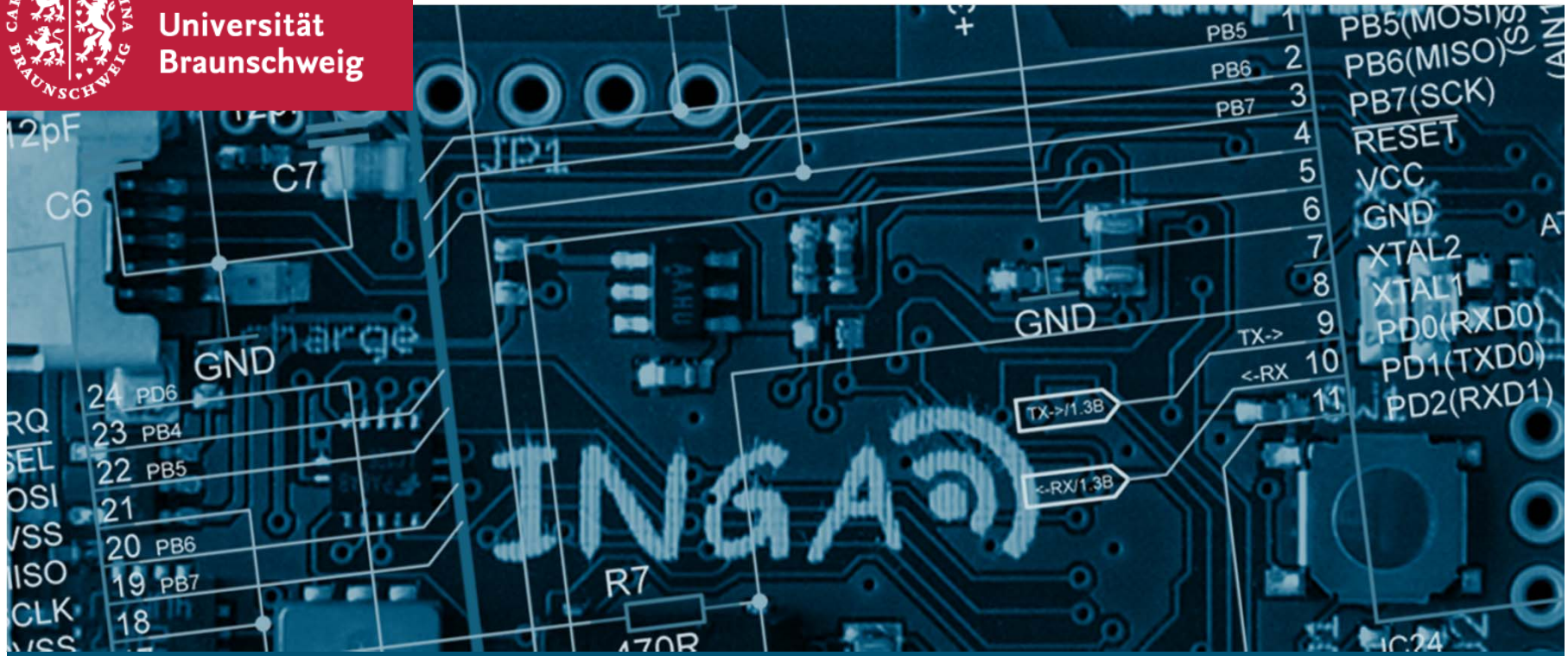
Technische Universität Braunschweig

Institut für Betriebssysteme und Rechnerverbund



Praktikum: Wireless Sensor Networks Tutorial

Keno Garlichs, Johannes van Balen, Yannic Schröder

Übersicht

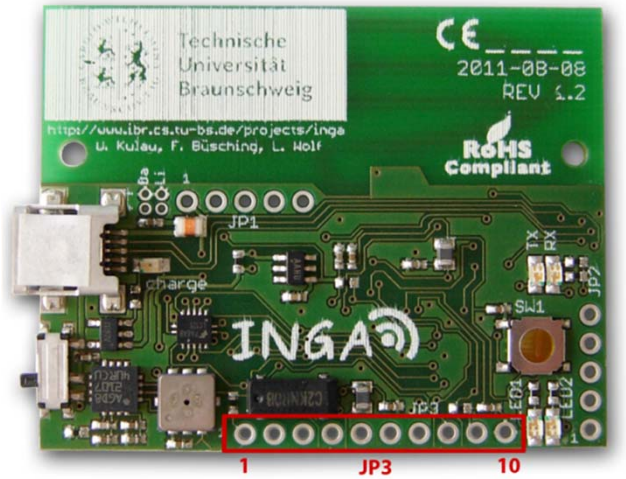
- Wer ist diese INGA eigentlich?
- AVR Mikrocontroller – wie wenig sind acht Bit?
- Entwicklungsumgebung
- git
- C-Crashkurs
- Contiki
- Organisatorisches

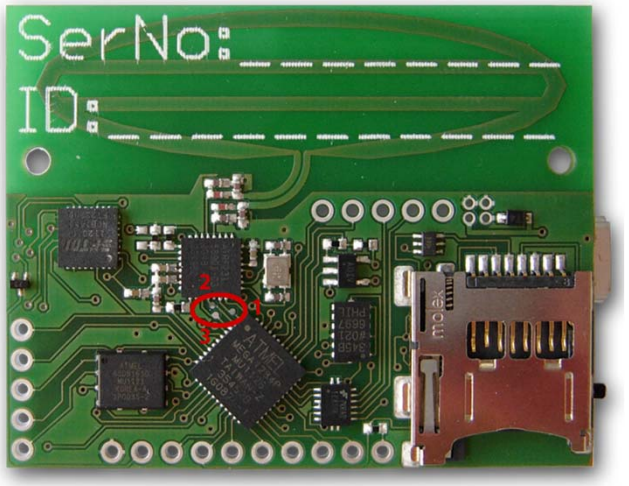
Wer ist diese INGA eigentlich?

