Technische Universität Braunschweig

Wireless Sensor Networks

Praktikum WSN-Lab - Tutorial

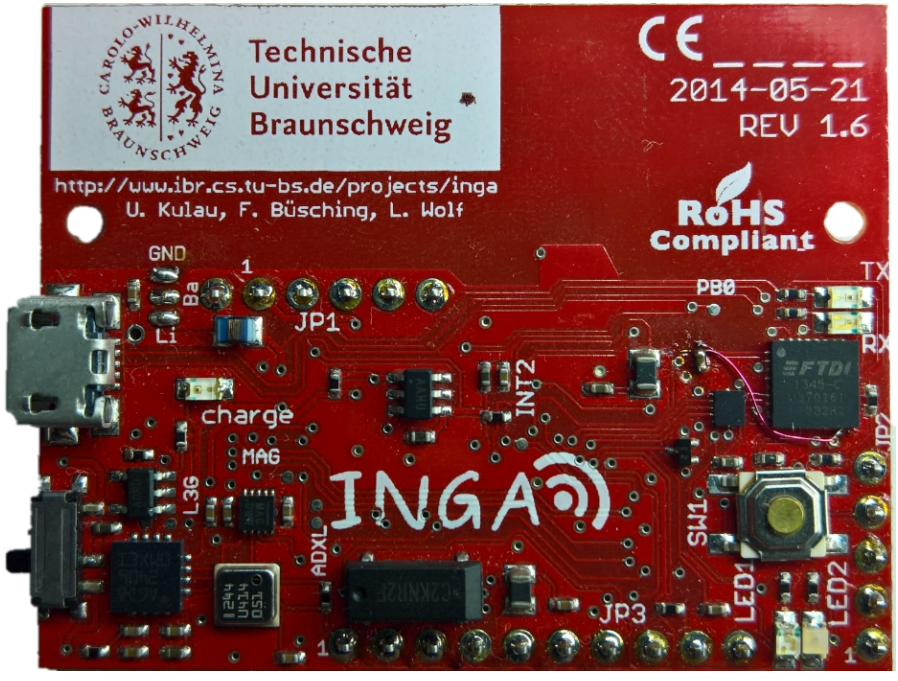
IBR - Connected and Mobile Systems, 17. April 2018

Übersicht

- Wer ist diese INGA eigentlich?
- AVR Mikrocontroller - Wie wenig sind acht Bit?
- Entwicklungsumgebung
- Git
- C-Crashkurs
- Contiki
- Organisatorisches

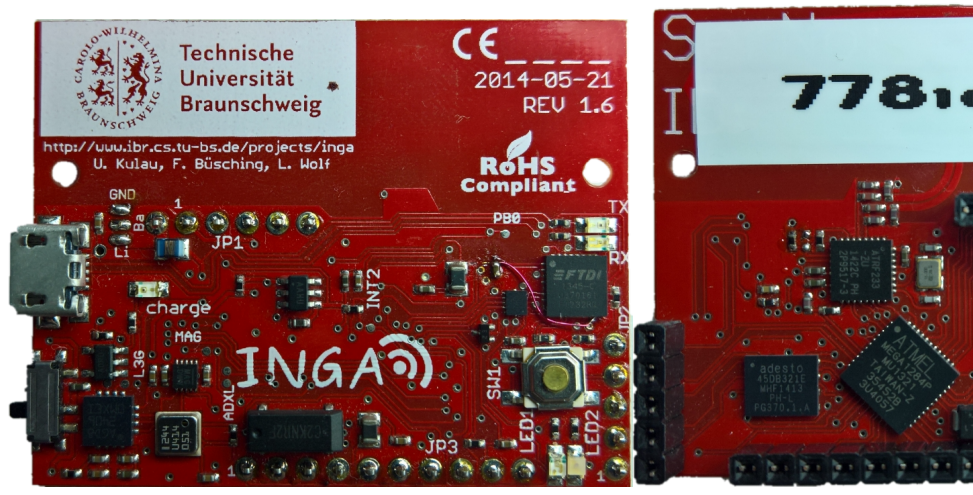
Wer ist diese INGA eigentlich?

