu Events & Prozessen

process_alloc_event()

Globale Event-Nummer holen (> 128)

process_post(process_ptr, eventno, ptr);

Prozess später starten
Rückgabe: PROCESS_ERR_OK oder PROCESS_ERR_FULL

process_post_synch(process_ptr, eventno, ptr);

Prozess sofort starten

process_poll(process_ptr);

Sendet ein PROCESS_EVENT_POLL zum Prozess

void process_exit (struct process *p);

Stoppt einen Prozess und sendet ein Event an alle anderen Prozesse

Beispiel zu zwei Prozessen

```
static process_event_t event_data_ready;

PROCESS(temp_process, "Temperature process");
PROCESS(print_process, "Print process");

AUTOSTART_PROCESSES(&temp_process, &print_process);

PROCESS_THREAD(temp_process, ev, data) {
 event_data_ready = process_alloc_event();
 while(1) {
  process_post(&print_process, event_data_ready, &valid_measure);
 }
}

PROCESS_THREAD(print_process, ev, data) {
 while(1) {
  PROCESS_WAIT_EVENT_UNTIL(ev == event_data_ready);
  ...
 }
}
```

Contiki: 2 Kommunikations Stacks

Zwei Kommunikations Stacks in Contiki

- uIP – TCP/IP
- Rime – geringer overhead

Nutzungsmöglichkeiten der Stacks

- keinen
- einen der beiden
- beide

Contiki: uIP

Makefile:

- um uIP zu aktivieren: `CFLAGS += -DWITH_UIP=1`
- um IPv6 zu aktivieren: `UIP_CONF_IPV6=1`

Contiki: uIP

UDP:

- `udp_new()`
- **tcpip_event** bei neuer Verbinung, ankommenden Daten, etc
- zum senden: `uip_udp_packet_send()`

TCP:

- `tcp_connect()` und `tcp_listen()`
- **tcpip_event** bei neuer Verbinung, ankommenden Daten, etc
- die zusendenen Daten werden aus dem Parameter `appstate` genommen

uIP APIs

Zwei uIP APIs

eventorientiert:

- ”raw” uIP API
- eher für kleinere Programme
- explizite Zustandsmaschinen

mit Protosockets:

- eher für größere Programme
- sequenzieller Code

Rime - der Name

Namensherkunft:

- Frost bestehend aus vielen dünnen Schichten wird wohl so im englischen bezeichnet
- Rime - Reim (Bestandteil der Kommunikation)

Rime - ein leichtgewichter Kommunikations Stack

Ein Satz von Kommunikations Abstraktionen (Aufsteigend in der Komplexität):

- Anonymous best-effort single-hop broadcast (abc)
- Identified best-effort single-hop broadcast (ibc)
- Stubborn identified best-effort single-hop broadcast (sibc)
- Best-effort single-hop unicast (uc)
- Stubborn best-effort single-hop unicast (suc)
- Reliable single-hop unicast (ruc)
- Unique anonymous best-effort single-hop broadcast (uabc)
- Unique identified best-effort single-hop broadcast (uibc)

Rime - ein leichtgewichter Kommunikations Stack

Ein Satz von Kommunikations Abstraktionen (weitere):