- Inexpensive Node for General Applications
- ein preiswerter Sensorknoten
- basiert auf AVR Raven
- ATmega1284 Mikrocontroller
- 2,4 GHz Radio Transceiver
- diverse Sensoren
- diverse Schnittstellen

INGA - Features

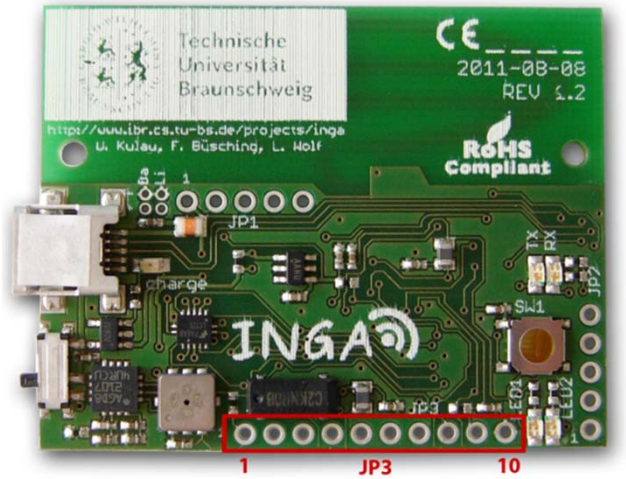
- kompatibel mit Contiki OS
  - programmiert in C
- über USB programmierbar
- SD-Karten Slot
- LEDs
- verschiedene Schnittstellen
  - I2C
  - SPI
  - AD-Wandler
  - UART

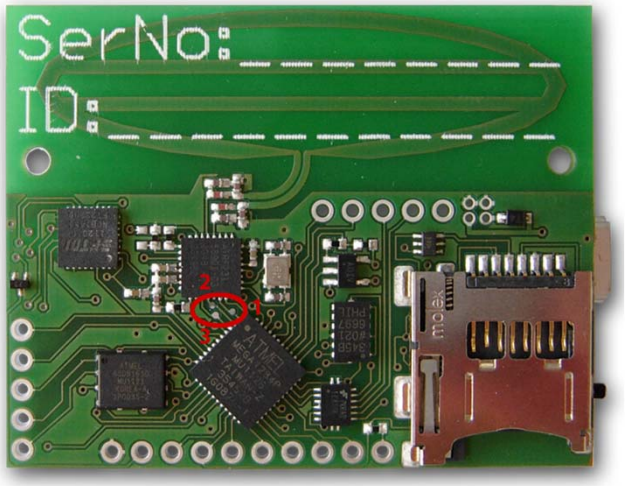




INGA - Sensoren

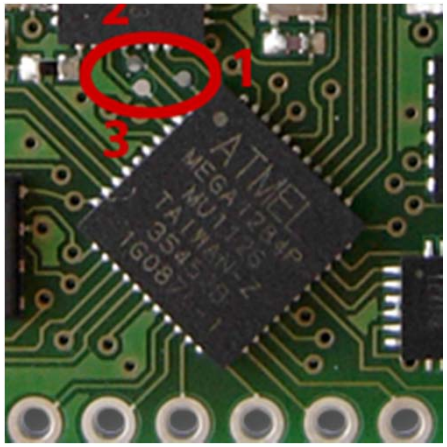
- Taster
- Temperatur (2x)
- Luftdruck
- optional
  - Accelerometer
  - Gyroskop

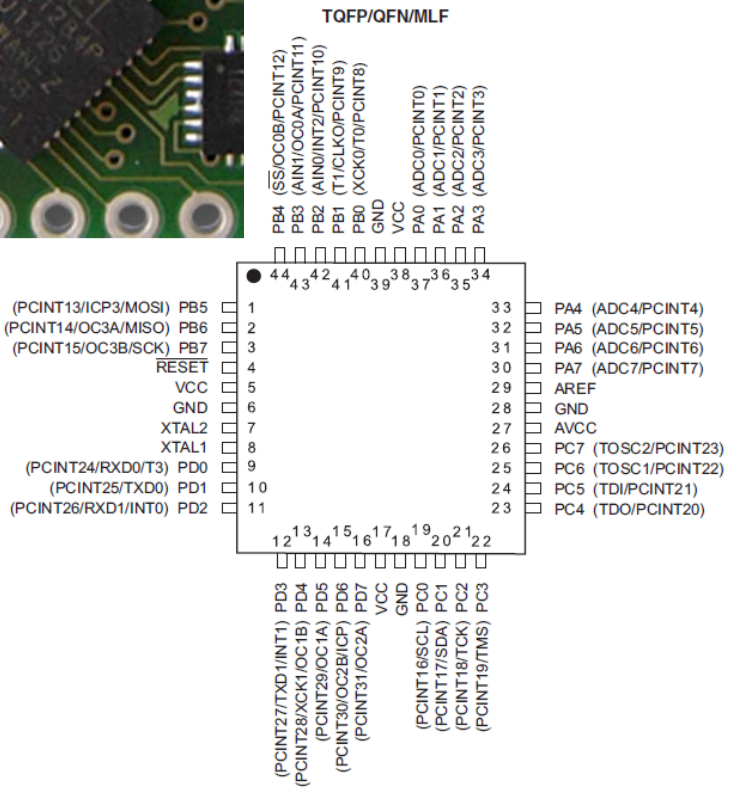




AVR Mikrocontroller

- 8 Bit Mikrocontroller
- 8 MHz Takt (max. 16 MHz)
- 128kB Flash
- 4kB EEPROM
- 16kB SRAM
- Hersteller: Atmel
- 5,90€ bei Reichelt
- einer der “größten” AVRs!





AVR Architektur

- Harvard-Architektur
  - Programm- und Datenspeicher sind getrennt
  - Programm im Flash, Daten im SRAM
  - Daten mit Trick auch im Flash (PROGMEM), aber nur Lesezugriff
- RISC-Befehlssatz
  - die meisten Befehle werden in einem Takt ausgeführt
  - Multiplikation dauert zwei Takte
  - Kein Divisionsbefehl!
    - Divisionen müssen aufwendig in Schleifen gelöst werden
  - Keine Floating-Point-Einheit!
    - Floating-Point muss auch mit Schleifen gelöst werden

Wie wenig sind acht Bit?

- 0xFF, 0b11111111
- Datenregister sind nur 8 Bit breit
- 0 ≤ uint8_t ≤ 255
- größere Variablen müssen in mehreren Takten berechnet werden
- Adressbus ist 16 Bit breit
  - um eine Speicheradresse anzusprechen werden je 8 Bit in das untere und das obere Adressregister geschrieben
- printf() ist sehr aufwendig
  - printf() mit floating point Unterstützung muss explizit gelinkt werden, sonst kann printf() nur Integer ausgeben
- uint32_t data[10000] ist eine doofe Idee (39kB)

Entwicklungsumgebung

Windows:

- VirtualBox mit Instant Contiki (Xubuntu)

Linux:

- VirtualBox mit Instant Contiki (Xubuntu)
- Ubuntu und Pakete selbst installieren (empfohlen)
  - gcc-avr, avrdude, avr-libc, git

Instant Contiki installieren

VirtualBox installieren (getestet mit 4.3.10)