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



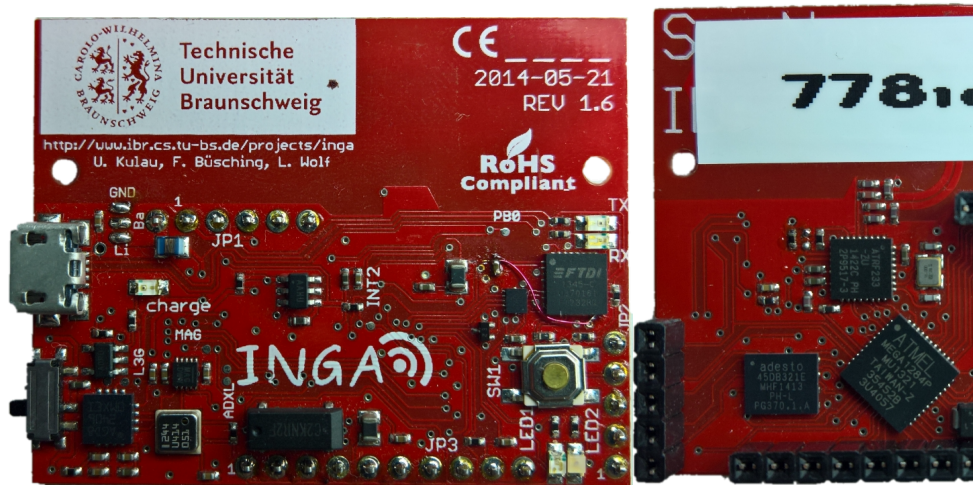
INGA - Features

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



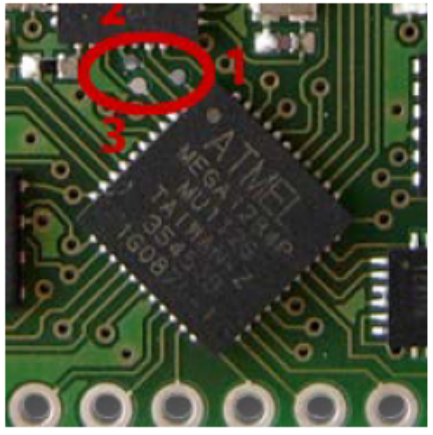
INGA - Sensoren

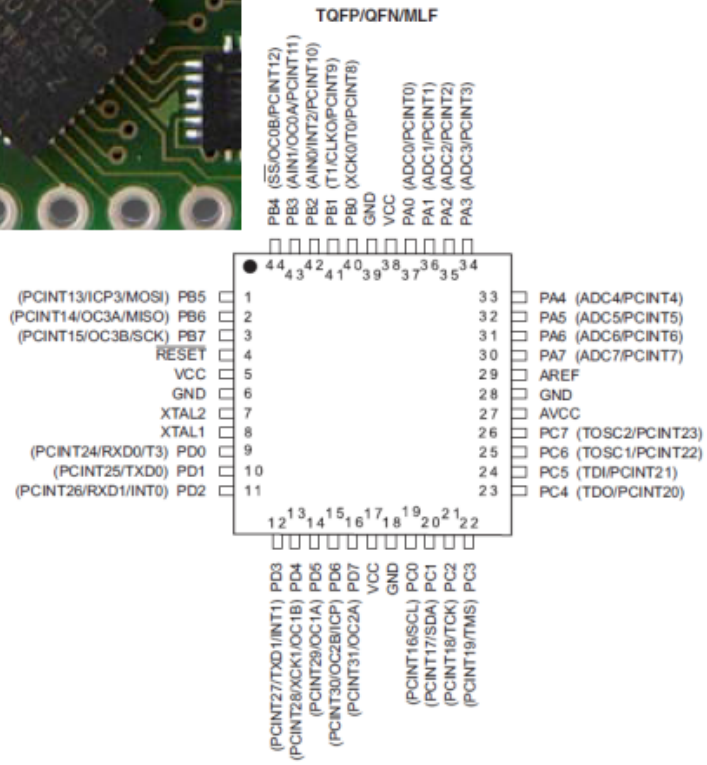
- Taster
- Temperatur (2x)
- Luftdruck
- Accelerometer
- Gyroskop
- Magnetometer



AVR Mikrocontroller

- 8 Bit Mikrocontroller
- 8MHz Takt (max. 16MHz)
- 128kB Flash
- 4kB EEPROM
- 16kB SRAM
- Hersteller: Atmel
- ca. 6€ bei Reichelt
- einer der „größten“ AVRs!





