Arbeitskreis Hardware

Prof. Dr. Michael Rohs, Dipl.-Inform. Sven Kratz
michael.rohs@ifi.lmu.de
MHCI Lab, LMU München
Organization

• **Objective:** Learn about embedded interactive systems
  – Just for fun, **no ECTS credits**!

• **Date:** Mondays 18-20+
  – 18-19 presentation and discussion of new topic
  – 19-20+ work on topic / project

• **Schedule overview**
  – 11 sessions
  – No class May 9th (CHI) and June 13th (Pfingsten)

• **Hardware components provided**
  – Buy AVR programmer (15 EUR) and power supply (7 EUR)
## Schedule (preliminary)

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic (preliminary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5.</td>
<td>Introduction to embedded interaction, microcontrollers, hardware &amp; software tools</td>
</tr>
<tr>
<td>9.5.</td>
<td><em>keine Veranstaltung (CHI)</em></td>
</tr>
<tr>
<td>16.5.</td>
<td>AVR architecture, AVR assembler, LED multiplexing/charlieplexing</td>
</tr>
<tr>
<td>23.5.</td>
<td>Sensors: light, force, temperature, humidity, capacity, inductivity, distance, acceleration</td>
</tr>
<tr>
<td>30.5.</td>
<td>Electronics basics, soldering, PCB design &amp; fabrication, EAGLE, 3D printing</td>
</tr>
<tr>
<td>6.6.</td>
<td>Displays (character LCDs, graphics LCDs), audio (speakers, amplification, op-amps)</td>
</tr>
<tr>
<td>13.6.</td>
<td><em>keine Veranstaltung (Pfingsten)</em></td>
</tr>
<tr>
<td>20.6.</td>
<td>I2C: interfacing to other chips (EEPROM, real-time clock, digital sensors)</td>
</tr>
<tr>
<td>27.6.</td>
<td>Actuation: stepper motors, servo motors</td>
</tr>
<tr>
<td>4.7.</td>
<td>Communication: fixed-frequency RF, ZigBee, Bluetooth</td>
</tr>
<tr>
<td>11.7.</td>
<td>Project</td>
</tr>
<tr>
<td>18.7.</td>
<td>Project</td>
</tr>
<tr>
<td>25.7.</td>
<td>Project</td>
</tr>
</tbody>
</table>
Technologies and Tools

Milling, drilling, cutting PCB: Roland Modela

ATtiny, Atmega microcontroller

PCB Design: EAGLE

Printing casings: RepRap 3D printer

www.rolanddg.com/product/3d/3d/mdx-20_15/mdx-20_15.html

en.wikipedia.org/wiki/RepRap
www.reprap.org/wiki/Mendel
Embedded Systems

• Computer systems with dedicated functionality
  – Cf. general-purpose computer (PC)
  – Microcontrollers, digital signal processors, sensors, actuators

• Often not perceived as a “computer”
  – Users may not know that a computer system is inside

• Examples
  – Wrist watches, mp3 players, digital cameras, GPS receivers, bike computers, heart rate monitors, cars (motor, ABS, ESP), traffic lights, microwave ovens, dishwashers, washing machines, door openers, weather stations, TV sets, remote controls, DVD players, factory automation systems, telephone switches, networked thermostats, implantable medical devices, toys
Technological Enablers

• Processing & storage
  – Cheap, fast, reliable, small, large capacity, energy efficient
  – Moore’s Law

• Networking
  – Cheap, fast, reliable, global, local, wireless, ad-hoc, low power

• Displays
  – Cheap, small, high quality, energy efficient, integrated

• Sensors & actuators
  – Cheap, small, accurate, invisible, many types
Computing Paradigms

“Ubiquitous computing names the third wave in computing, just now beginning. First were mainframes, each shared by lots of people. Now we are in the personal computing era, person and machine staring uneasily at each other across the desktop. Next comes ubiquitous computing, or the age of calm technology, when technology recedes into the background of our lives.” Mark Weiser
Vision of Ubiquitous Computing

• “The most profound technologies are those that disappear. They weave themselves into the fabric of every day life, until they are indistinguishable from it.” (Mark Weiser)

• Vision
  – Computers embedded in everyday things
  – Seamless integration into our environment
  – All components are connected and exchange information

• Ubiquitous computing vs. virtual environments
  – Computers in the world, instead of world in the computer

• Calm Technology
  – Technology moves into the background
Embedded & Tangible Interaction

• Challenges for human-computer interaction
  – How to interact with so many systems?
  – How to keep users from constant interruptions and distractions?
  – Device interaction happens in an everyday situation. How to take that into account?
  – What are novel forms of interaction?
  – Design opportunities?