- https://www.virtualbox.org/wiki/Downloads

Instant Contiki (.ova) herunterladen

- Link auf der Website zum Praktikum
- oder: https://www.ibr.cs.tu-bs.de/courses/ss14/wsn/download/Instant_Contiki_Xubuntu_14.04.ova

ova-Datei in VirtualBox importieren

- Doppelklick auf ova-Datei
- importieren

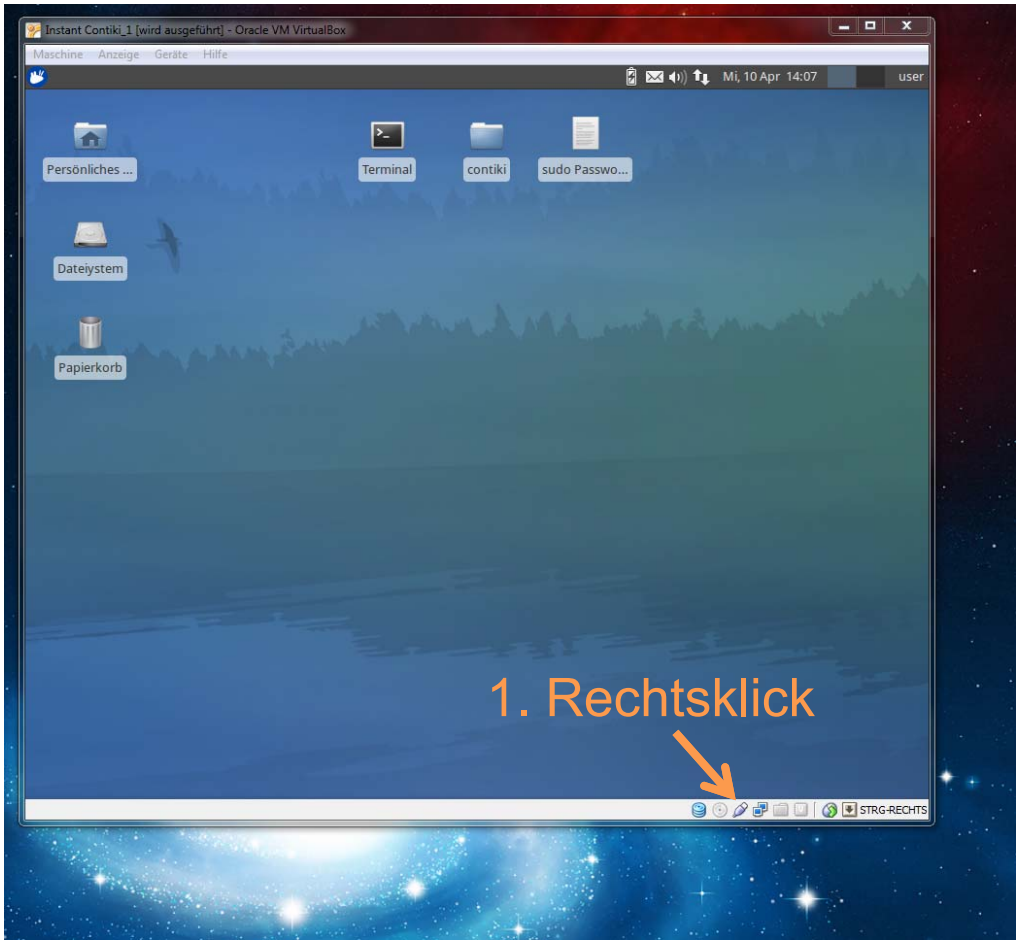
Instant Contiki benutzen

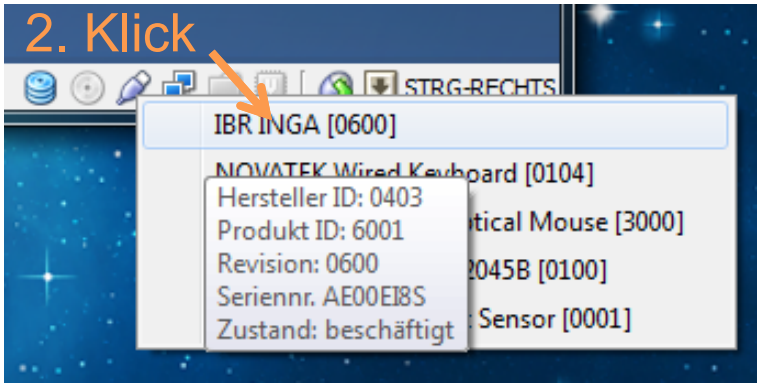
VM starten

- Benutzer: user
- Passwort: contiki

Contiki befindet sich auf dem Desktop (`~/Schreibtisch/contiki`)

INGA an Instant Contiki „anschließen“





Linux selbst einrichten

Pakete für AVR-Toolchain installieren

- `sudo apt-get install gcc-avr avrdude avr-libc git`

Rechte setzen

- `sudo gedit /etc/udev/rules.d/99-inga-usb.rules`
- `KERNEL=="ttyUSB[0-9]*", GROUP="plugdev", MODE="0666"`
- `sudo restart udev`

Contiki INGA Repository auschecken

- `git clone -b develop git://git.ibr.cs.tu-bs.de/project-cm-2012-inga-contiki.git contiki`
- `git branch -m develop master`

INGA Tool

- Flashen von INGA ohne manuellen Start des Bootloaders
  - manueller Bootloader: bei gedrücktem Taster einschalten (nach Programmieren aus- und wieder einschalten)

Abhängigkeiten installieren

- `sudo apt-get install libusb-0.1-4 libusb-dev libftdi1 libftdi-dev libpopt0 libpopt-dev libudev1 libudev-dev`

Kompilieren

- `cd tools/inga/inga_tool`
- `make`

INGA Tool

Rechte setzen

- `sudo gedit /etc/udev/rules.d/99-inga-usb.rules`
- `SUBSYSTEM=="usb", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="6001", ATTR{product}=="INGA", GROUP="plugdev"`
- `sudo restart udev`