AVR Architektur

- Harvard-Architektur
  - Programm- und Datenspeicher sind getrennt
  - Programm im Flash, Daten im SRAM
- RISC-Befehlssatz
  - die meisten Befehle werden in einem Takt ausgeführt
  - Multiplikation dauert zwei Takte
  - Keine Hardware-Divisions-Einheit!
    - 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
- $0 \leqslant \textit{uint8\_t} \leqslant 255$
- 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 (Ubuntu)

Linux:

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

Raum IZ G40:

- alle Pakete sind installiert

  $\Rightarrow$ nur Repository auschecken

git

- Verteilte Versionsverwaltung
- Contiki wird mittels git entwickelt
- IBR entwickelt Contiki für INGA
  ⇒ Unterstützung der speziellen INGA Hardware

- Anonymer Checkout
  ⇒ *git clone https://github.com/ibr-cm/contiki-inga.git*
  ⇒ alternativ: fork in den eigenen GitHub-Account
- Git-Tutorial:
  `https://guides.github.com/activities/hello-world/`
- Git Cheat Sheet: `https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf`

(1) Instant Contiki installieren

VirtualBox installieren

- `https://www.virtualbox.org/wiki/Downloads`
- Ggfs. Virtualisierung im Bios erlauben

Instant Contiki (.ova) herunterladen

- Link auf der Website zum Praktikum

ova-Datei in VirtualBox importieren

- Doppelklick auf ova-Datei
- importieren

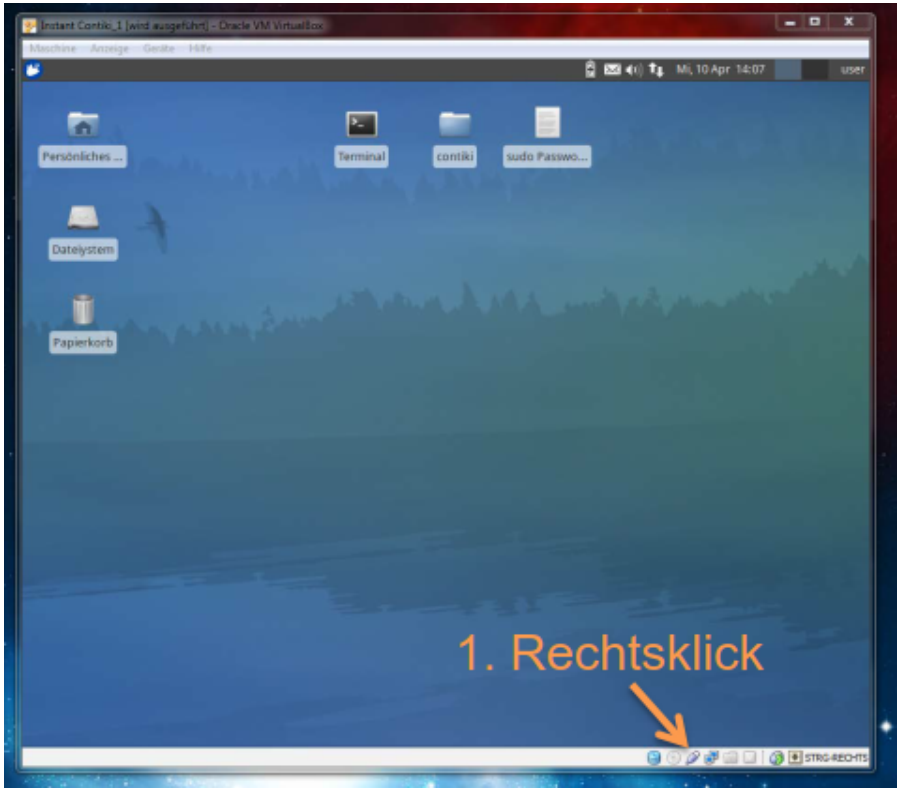
(1) Instant Contiki benutzen

VM starten

- Benutzer: user
- Passwort: contiki

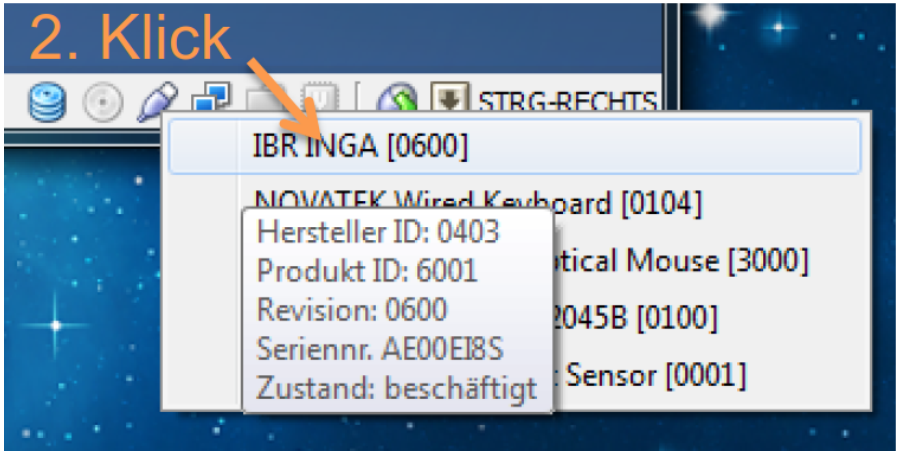
Contiki befindet sich nun auf dem Desktop

(1) INGA an Instant Contiki „anschließen“



Kein USB-Gerät angezeigt? Füge Dich der Gruppe vboxusers hinzu!

- Beende alle VM’s
- Terminal: *sudo usermod -aG vboxusers $USER$*
- Hauptsystem neu starten



(2) Linux selbst einrichten

Pakete für AVR-Toolchain installieren

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

Contiki INGA Repository auschecken

- *git clone https://github.com/ibr-cm/contiki-inga.git*

Rechte setzen

- *sudo cp “pfad_zum_repo“/contiki-inga/example/inga/99-inga-usb.rules /etc/udev/rules.d/99-inga-usb.rules*