• Interaction themes
  – Natural interfaces
  – Context-aware applications
  – Automatic capture and access
  – Continuous interaction
Microcontrollers

• Integrates processor, memory, I/O peripherals, and sensors on a single chip
  – Replaces many traditional hardware components in a single chip
  – Lower cost, fewer additional components, smaller circuit board
  – Very memory efficient (sleep modes)
  – Software flexibility through software

• Memory types
  – Flash: program
  – RAM: working memory (stack, heap)
  – EEPROM: non-volatile memory

• Interrupt-driven I/O
  – Sources: signal changes, timer overflow, ADC conversion done
  – Interrupts can wake microcontroller from low-power sleep state
Microcontrollers

Source: Gadre, Malhotra: tinyAVR projects
Microcontrollers

• I/O Pins
  – Used as input or output (controlled by software)
  – Serial communications (UART, I²C, SPI)
  – Signal generation (PWM, timers)
  – Analog input (ADC conversion)

• Development
  – In-circuit programming and debugging, field update of firmware
  – Programming in assembly language or C

• Selectable clock frequencies
  – Lower clock rate → less energy

• No floating point unit (typically)
Atmel AVR: ATtiny, ATmega

• 8-bit RISC chip, Harvard architecture

• ATtiny
  1–8 kB program memory
  6–32-pin package

• ATmega
  4–256 kB program memory
  28–100-pin package
  Extended instruction set
  • Multiply instructions
  • Handling larger program memories

• Large family of devices, specific features
Many types of AVRs: Choose depending on required features

**ATtiny13**
- 6 I/O pins, 1.8-5.5V operation
- 20 MPIS @ 20 MHz (clock rate selectable), internal oscillator
- 64B RAM, 64B EEPROM, 1kB Flash program memory
- 8-bit timer, 2 PWM channels, 10-bit ADC, analog comparator
- Price: €1.15

**ATtiny45**
- 6 I/O pins, 1.8-5.5V operation
- 20 MPIS @ 20 MHz (clock rate selectable), internal oscillator
- 256B RAM, 256B EEPROM, 4kB Flash program memory
- 2 8-bit timers, 4 PWM channels, 10-bit ADC, analog comparator, SPI, TWI, temperature sensor
- Price: €2.05
Many types of AVRs: Choose depending on required features

**ATmega8**
- 23 I/O pins, 2.7-5.5V operation
- 16 MIPS @ 16 MHz (clock rate selectable), internal oscillator
- 1kB RAM, 512B EEPROM, 8kB Flash program memory
- 2 8-bit timers, 1 16-bit timer, 3 PWM channels, 10-bit ADC, analog cmp., SPI, TWI, USART
- Price: €2.60

**ATmega328P**
- 23 I/O pins, 1.8-5.5V operation
- 20 MIPS @ 20 MHz (clock rate selectable), internal oscillator
- 2kB RAM, 1kB EEPROM, 4kB Flash program memory
- 2 8-bit timers, 1 16-bit timer, 6 PWM channels, 10-bit ADC, analog cmp., SPI, TWI, USART, temperature sensor
- Price: €3.30
Pinout ATtiny13

- Multiplexed pin functions, software configurable
  - Example: Flash/EEPROM programming via SPI:
    - MOSI = master out, slave in (from programmer to ATtiny)
    - MISO = master in, slave out (from ATtiny to programmer)
    - SCK = serial clock
  - Example: ADC1 = ADC input channel 1
  - Example: PCINT3 = pin change interrupt 3