Reset (testweise) durchführen

- `./inga_tool -d /dev/ttyUSBx -r`
- „x“ ist meist 0 für den ersten INGA

INGA flashen

- In diesem Beispiel soll die Datei „`project.c`“ geflashed werden

Target speichern

- `make TARGET=inga savetarget`

Alle angeschlossenen INGAs flashen

- `make project.upload`

oder um bestimmte INGAs zu flashen

- `make project.upload MOTES=/dev/ttyUSB0,/dev/ttyUSB42`

Debug-Ausgaben (printf) anzeigen

- `make login`

oder für bestimmten INGA

- `make login MOTES=/dev/ttyUSB0`

makefile

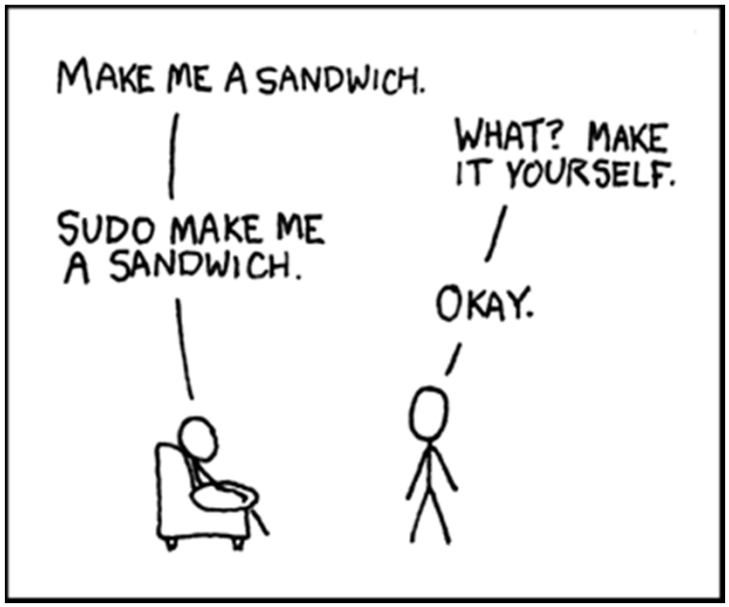
- im Makefile steht wo Contiki liegt, wie das Projekt heißt und was Kompiler und Linker sonst noch so beachten sollen

Inhalt eines Makefiles:

```
CONTIKI = Pfad_zum_Contiki_Verzeichnis
all: Projekt_Dateiname_ohne_Extention

include $(CONTIKI)/Makefile.include
```

Beispiel

```
CONTIKI = ../..
all: hello-world

include $(CONTIKI)/Makefile.include
```



makefile - Optionen

Apps laden

- `APPS=servreg-hack`

IPv6 verwenden

- `WITH_UIP6=1`

printf mit float-Unterstützung

- `LDFLAGS+=-Wl,-u,vfprintf -lprintf_flt`

weitere Möglichkeiten in den `examples`

git

- Verteile Versionsverwaltung
- Contiki wird mit git entwickelt
- IBR erweitert das Repository für INGA
  - Unterstützung der speziellen Hardware von INGA

- Anonymes Checkout
- `git clone -b develop git://git.ibr.cs.tu-bs.de/project-cm-2012-inga-contiki.git contiki`
- `git branch -m develop master`

- Branch „develop“ ist der aktuellste Branch und wird demnächst neuer Master vom IBR

git - commit

- lokal
  - `git add foo.c`
  - `git commit -a`
    - Commit-Nachricht eingeben

- Aktualisieren des Remote-Repositories
  - `git pull origin master`
  - `git push origin master`

git - branch

Zum testen von Features sind Branches hilfreich

Liste aller Branches anzeigen

- `git branch -a`
- der aktuelle Branch ist mit * gekennzeichnet

Branch wechseln

- `git checkout <branchname>`

Branches zusammenführen

- `git merge <branchname>`

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

git - tools

- weitere nützliche Befehle
  - `git diff`
  - `git diff –cached`
  - `git log`
  - `git status`

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

C-Crashkurs

- imperative Programmiersprache
- erfunden 1972 von Dennis Ritchie

- Header-Dateien (.h) deklarieren Funktionen
- C-Dateien (.c) implementieren Funktionen

```
#include <stdio.h>
#include <stdlib.h>

uint8_t main(void)
{
    printf("Hallo Welt!\n");
    return 0;
}
```

Header Beispiel – helper_functions.h

```
#ifndef HELPER_FUNCTIONS_H
#define HELPER_FUNCTIONS_H

#define MY_CONSTANT 42  // this is an inline comment

/* this is a block comment

it spans multiple lines */

uint8_t is42(uint8_t check);

void add42(uint8_t *sum);

#endif // HELPER_FUNCTIONS_H
```

C Beispiel – helper_functions.c

```c
#include "helper_functions.h"

uint8_t is42(uint8_t check)
{
  if (check == MY_CONSTANT)
  {
    return 1; // true, returns 1, the avr does not know "boolean"
              // every boolean is 8 Bits (uint8_t)
  }
  else
  {
    return 0; // not true, return 0
  }
}

void add42(uint8_t *sum)
{
  *sum += 42;
}

#endif // HELPER_FUNCTIONS_H
```

Pointer

- Pointer sind Speicheradressen und zeigen somit auf ein Byte im Speicher

```
uint8_t a;       // normale Variable a
uint8_t *b = &a; // b ist ein Pointer auf die Adresse von a

*b = 5; // a = 5

uint8_t c[5];   // Array mit 5 Werten, belegt 5 Byte im Speicher
a = c[0];       // zufälliger Wert wird a zugewiesen, c[0] ist nicht initialisiert!
a = c[42];      // Zugriff findet ohne Fehler statt, Wert wird aus Speicher hinter c genommen

c[42] = 0;      // der Speicher HINTER c wird verändert, NIEMALS MACHEN!!!

uint8_t *d = &c[2]; // d zeigt auf c[2]
d += 1;             // d zeigt auf c[3]
```

Datentypen

- `uint8_t, uint16_t, uint32_t, uint64_t`
- `int8_t, int16_t, int32_t, int64_t`
- `float` (32 Bit)

- `(unsigned) char, short, int, long, long long`
  - unübersichtlich, die Anzahl der Bits ist nicht erkennbar

- `double` ist genauso groß wie `float` (32 Bit)

printf Debugging

- die Standardausgabe ist auf die serielle Schnittstelle (USB) umgeleitet
- Debugging per `printf()` möglich

```
#include <stdio.h>

uint8_t main()
{
  printf("%d\n", 0x2); // gibt eine 2 auf der Konsole aus, Zeilenumbruch am Ende
  printf("%x", 42); // gibt die Zahl 42 hexadezimal aus
  printf("%s ist %d\n", "Der Wert", 42); // gibt "Der Wert ist 42" aus (mit
Zeilenumbruch)
  uint8_t buffer[6] = {0};  // erzeuge 6 Zeichen Puffer, initialisiert mit 0
  sprintf(buffer, "%d", 5); // schreibe eine 5 als Zeichen in den Puffer
  printf("%s\n", buffer);   // gibt den Inhalt von Buffer als String aus
                            // Strings enden beim ersten Byte das 0 ist
}
```