- *sudo gedit /etc/udev/rules.d/99-inga-usb.rules* Eventuell Anpassung der Gruppe, falls z.B. nicht beide INGAs ansprechbar sind
- *sudo restart udev*

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-1.0-0 libusb-dev libftdi1 libftdi-dev libpopt0 libpopt-dev libudev1 libudev-dev libssl-dev*

Kompilieren

- *cd "pfad_zum_repo"/tools/inga/inga_tool*
- *make*

INGA Tool

Reset (testweise) durchführen

- *./inga_tool -d /dev/inga/node-x -r*
- „x“ ist die Seriennummer des INGA-Knotens
- Die Seriennummer findet man nach Anschluss standardmäßig unter */dev/inga/*

INGA flashen

Flashen der Datei "project.c"

Target speichern

- *make TARGET=inga savetarget*

Alle angeschlossenen INGAs flashen

- *make project.upload*

oder um bestimmte INGAs zu flashen

- *make project.upload MOTES=/dev/inga/node-ab42*

INGA Ausgaben

Debug-Ausgaben (printf) anzeigen

- *make login*

oder auch nur für bestimmten INGA (Beispiel)

- *make login MOTES=/dev/inga/node-ab42*

makefile

Im Makefile steht, wo Contiki liegt, wie das Projekt heißt und was Compiler und Linker sonst noch so beachten sollen

Inhalt eines Makefiles:

CONTIKI = Pfad_zum_Contiki_Verzeichnis
all: Projekt_Dateiname_ohne_Extension
include $(Contiki)/Makefile.include

Beispiel:

CONTIKI = ../wsnlab/contiki-inga
all: hello-world
include $(CONTIKI)/Makefile.include

makefile - Optionen

Rime verwenden

- *CONTIKI_WITH_RIME=1*

printf mit float-Unterstützung

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

weitere Möglichkeiten in den *examples/*

C-Crashkurs

- imperative Programmiersprache
- erfunden 1972 von Dennis Ritchie

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

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

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

Header Beispiel - helper_functions.h

```
1  #pragma once
2
3  #define MY_CONSTANT 42 //this is an inline comment
4
5  /* this is a block comment
6
7  it spans multiple lines*/
8
9  uint8_t is42(uint8_t check);
10
11 void add42(uint8_t *sum);
```

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;
}

```

Pointer

Pointer sind Speicheradressen und zeigen somit auf ein Byte im Speicher