DIP = DIL = Dual In-line Package
SOIC = Small-Outline Integrated Circuit

Source: Atmel data sheet
Pinout ATmega8

(RESET) PC6 □ 1
(RXD) PD0 □ 2
(TXD) PD1 □ 3
(INT0) PD2 □ 4
(INT1) PD3 □ 5
(XCK/T0) PD4 □ 6
VCC □ 7
GND □ 8
(XTAL1/TOSC1) PB6 □ 9
(XTAL2/TOSC2) PB7 □ 10
(T1) PD5 □ 11
(AIN0) PD6 □ 12
(AIN1) PD7 □ 13
(ICP1) PB0 □ 14
28 □ PC5 (ADC5/SCL)
27 □ PC4 (ADC4/SDA)
26 □ PC3 (ADC3)
25 □ PC2 (ADC2)
24 □ PC1 (ADC1)
23 □ PC0 (ADC0)
22 □ GND
21 □ AREF
20 □ AVCC
19 □ PB5 (SCK)
18 □ PB4 (MISO)
17 □ PB3 (MOSI/OC2)
16 □ PB2 (SS/OC1B)
15 □ PB1 (OC1A)
AVR Memory Layout

Program Memory

internal Flash
word size = 16 bits

Data Memory

32 registers
64 I/O registers

internal SRAM
word size = 16 bits

Data Storage

internal EEPROM
word size = 8 bits
AVR I/O Ports

• I/O pin either input or output
  – Individually software-controlled

• Pin as output
  – States: low, high
  – Can drive 40mA (→ LED)

• Pin as input
  – Internal pull-up resistor (enabled/disabled in software)
  – high resistance state (high-Z) if pull-up disabled
Accessing the I/O Ports

• Three memory addresses for each I/O port
  – Data Direction Register: DDRx
    • 1 = output
    • 0 = input
  – Data Register: PORTx
    • if input: 1 = pull-up enabled, 0 = pull-up disabled
    • if output: 1 = PIN driven high, 0 = PIN driven low
  – Port Input Pins: PINx
    • read: PIN state (independent of DDRx)
    • write 1: toggles PORTx

![Diagram of I/O Port Access](image)
AVR I/O Ports: Pin Control Example

<table>
<thead>
<tr>
<th>PIN</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>in/out</td>
<td>out</td>
<td>out</td>
<td>out</td>
<td>out</td>
<td>in</td>
<td>in</td>
<td>in</td>
<td>in</td>
</tr>
<tr>
<td>value</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>pullup</td>
<td>hi-z</td>
<td>hi-z</td>
<td>hi-z</td>
</tr>
</tbody>
</table>

Assembly

```assembly
ldi r16, (1<<PB4) | (1<<PB1) | (1<<PB0)
ldi r17, (1<<DDB3) | (1<<DDB2) |
| (1<<DDB1) | (1<<DDB0)
out PORTB,r16
out DDRB,r17
nop    // synchronization
in r16,PINB
```

C

```c
unsigned char i;
PORTB = (1<<PB4) | (1<<PB1) | (1<<PB0);
DDRB = (1<<DDB3) | (1<<DDB2) |
| (1<<DDB1) | (1<<DDB0);
__no_operation(); // synchronization
i = PINB;
```
“μC Hello World”: Blinking an LED

```c
#define F_CPU 1200000
#include <avr/io.h>
#include <util/delay.h>

int main()
{
    DDRB = 0b010000;
    while (1) {
        PORTB = 0b010000;
        _delay_ms(500);
        PORTB = 0b000000;
        _delay_ms(500);
    }
    return 0;
}
```
Downloading the Program to the μC

- Serial programming via Serial Peripheral Interface (SPI)
  - MISO, MOSI, SCK
Memory Programming

• Tasks
  – Download/upload program code to/from Flash memory
  – Download/upload data to/from internal EEPROM
  – Configuring the microcontroller (“fuse bits”)

• Programming options
  – Serial programming
    • In-system programming (ISP)
    • High-voltage serial programming (HVSP, only 8-pin controllers)
  – High-voltage parallel programming
    • If RESET pin used as I/O pin: high-voltage programming
  – debugWire on-chip debug system
    • Uses RESET pin for debugging and Flash/EEPROM programming
How to set the Fuses?