Mikrocontroller Rechentricks

```
uint8_t a = 0b00001111;

// ein Bit in einem Byte setzen
a = a | (1<<7);                          // a = 0b10001111
// ein Bit löschen
a &= ~(1<<1);                            // a = 0b10001101
// die unteren 3 Bit ausschneiden
a &= 0b00000111;                         // a = 0b00000101
// die oberen 5 Bit setzen
a |= 0b11111000;                         // a = 0b11111101 = 253
// durch 4 teilen = Bitshift um 2 nach unten
a = a >> 2;                              // a = 0b00111111 = 63 (253 / 4.f = 63,25)

// mehr Tricks unter: http://www.mikrocontroller.net/articles/AVR-GCC-Tutorial
```

Contiki

Themen an Tag 1

- LEDs
- Timer
- Button
- Sensoren auslesen
- Watchdog

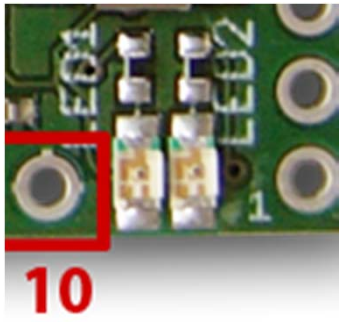
Contiki

Themen an Tag 2

- Prozesse
- Events
- uIP
- Rime
- HowTo: Ein Projekt kompilieren
- Cooja - Netwerksimulator

LEDs

```
#include "leds.h"
// LED Konstanten (sind bereits in leds.h definiert)
#define LEDS_ALL      7   // beide LEDs
#define LEDS_GREEN    1   // grüne LED
#define LEDS_YELLOW   2   // orange LED

// LEDs initialisieren
leds_init();              // muss nur einmal aufgerufen werden

// LEDs ansteuern
leds_on(LEDS_ALL);        // alle LEDs einschalten
leds_off(LEDS_GREEN);     // grüne LED ausschalten
leds_invert(LEDS_YELLOW); // orange LED umschalten
// mehr Infos in examples/inga/demo/led_demo.c
```



Einfacher Timer

```
// Timer, der wartet bis eine Zeit abgelaufen ist

// Timer definieren
static struct timer myTimer;

// Timer starten (~1/10 Sekunde)
timer_set(&myTimer, CLOCK_SECOND / 10); // Integer-Division!

// Timer neustarten
timer_restart(&myTimer);   // Timer auf 1/10 Sekunde neu laden
timer_reset(&myTimer);     // Timer neu starten vom letzten Ablaufzeitpunkt

// Prüfen ob Timer abgelaufen ist
timer_expired(&myTimer);
```

Event Timer

```
// Timer, der der nach Ablauf ein Event sendet

// Timer definieren
static struct etimer myTimer;

// Timer starten (~1/10 Sekunde)
etimer_set(&myTimer, CLOCK_SECOND / 10); // Integer-Division!

// Timer neustarten
etimer_restart(&myTimer);  // Timer auf 1/10 Sekunde neu laden
etimer_reset(&myTimer);    // Timer neu starten vom letzten Ablaufzeitpunkt

// Prüfen ob Timer abgelaufen ist
etimer_expired(&myTimer);
```

Callback Timer

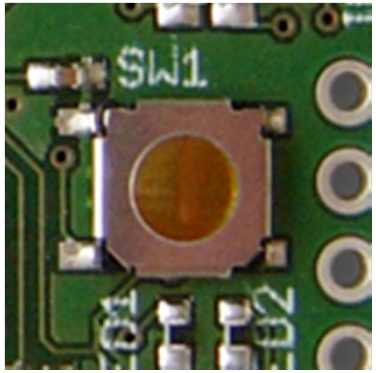
```
// Nach Ablauf des Timers wird
// eine Callback-Funktion aufgerufen

// Timer definieren
static struct ctimer myTimer;
// Pointer auf die Funktion die als Callback aufgerufen werden soll
static void (*light_off) = leds_off;
// Parameter für den Funktionsaufruf
static unsigned char led = LEDS_YELLOW;
// LED initialisieren und einschalten
leds_init();
leds_on(LEDS_YELLOW);
// Timer starten
ctimer_set(&myTimer, CLOCK_SECOND*2, light_off, &led);
```

Echtzeit Timer

```
// Timer, der ein Event zu einem
// exakten Zeitpunkt auslöst

// Timer definieren
static struct rtimer myTimer;

// funktioniert mit INGA (noch) nicht!
```

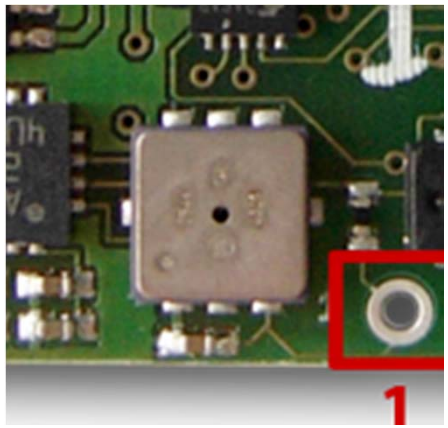
Button

```
#include "button-sensor.h";
// Pointer auf Sensor holen
static const struct sensors_sensor *button_sensor;
button_sensor = sensors_find("Button");
// Button aktivieren
uint8_t status = SENSORS_ACTIVATE(*button_sensor);

// Auf Button warten
// Prozess gibt Kontrolle ab und wartet darauf, dass der Button ein Event sendet
PROCESS_WAIT_EVENT_UNTIL(ev == sensors_event && data == &button_sensor);
// Button Events werden bei drücken und loslassen generiert!
// ev und data sind Parameter des Prozesses
// mehr Infos in examples/inga/sensors/button-example.c
```



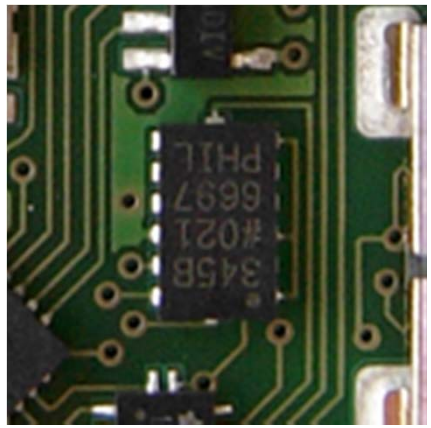
Luftdruck- & Temperatursensor

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

// Pointer auf Sensor holen
static const struct sensors_sensor *temppress_sensor;
temppress_sensor = sensors_find("Press");
// Sensor aktivieren
uint8_t status = SENSORS_ACTIVATE(*temppress_sensor);