```
1  uint8_t a;      //normale Variable a
2  uint8_t *b = &a;//b ist ein Pointer auf die Adresse von a
3
4  *b = 5;         //a=5
5
6  uint8_t c[5];   //Array mit 5 Werten, 5 Byte im Speicher
7  a = c[0];       /*c[0] ist nicht initialisiert
8                  -> Unbekannter Wert wird a zugewiesen*/
9
10 a = c[42];      /*Kein Fehler, Zugriff findet statt.
11                 Wert kommt aus Speicher hinter c*/
12
13 c[42] = 0;      /*Der Speicher HINTER c wird veraendert
14                 -> NIEMALS MACHEN!!!*/
15
16 uint8_t *d = &c[2];     //d zeigt auf c[2]
17 d += 1;                 //d zeigt auf c[3]
18
19
```

Datentypen

- (unsigned) char, short, int, long, long long
  > Unübersichtlich, keine Bitanzahl erkennbar

- ***uint8_t, uint16_t, uint32_t***
- ***int8_t, int16_t, int32_t***
- ***float (32 Bit)***

- double ist genauso groß wie float (32 Bit)

- für Benutzung mit *printf* im Makefile aktivieren (siehe Folie 20)

printf Debugging (1/2)

Standardausgabe wird auf serielle Schnittstelle (USB) umgeleitet

$\Rightarrow$ "Debugging" per *printf()* möglich

```
1  #include <stdio.h>
2
3  uint8_t main()
4  {
5    //gibt 2 mit Zeilenumbruch auf Konsole aus
6    printf("%d\n", 0x2);
7
8    //gibt die Zahl 42 hexadezimal aus
9    printf("%x", 42);
10
11   //gibt "Der Wert ist 42" aus (mit Zeilenumbruch)
12   printf("%s␣ist␣%d\n", "Der␣Wert", 42);
13
```

printf Debugging (2/2)

```
1
2
3
4   //erzeuge 6 Zeichen Puffer, initialisiert mit 0
5   uint8_t buffer[6] = {0};
6
7   //schreibe eine 5 als Zeichen in den Puffer
8   sprintf(buffer, "%d", 5);
9
10  //gibt den Inhalt von Buffer als String aus
11  printf("%s\n", buffer);
12
13  //->Strings enden beim ersten Byte, das 0 ist
14 }
```

Mikrocontroller Rechentricks

Ausnutzung von Bitoperatoren:

```
1  uint8_t a = 0b00001111;
2
3  //ein Bit in einem Byte setzen (Shift-Operator <<)
4  a = a | (1<<7);          // a = 0b10001111
5
6  //ein Bit loeschen (Bitweises NICHT ~)
7  a &= ~(1<<1);            // a = 0b10001101
8
9  //die unteren 3 Bit ausschneiden (Bitweises UND &)
10 a &= 0b00000111;         // a = 0b00000101
11
12 //die oberen 5 Bit setzen (Bitweises ODER |)
13 a |= 0b11111000;         // a = 0b11111101 = 253
14
15 //durch 4 teilen (Bitshift um 2 nach unten)
16 a = a >> 2;              // a = 0b00111111 = 253 / 4 = 63
```

Mehr Tricks unter:

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

Contiki

Themen Teil 1

- LEDs
- Timer
- Button
- Sensoren auslesen
- Watchdog
- HowTo: Ein Projekt kompilieren

Contiki

Themen Teil 2

- Prozesse
- Events
- Rime
- Cooja - Netzwerksimulator

LEDs

```
1  #include "leds.h"
2
3  // LED Konstanten (sind bereits in leds.h definiert)
4  #define LEDS_ALL        // beide LEDs
5  #define LEDS_GREEN      // gruene LED
6  #define LEDS_YELLOW     //gelbe (orange) LED
7
8  // LEDs initialisieren (sollte nicht mehr noetig sein)
9  leds_init();    //muss nur einmal aufgerufen werden
10
11 // LEDs ansteuern
12 leds_on(LEDS_ALL);              // alle LEDs einschalten
13 leds_off(LEDS_GREEN);           // gruene LED ausschalten
14 leds_toggle(LEDS_YELLOW);       // gelbe LED umschalten
15 leds_toggle(LEDS_GREEN | LEDS_YELLOW);// beide umschalten
16
17 //mehr Infos in examples/inga/demo/led_demo.c
```

Einfacher Timer