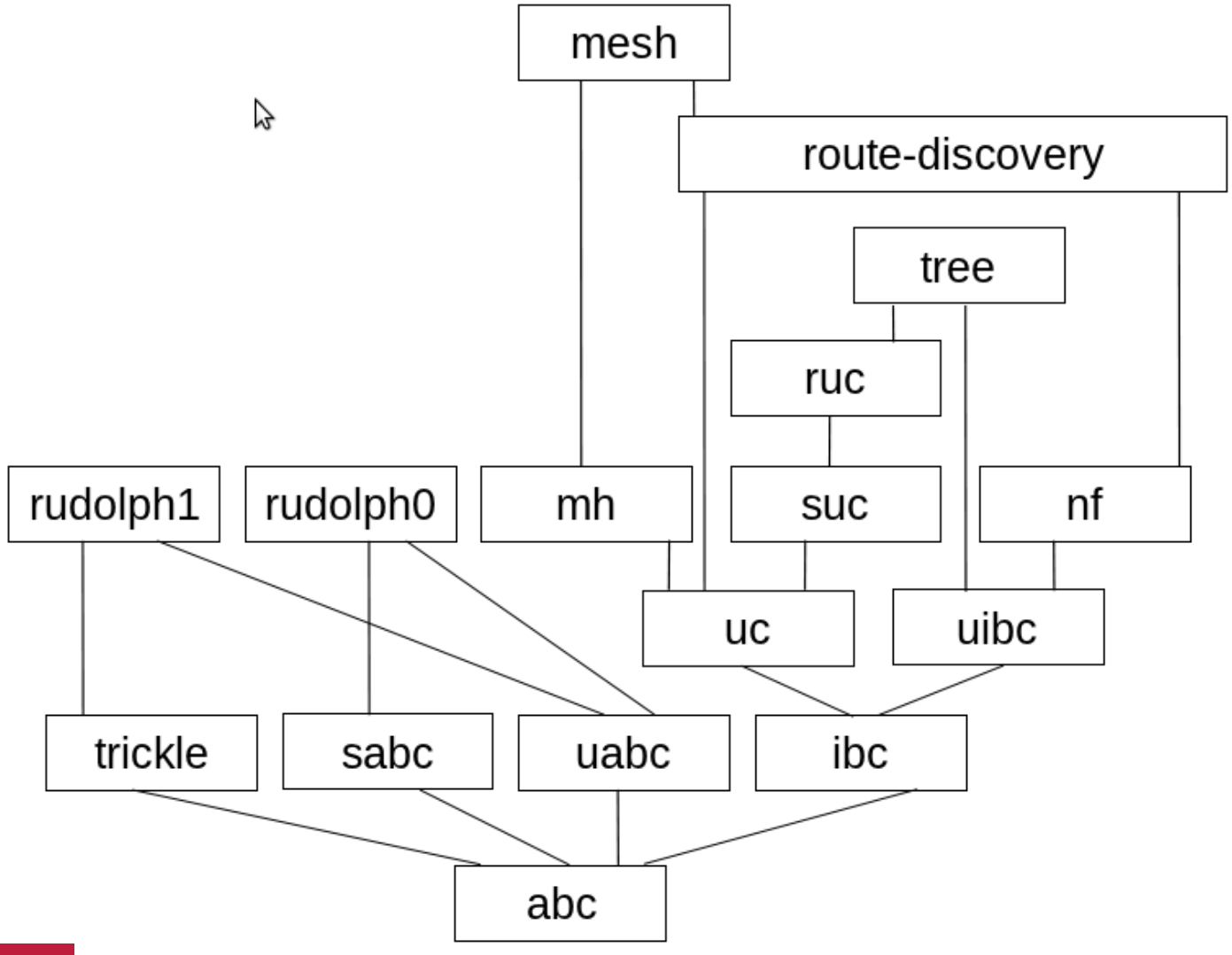
- Best-effort multi-hop unicast (mh)
- Best-effort multi-hop flooding (nf)
- Reliable multi-hop flooding (trickle)
- Hop-by-hop reliable mesh routing (mesh)
- Best-effort route discovery (route-disovery)
- Single-hop reliable bulk transfer (rudolph0)
- Multi-hop reliable bulk transfer (rudolph1)
- Hop-by-hop reliable data collection tree routing (tree)

Rime - Layer Vorteile

geringere Komplexität duch Layer:

- einfache Module (je 100 - 600 Byte Compiled Code)
- übersichtlicher
- kein komplett modulares Framework
  - daher gibt es Abhängigkeiten

Rime - Layer Abhängigkeiten



Rime - Channels

Kanäle:

- jede Kommunikation wird anhand eines 16Bit Kanals identifiziert
- Knoten müssen sich auf ein Modul pro Kanal einigen (uc <-> uc auf Kanal 5)
- Kanäle < 128 sind reserviert vom System

Rime - Programmiermodel

Callbacks:

- Module kommunizieren via Callbacks

Öffnen einer Verbindung mittesl eines Moduls:

- Argumente: Modul struct, channel, callbacks
- sobald was passiert wird ein Callback aufgerufen

Rime - Beispiel Broadcast

Anonymous best-effort single-hop broadcast:

```
void recv(struct abc_conn *c) {    /* Called when a */
  printf("Message received\n");    /* message is    */
}                                  /* received.     */

struct abc_callbacks cb = {recv}; /* Callback */
struct abc_conn c;                /* Connection */

void setup_sending_a_message_to_all_neighbors(void) {
  abc_open(&c, 128, &cb);         /* Channel 128 */
}

void send_message_to_neighbors(char *msg, int len) {
  rimebuf_copyfrom(msg, len);     /* Setup rimebuf */
  abc_send(&c);                   /* Send message  */
}
```

Rime - Beispiel Multi-Hop

Reliable multi-hop flooding:

```
void recv(struct trickle_conn *c) {    /* Called when a */
  printf("Message received\n");        /* message is    */
}                                      /* received.     */

struct trickle_callbacks cb = {recv}; /* Callback */
struct trickle_conn c;                /* Connection */

void setup_sending_a_message_to_network(void) {
  trickle_open(&c, 128, &cb);          /* Channel 128 */
}

void send_message_to_neighbors(char *msg, int len) {
  rimebuf_copyfrom(msg, len);          /* Setup rimebuf */
  trickle_send(&c);                    /* Send message  */
}
```

Node ID

Node ID setzen:

- notwendig für die Kommunikation!
- `cd examples/inga_demo/`
- `make TARGET=inga NODE_ID=<id> set_nodeid.upload`
  dabei ist <id> hexToDec(der auf den Knoten geschriebenen Hex-Zahl)

Zusammenfassung

Rime (siehe auch examples/rime):

- hat mehr Beispiele
- includes nicht vergessen (z.B. `#include "net/rime.h"`)

Cooja

- Simulation von mehreren Knoten (emulated MSP430 nodes) und räumlicher Verteilung
- Erweiterbarer Java-basierter Simulator

Wie geht es weiter?

Tipp

- Code lesen
- Beispiele anschauen
- Selber experementieren!

Links

- *http://www.sics.se/~adam/contiki-workshop-2007/workshop07programming.ppt*
- *http://www.sics.se/~adam/contiki/docs/main.html*

Wichtige Links

Praktikumsseite

https://www.ibr.cs.tu-bs.de/courses/ss12/wsn/index.html

Wiki

https://www.ibr.cs.tu-bs.de/trac/wsn/wiki/

Offizielle Seite

http://www.sics.se/contiki

Fragen & Probleme

Hiwis

- Robert Hartung
  hartung@ibr.cs.tu-bs.de
- Mailingliste
  wsn@ibr.cs.tu-bs.de

Wöchtenliche Treffen

Mi 13:15 - 14:45 im IZ G40

??? Do 13:15 - 14:45 oder Di 12:15 - 13:45 im IZ G40 ???

Mailingsliste

http://www.ibr.cs.tu-bs.de/mailman/listinfo/wsn

Wiki

https://www.ibr.cs.tu-bs.de/trac/wsn/wiki/

Danksagungen

Präsentation basiert u.a. auf

- den Folien von Thiemo Voigt vom Swedish Institute of Computer Science
- und der Contikieinführung von Adam Dunkels <adam@sics.se> aus dem Jahr 2007 (*http://www.sics.se/∼adam/contiki-workshop-2007/workshop07programming.ppt*)
- Unter Mithilfe von Karsten Hinz, HiWi im WS 2011/12