// Werte auslesen
// Druck
int32_t pressure = ((int32_t) temppress_sensor->value(PRESS_H) << 16);
pressure |= (temppress_sensor->value(PRESS_L) & 0xFFFF);
int16_t temperature = temppress_sensor->value(TEMP);  // Temperatur
// mehr Infos in examples/inga/sensors/temp-pressure-example.c
```

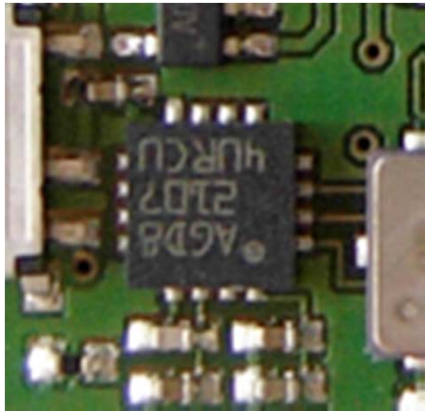


Accelerometer (Beschleunigungssensor)

```
#include "acc-sensor.h"
// Pointer auf Sensor holen
static const struct sensors_sensor *acc_sensor;
acc_sensor = sensors_find("Acc");
// Sensor aktivieren
uint8_t status = SENSORS_ACTIVATE(*acc_sensor);
// Konfigurieren
acc_sensor->configure(ACC_CONF_SENSITIVITY, ACC_2G);
acc_sensor->configure(ACC_CONF_DATA_RATE, ACC_100HZ);
// Werte auslesen
int16_t x = acc_sensor->value(ACC_X); // x-Achse
int16_t y = acc_sensor->value(ACC_Y); // y-Achse
int16_t z = acc_sensor_>value(ACC_Z); // z-Achse
// mehr Infos in examples/inga/sensors/acc-example.c
```



Gyroskop (Winkelgeschwindigkeit)



```
#include "gyro-sensor.h"
// Pointer auf Sensor holen
static const struct sensors_sensor *gyro_sensor;
gyro_sensor = sensors_find("Gyro");
// Sensor aktivieren
uint8_t status = SENSORS_ACTIVATE(*gyro_sensor);
// Konfigurieren
gyro_sensor->configure(GYRO_CONF_SENSITIVITY, GYRO_250DPS);
gyro_sensor->configure(GYRO_CONF_DATA_RATE, GYRO_100HZ);
// Werte auslesen
int16_t x = gyro_sensor->value(GYRO_X); // x-Achse
int16_t y = gyro_sensor->value(GYRO_Y); // y-Achse
int16_t z = gyro_sensor->value(GYRO_Z); // z-Achse
// mehr Infos in examples/inga/sensors/gyro-example.c
```

Watchdog

- Der Watchdog zieht den Reset des AVR, wenn der Watchdog-Timer abläuft
- Verhindert das „aufhängen“ eines Programms
- Der Timer muss während der ordnungsgemäßen Programmausführung regelmäßig zurückgesetzt werden
- Contiki kümmert sich um das Zurücksetzen des Timers
- Wenn man weiß was man tut: `wdt_disable();`

Hello World vom INGA

```
// definiere einen neuen Prozess mit den Namen "Hello World process"
PROCESS(hello_world_process, "Hello World process");

AUTOSTART_PROCESSES(&hello_world_process);      // starte den Prozess bei Systemstart

PROCESS_THREAD(hello_world_process, ev, data)        // Implementation des Prozesses
{
  PROCESS_BEGIN();                                   // Start-Makro

  printf("Hello World!\n"); // Schreibe "Hello World!" auf die serielle Schnittstelle
  while(1) {                                         // Endlosschleife
    PROCESS_PAUSE();     // Gebe die Kontrolle ab (verhindert ablaufen des Watchdogs)
  }

  PROCESS_END();                                     // End-Makro
}
```

Hello World kompilieren und flashen

Ins Verzeichnis des Hello World Beispiels wechseln

- `cd examples/hello-world/`

Ziel als inga setzen und speichern

- `make TARGET=inga savetarget`

Kompilieren

- Knoten in Bootloader setzen (wahlweise automatisch mit inga-tool)
- `make hello-world.upload`

Login auf den Knoten (Konsolenausgabe ansehen), vorher neu starten!

- `make login`

Wenn inga-tool läuft

- `make hello-world.upload login`

Contiki - Prozesse

- Kernel ist Event basiert
- Prozesse werden aufgerufen, wenn ein Event für sie auftritt
  - z.B. wenn einer etimer abläuft
- Scheduling ist NICHT preemptiv
  - Prozesse laufen bis sie freiwillig die Kontrolle wieder abgeben
  - oder der Watchdog zuschlägt

Prozesse

- Variablen werden nicht gespeichert, wenn der Scope eines Prozesses verlassen wird
  - Lösung: lokale, statische Variablen (`static`)
- Verwendung von `switch` vermeiden
  - Prozesse werden durch `switch` unterbrochen und weitergeführt
  - besser: `if/(else if)/else`

```
// Prozess starten
process_start (struct process *p, const char *arg)

// Prozess beenden
process_exit (struct process *p)
```

Events - Senden

```
// Globale Eventnummer holen
process_event_t myEvent;
myEvent = process_alloc_event();

// asynchrones Event an Prozess senden (wird beim nächsten Aufruf des
Prozesses bearbeitet)
process_post(&myProcess, myEvent, NULL);
// an Stelle von NULL beliebiger Pointer möglich

// synchrones Event senden (wird sofort bearbeitet)
process_post_synch(&myProcess, myEvent, NULL);
```

Events - Empfangen

```
// Auf ein Event warten
PROCESS_WAIT_EVENT();
PROCESS_YIELD();

// Auf ein Ereignis unter einer Beding warten (z.B. auf Timer)
PROCESS_WAIT_EVENT_UNTIL(condition c);

// Prozess kurz unterbrechen (um so schnell wie möglich weiter zu rechnen)
PROCESS_PAUSE();
```

Zwei Prozesse - Beispiel