```
1  // Timer, der wartet bis eine Zeit abgelaufen ist
2
3  // Timer definieren
4  static struct timer myTimer;
5
6  // Timer starten (~1/10 Sekunde)
7  timer_set(&myTimer, CLOCK_SECOND / 10);
8
9  // Timer neustarten
10 timer_restart(&myTimer); // Timer neu laden
11 timer_reset(&myTimer); // Neustart vom Ablaufzeitpunkt
12
13 // Pruefen ob Timer abgelaufen ist
14 timer_expired(&myTimer);
```

Event Timer

```
1  // Timer, der der nach Ablauf ein Event sendet
2
3  // Timer definieren
4  static struct etimer myTimer;
5
6  // Timer starten (~1/10 Sekunde)
7  etimer_set(&myTimer, CLOCK_SECOND / 10);
8
9  // Timer neustarten
10 etimer_restart(&myTimer); // Timer neu laden
11 etimer_reset(&myTimer); // Neustart vom Ablaufzeitpunkt
12
13 // Pruefen ob Timer abgelaufen ist
14 etimer_expired(&myTimer);
```

Callback Timer

```
1  // Nach Ablauf des Timers wird
2  // eine Callback-Funktion aufgerufen
3
4  // Timer definieren
5  static struct ctimer myTimer;
6  /* Pointer auf die Funktion die als
7   Callback aufgerufen werden soll */
8  static void (*light_off) = leds_off;
9  // Parameter fuer den Funktionsaufruf
10 static unsigned char led = LEDS_YELLOW;
11 // LED initialisieren und einschalten
12 leds_init();
13 leds_on(LEDS_YELLOW);
14 // Timer starten
15 ctimer_set(&myTimer, CLOCK_SECOND*2, light_off, &led);
```

Button

```
1  #include "dev/button-sensor.h";
2
3  // Button aktivieren
4  SENSORS_ACTIVATE(button_sensor);
5
6  // Auf Button warten
7  // Prozess gibt Kontrolle ab und wartet auf Button Event
8  PROCESS_WAIT_EVENT_UNTIL(ev == sensors_events
9                           && data == &button_sensor);
10
11 // Event wird bei druecken und loslassen generiert!
12
13 // ev und data sind Parameter des Prozesses
14 // mehr Infos in examples/inga/demo/button_demo.c
```

Luftdruck- & Temperatursensor

```
1  #include "pressure-sensor.h"
2
3  // Sensor aktivieren
4  SENSORS_ACTIVATE(pressure_sensor);
5
6  /*
7   Werte auslesen
8  */
9  // Druck
10 uint32_t pressure = (pressure_sensor.value(PRESS_H)<<16) |
       pressure_sensor.value(PRESS_L);
11 // Temperatur
12 int16_t temperature = pressure_sensor.value(TEMP);
13
14 // hilfreich ist ein Blick in platform/inga/dev
```

Accelerometer (Beschleunigungssensor)

```
1  #include "acc-sensor.h"
2
3  // Sensor aktivieren
4  SENSORS_ACTIVATE(acc_sensor);
5
6  // Werte auslesen
7  int16_t x = acc_sensor.value(ACC_X); // x-Achse
8  int16_t y = acc_sensor.value(ACC_Y); // y-Achse
9  int16_t z = acc_sensor.value(ACC_Z); // z-Achse
10
11 // Siehe platform/inga/dev fuer mehr Informationen
```

Gyroskop (Winkelgeschwindigkeit)

```
1 #include "gyro-sensor.h"
2
3 // Sensor aktivieren
4 SENSORS_ACTIVATE(gyro_sensor);
5
6 // Werte auslesen (AS = Angular Speed)
7 int16_t x = gyro_sensor.value(GYRO_X); // x-Achse
8 int16_t y = gyro_sensor.value(GYRO_Y); // y-Achse
9 int16_t z = gyro_sensor.value(GYRO_Z); // z-Achse
```

Watchdog

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

Hello World vom INGA