- AVRFuses tool
  - [http://www.vonnieda.org/software/avrfuses](http://www.vonnieda.org/software/avrfuses)

- Online fuse calculator
  - [http://www.engbedded.com/fusecalc](http://www.engbedded.com/fusecalc)

- ATtiny13 datasheet, 17.2 Fuse Bytes
  - ATtiny13 has two fuse bytes
  - Default: high byte = 0b11111111, low byte = 0b01101010

<table>
<thead>
<tr>
<th>Fuse Bit</th>
<th>Bit No</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>7</td>
<td>-</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>-</td>
<td>6</td>
<td>-</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>-</td>
<td>5</td>
<td>-</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>SELFPRGEN(^1)</td>
<td>4</td>
<td>Self Programming Enable</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>DWEN(^2)</td>
<td>3</td>
<td>debugWire Enable</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>BODEVELLO(^3)</td>
<td>2</td>
<td>Brown-out Detector trigger level</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>BODEVEL1(^3)</td>
<td>1</td>
<td>Brown-out Detector trigger level</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>RSTDISBL(^4)</td>
<td>0</td>
<td>External Reset disable</td>
<td>1 (unprogrammed)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fuse Bit</th>
<th>Bit No</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>7</td>
<td>-</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>-</td>
<td>6</td>
<td>-</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>-</td>
<td>5</td>
<td>-</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>SPIEN(^1)</td>
<td>7</td>
<td>Enable Serial Programming and Data Downloading</td>
<td>0 (programmed) (SPI prog. enabled)</td>
</tr>
<tr>
<td>EESAVE</td>
<td>6</td>
<td>Preserve EEPROM memory through Chip Erase</td>
<td>1 (unprogrammed) (memory not preserved)</td>
</tr>
<tr>
<td>WDTON(^2)</td>
<td>5</td>
<td>Watchdog Timer always on</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>CKDIV8(^3)</td>
<td>4</td>
<td>Divide clock by 8</td>
<td>0 (programmed)</td>
</tr>
<tr>
<td>SUT1(^4)</td>
<td>3</td>
<td>Select start-up time</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>SUTO(^4)</td>
<td>2</td>
<td>Select start-up time</td>
<td>0 (programmed)</td>
</tr>
<tr>
<td>CKSEL1(^5)</td>
<td>1</td>
<td>Select Clock source</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>CKSELO(^5)</td>
<td>0</td>
<td>Select Clock source</td>
<td>0 (programmed)</td>
</tr>
</tbody>
</table>
AVR Configuration via “Fuse Bits”

Caution: Wrong fuse bit settings may render chip unusable!

Tool: AVRFuses (www.vonnieda.org/AVRFuses/)
Configuring AVRFuses for the Programmer and USB Port

mySmartUSB light:

USBasp:

/dev/cu.SLAB_USBtoUART

http://shop.myavr.ch/index.php?sp=article.sp.php&artID=200006

http://www.fischl.de/usbasp/
USB Drivers for “mySmartUSB light”

- USB chip CP2102 from Silicon Laboratories
- Windows
  
  http://shop.myavr.ch/index.php?sp=article.sp.php&artID=200006
- Mac OS X, Linux
  
AVR ISP Connector

- Image of small PCB with one row of connectors
- http://itp.nyu.edu/physcomp/Tutorials/AVRCPeopleProgramming-Programmer

2 VCC
4 MOSI
6 GND

not recommended:

better solution: solder small PCB with 6x1 pins
• Fuses show factory configuration of ATtiny13

• Brown-out detection
  – reset when Vcc below level

• Reset disabled
  – use reset pin as I/O pin: dangerous!