```
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);
    ...
  }
}
```

Kommunikation - Stacks

Zwei Stacks in Contiki verfügbar

- uIP (TCP/IP)
- Rime (leichtgewichtig)

Nutzungsmöglichkeiten der Stacks

- keinen
- einen
- (beide)

Ports

- HTONS() benutzen

uIP

Makefile

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

UDP

- `udp_new()`
- `tcpip_event` bei neuer Verbindung, ankommenden Daten, etc.
- zum Senden: `uip_udp_packet_send()`

TCP

- `tcp_connect()` und `tcp_listen()`
- `tcpip_event` bei neuer Verbindung, ankommenden Daten, etc.
- die zusendenden Daten werden aus dem Parameter `appstate` genommen

uIP - APIs

- zwei APIs je nach „Geschmack“

eventorientiert:

- „raw“ uIP API
- eher für kleinere Programme
- explizite Zustandsmaschinen

mit Protosocket:

- eher für größere Programme
- sequentieller Code

Rime - ein leichtgewichtiger Kommunikations-Stack

Verschiedene Abstraktionsstufen der Kommunikation (aufsteigend in ihrer 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 leichtgewichtiger Kommunikations-Stack

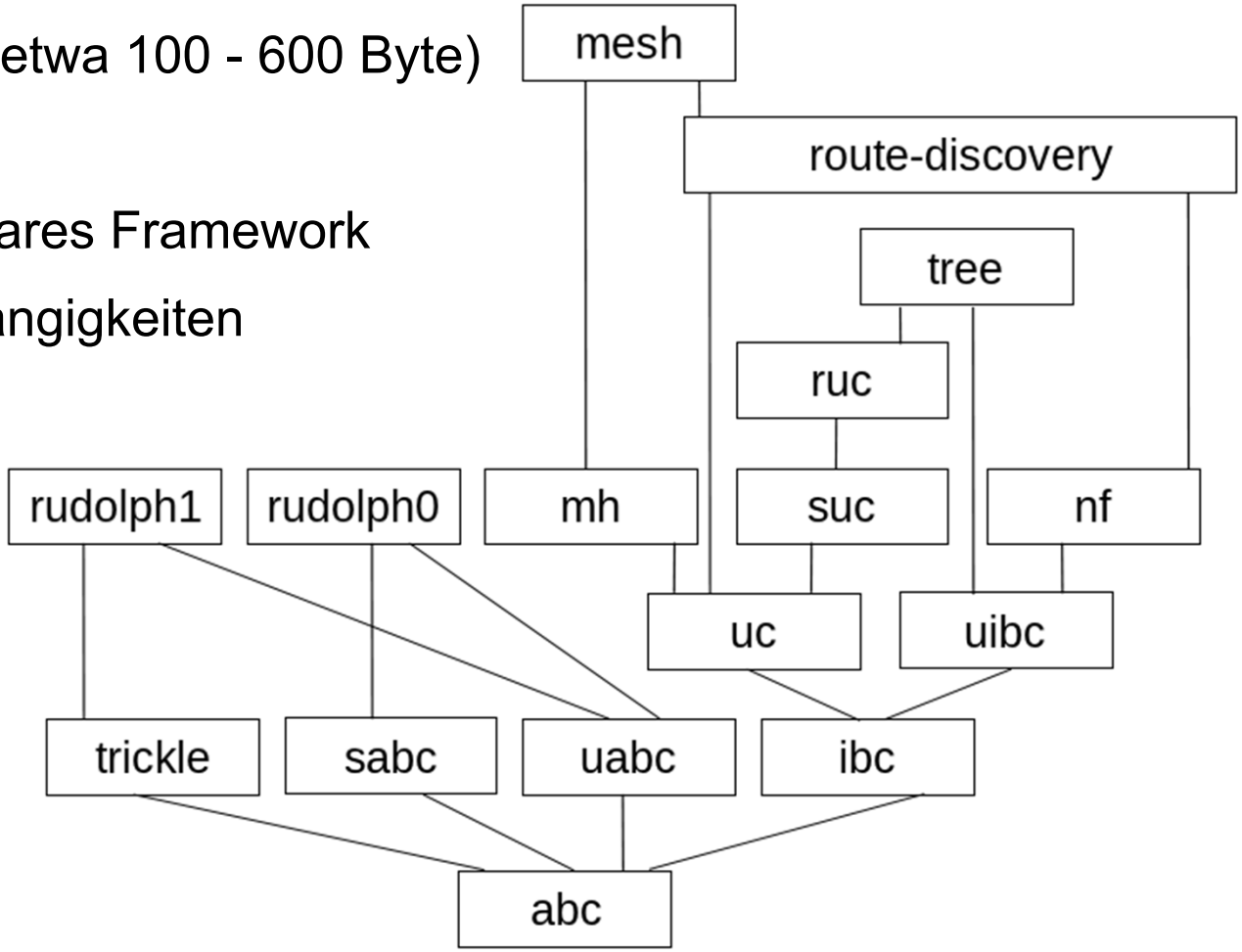
Weitere Abstraktionsstufen der Kommunikation:

- 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 durch Layer

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



Rime - Channels

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

- nicht verwechseln mit Sendefrequenzen im 2,4 GHz Band!

Rime - Programmiermodell

Callbacks

- Module kommunizieren via Callbacks

Öffnen einer Verbindung mittels eines Moduls

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

Beispiele

- gute Beispiele in `examples/rime`
- `#include "net/rime.h"` nicht vergessen!

Rime – Broadcast Beispiel

```
// Anonymous best-effort single-hop broadcast
// Called when a message is received
void recv(struct abc_conn *c) {
  printf("Message received\n");
}

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 - Multi-Hop Beispiel

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

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

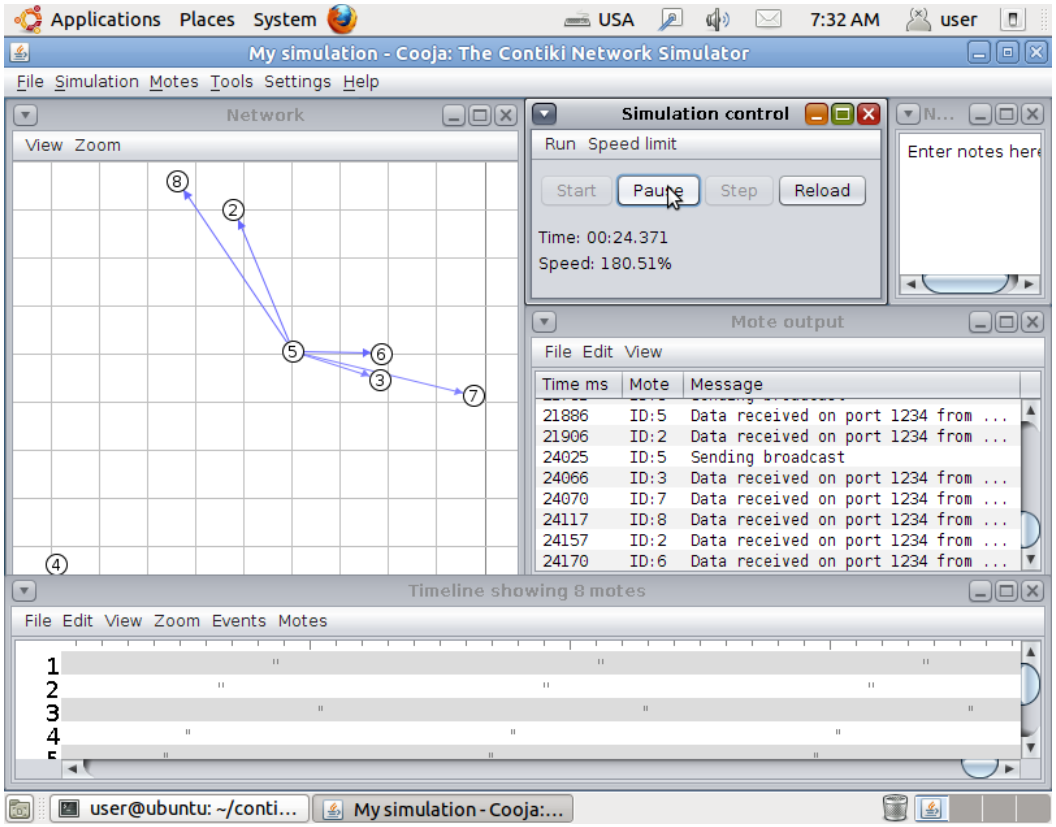
- die Node ID ist notwendig für die Kommunikation
- meist ist diese bereits richtig im EEPROM gesetzt
- Contiki meldet beim Boot die Node ID
- Node ID ist auf den INGAs vermerkt (hexadezimale Zahl!)