```
1  // definiere einen neuen Prozess mit den Namen "Hello World process"
2  PROCESS(hello_world_process, "Hello World process");
3
4  // starte den Prozess bei Systemstart
5  AUTOSTART_PROCESSES(&hello_world_process);
6
7  // Implementation des Prozesses
8  PROCESS_THREAD(hello_world_process, ev, data)
9  {
10   PROCESS_BEGIN(); // Start-Makro
11
12   printf("Hello World!\n");
13   while(1) // Endlosschleife
14   {
15     // Gebe die Kontrolle ab (verhindert ablaufen des Watchdogs)
16     PROCESS_PAUSE();
17   }
18
19   PROCESS_END(); // End-Makro
20
21 }
```

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), 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 ein *etimer* abläuft
- Scheduling ist NICHT preemptiv
  - Prozesse laufen bis sie freiwillig die Kontrolle wieder abgeben
  - oder der Watchdog zuschlägt

Contiki - 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*

```
1 // Prozess starten
2 process_start (struct process *p, const char *arg)
3
4 // Prozess beenden
5 process_exit (struct process *p)
6
```

Events - Senden

```
1  // Globale Eventnummer holen
2  process_event_t myEvent;
3  myEvent = process_alloc_event();
4
5  // asynchrones Event an Prozess senden
6  // (wird beim naechsten Aufruf des Prozesses bearbeitet)
7  process_post(&myProcess, myEvent, NULL);
8  // an Stelle von NULL beliebiger Pointer moeglich
9
10 // synchrones Event senden (wird sofort bearbeitet)
11 process_post_synch(&myProcess, myEvent, NULL);
```

Events - Empfangen

```
1 // Auf ein Event warten
2 PROCESS_WAIT_EVENT();
3 PROCESS_YIELD();
4
5 // Auf ein Ereignis unter einer Bedingung warten
6 PROCESS_WAIT_EVENT_UNTIL(condition c); // (z.B. auf Timer)
7
8 // Prozess kurz unterbrechen (moeglichst schnell weiter)
9 PROCESS_PAUSE();
```

Zwei Prozesse - Beispiel

```
1  static process_event_t event_data_ready;
2
3  PROCESS(temp_process, "Temperature process");
4  PROCESS(print_process, "Print process");
5
6  AUTOSTART_PROCESSES(&temp_process, &print_process);
7
8  PROCESS_THREAD(temp_process, ev, data) {
9     event_data_ready = process_alloc_event();
10    while(1) {
11       process_post(&print_process, event_data_ready, &valid_measure);
12    }
13 }
14
15 PROCESS_THREAD(print_process, ev, data) {
16    while(1) {
17       PROCESS_WAIT_EVENT_UNTIL(ev == event_data_ready);
18       ...
19    }
20 }
```

Kommunikation - Stacks

Zwei Stacks in Contiki verfügbar

- IPv6 (TCP/IP)
- Rime (leichtgewichtig)

IPv6

Makefile

- IPv6 standardmäßig aktiviert

UDP

- *udp_new()* und *udp_bind()*
- *udp_event* bei ankommenden Daten, etc.
- zum Senden: uip_udp_packet_send()

TCP

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

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

  $\Rightarrow$ 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

  $\Rightarrow$ 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

```
1  // Anonymous best-effort single-hop broadcast
2  // Called when a message is received
3  void recv(struct abc_conn *c) {
4     printf("Message␣received\n");
5  }
6
7  struct abc_callbacks cb = {recv};        // Callback
8  struct abc_conn c;                       // Connection
9
10 void setup_sending_a_message_to_all_neighbors(void) {
11    abc_open(&c, 128, &cb);               // Channel 128
12 }
13
14 void send_message_to_neighbors(char *msg, int len) {
15    rimebuf_copyfrom(msg, len);           // Setup rimebuf
16    abc_send(&c);                         // Send message
17 }
```

Rime – Multi-Hop Beispiel

```
1  // Reliable multi-hop flooding
2  // Called when a message is received
3  void recv(struct trickle_conn *c) {
4     printf("Message␣received\n");
5  }
6
7  struct trickle_callbacks cb = {recv};          // Callback
8  struct trickle_conn c;                         // Connection
9
10 void setup_sending_a_message_to_network(void) {
11    trickle_open(&c, 128, &cb);                 // Channel 128
12 }
13
14 void send_message_to_neighbors(char *msg, int len) {
15    rimebuf_copyfrom(msg, len);                 // Setup rimebuf
16    trickle_send(&c);                           // Send message
17 }
```

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!)

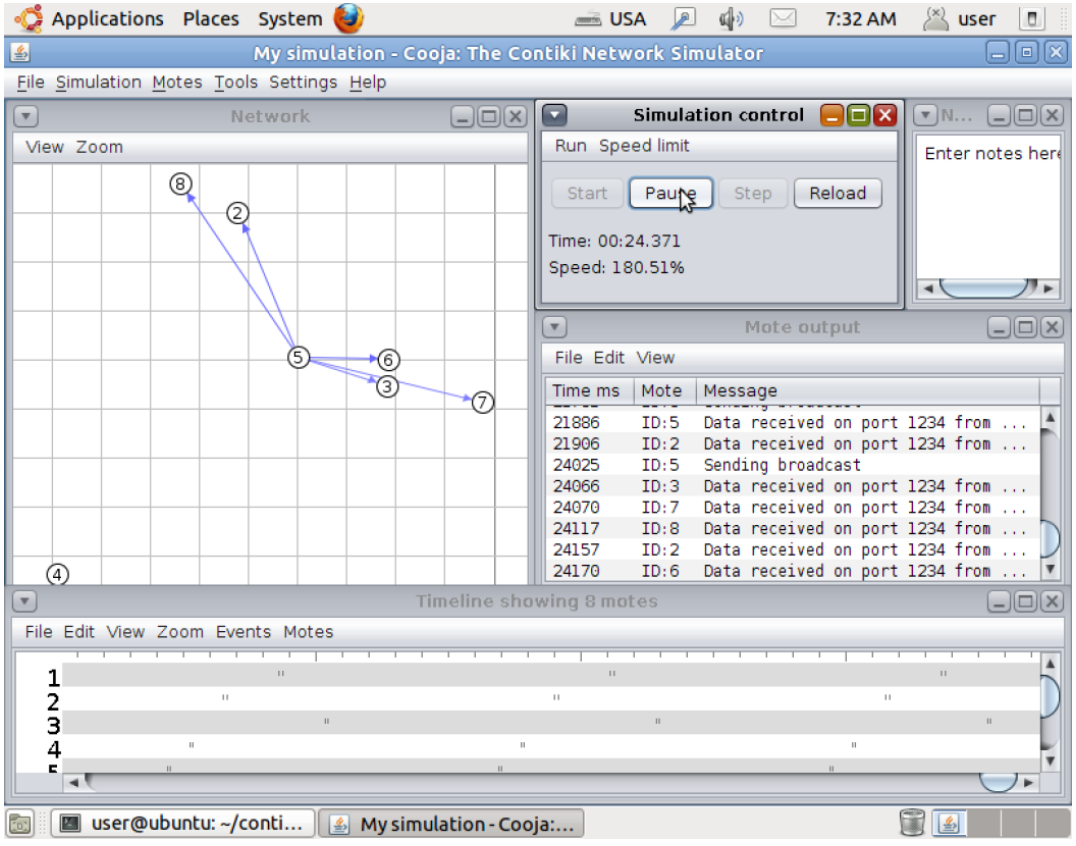
Node ID neu setzen oder Channel ändern

- cd examples/inga/nodesetup/
- make NODE_ID=<id> [RADIO_CHANNEL=<value>] setup

  $\Rightarrow$ siehe examples/inga/nodesetup/README.md

Cooja

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



Und nun?

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

Wichtige Links

Praktikumsseite

- https://www.ibr.cs.tu-bs.de/courses/ss18/wsn/index.html

GitHub Projekt

- https://github.com/ibr-cm/contiki-inga

Offizielle Contiki Homepage

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

AVR-GCC-Tutorial von mikrocontroller.net

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

Fragen & Probleme

Hiwis

- Daniel Heidorn (d.heidorn@tu-bs.de)
- Lars Giesmann (l.giesmann@tu-bs.de)

Sprechstunde

- nach Vereinbarung, MikLab (IZ 147)

Acknowledgements

Präsentation basiert auf

- Folien von Thiemo Voigt vom Swedish Institute of Computer Science
- Contiki Einführung von Adam Dunkels
- Karsten Hinz
- Robert Hartung
- Yannic Schröder

Technische Universität Braunschweig

Viel Spaß beim Basteln!