Node ID neu setzen

- `cd examples/inga/node-setup/`
- `make NODE_ID=<id> setup`
  - dabei ist <id> die auf den Knoten geschriebene Hex-Zahl (z.B. 0x42)

Cooja

- Simulation von mehreren Knoten und räumlicher Verteilung
- Erweiterbarer Java-basierter Simulator



Und nun?

- Code lesen
- Beispiele anschauen
  - `examples/inga/demo`
  - `examples/inga/sensors`
  - `examples/hello_world`
  - `examples/rime`
  - `examples/udp-ipv6`
  - `examples/inga/net`
- Selber experimentieren!

Wichtige Links

Praktikumsseite

- https://www.ibr.cs.tu-bs.de/courses/ss14/wsn/index.html

INGA Wiki

- http://trac.ibr.cs.tu-bs.de/project-cm-2012-inga

Doxygen von Contiki 3.x

- http://ejoerns.github.io/contiki-inga/doxygen/

Offizielle Contiki Homepage

- http://www.contiki-os.org

AVR-GGC-Tutorial von mikrocontroller.net

- http://www.mikrocontroller.net/articles/AVR-GCC-Tutorial

Fragen & Probleme

Hiwis

- Keno Garlichs (garlichs@ibr.cs.tu-bs.de)
- Johannes van Balen (vanbalen@ibr.cs.tu-bs.de)
- Yannic Schröder (yschroed@ibr.cs.tu-bs.de)

Sprechstunde

- keine feste Uhrzeit
- einfach im Mikrocontroller-Labor vorbeikommen (IZ 147)
- oder Termin per Mail vereinbaren

Mailingliste & Wiki

Mailingliste

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

Wiki zum Praktikum

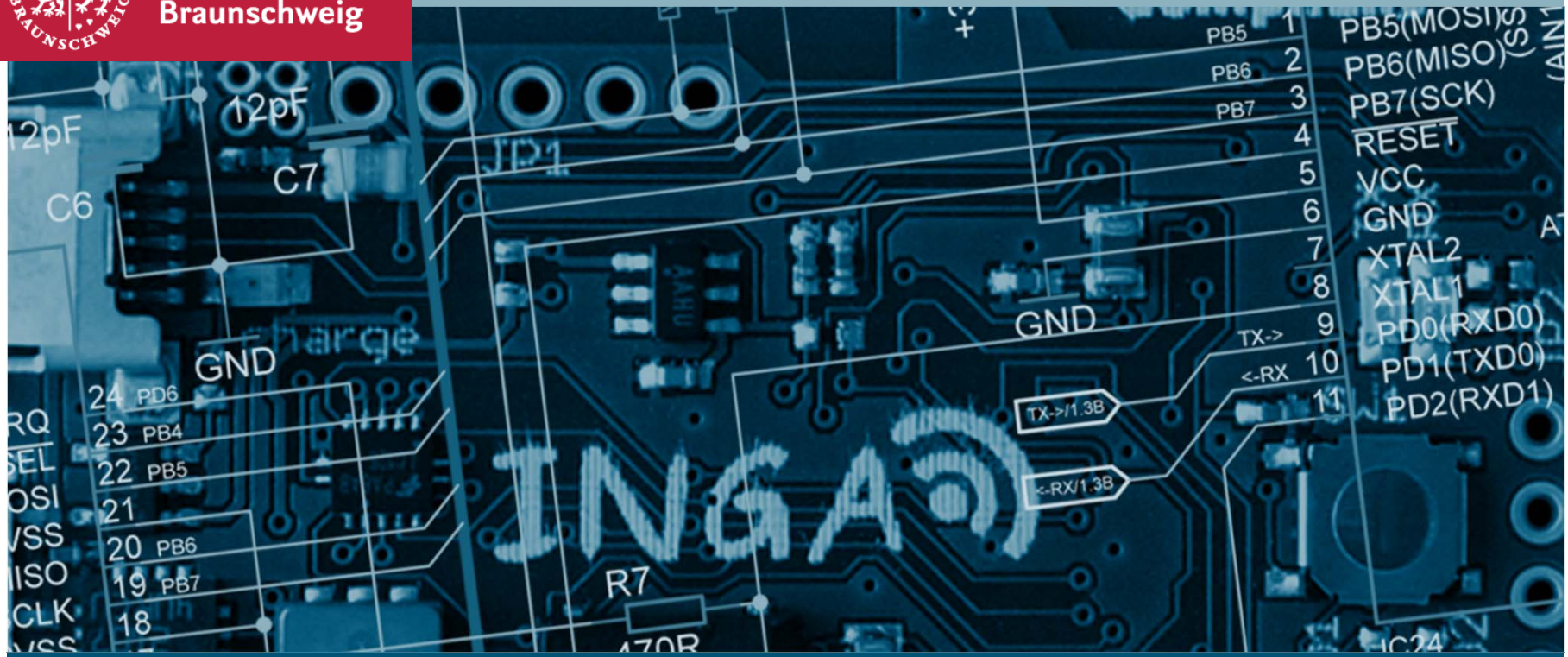
- https://trac.ibr.cs.tu-bs.de/course-cm-wsn
- Anmeldung mit y-IBR-Account!

Acknowledgements

Präsentation basiert auf

- Folien von Thiemo Voigt vom Swedish Institute of Computer Science
- Contiki Einführung von Adam Dunkels aus dem Jahr 2007 (http://www.sics.se/adam/contiki-workshop-2007/workshop07programming.ppt)
- Karsten Hinz, Hiwi im WS 2011/12
- Robert Hartung, Hiwi im WS 2012/13

Technische Universität Braunschweig



Viel Spaß beim hacken!