• Start-up time
  – delay until conditions are stable
AVR Clock Options

• Clock frequency can be chosen
  – Application requirements, power consumption
  – Clock prescaler register (divide clock by factor)
  – Component clocks can be disabled to reduce power consumption

• Clock source can be chosen
  – Internal resistor capacitor (RC) oscillator
    • Convenient, but not precise (temperature, operating voltage)
    • ATtiny13: 4.8MHz, 9.6MHz (at 3V and 25°C), 128kHz (low power)
  – External crystal oscillator
    • Highly precise, requires external quartz

• Clock source distributed to modules
  – $CLK_{CPU}$, $CLK_{I/O}$, $CLK_{flash}$, $CLK_{ADC}$
  – $CLK_{ADC}$ allows switching off other clocks during ADC conversion
AVR Development Toolchain & IDEs

- Free AVR toolchain
  - GNU C compiler: avr-gcc (gcc.gnu.org)
  - C library: avr-libc
  - Down-/Uploader: avrdude (www.nongnu.org/avr-libc/)

- CrossPack for Mac OS X
  - avr-gcc on Mac OS X, Xcode can be used (but not required)
  - oder: “sudo port install avr-gcc” (mit MacPorts)

- WinAVR for Windows
  - IDE for avr-gcc on Windows

- Atmel AVR Studio
  - http://www.atmel.com
AVR-GCC Toolchain Overview

- User’s input files
- GCC
- GNU Binutils
- AVR Libc
- GDB / AVaRICE / Simulavr
- AVRDUDE

AVR Libc

• AVR Libc Home Page
  – http://www.nongnu.org/avr-libc/

• up to date?
CrossPack: Creating a Project (Mac OS X)

bash$ avr-gcc-select 3
Current default compiler: gcc 3
bash$ avr-project BlinkLED
Using template: /usr/local/CrossPack-AVR-20100115/etc/templates/TemplateProject
bash$ cd BlinkLED/
bash$ ls -l
total 0
drwxr-xr-x  4 michaelrohs  staff  136 Apr  2 22:44 BlinkLED.xcodeproj
drwxr-xr-x  4 michaelrohs  staff  136 Apr  2 22:44 firmware
bash$ cd firmware/
bash$ ls -l
total 24
-rw-r--r--  1 michaelrohs  staff  4139 Apr  2 22:44 Makefile
-rw-r--r--  1 michaelrohs  staff   348 Apr  2 22:44 main.c
Generated Project in XCode

```c
#include <avr/io.h>

int main(void)
{
    /* insert your hardware initialization here */
    for(;;){
        /* insert your main loop code here */
    }
    /* insert your main loop code here */
    return 0;    /* never reached */
}
```
• Adapt Makefile as required
  – DEVICE, CLOCK, FUSES
  – PROGRAMMER
  – OBJECTS

```bash
DEVICE     = atmega8
CLOCK      = 8000000
PROGRAMMER = -c stk500v2 -P avrdude
OBJECTS    = main.o
FUSES       = -U hfuse:w:0xd9:m -U lfuse:w:0x24:m
```

```
DEVICE     = attiny13
CLOCK      = 9600000
PROGRAMMER = -c USBasp
OBJECTS    = main.o
FUSES       = -U lfuse:w:0x7a:m -U hfuse:w:0xff:m
```
Building within XCode
Flashing AVR from within XCode

- Duplicate existing “firmware” target
- Rename to “install”
- Change Info | Arguments to “flash”

→ store custom template in
  ~/.CrossPack-AVR/templates/TemplateProject
avrdude: AVDevice initialized and ready to accept instructions

Reading | #FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF | 100% 0.01s
avrdude: Device signature = 0x1e9007
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
       To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "main.hex"
avrdude: writing flash (132 bytes):

Writing | #FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF | 100% 1.13s
avrdude: 132 bytes of flash written
avrdude: verifying flash memory against main.hex:
avrdude: load data flash data from input file main.hex:
avrdude: input file main.hex contains 132 bytes
avrdude: reading on-chip flash data:

Reading | #FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF | 100% 0.71s
avrdude: verifying ...
avrdude: 132 bytes of flash verified
avrdude done. Thank you.

Build succeeded 3.4.2011 0:29
No issues
USBasp programmer
http://www.fischl.de/usbasp/
with selectable SCK rate and option to power circuit

breadboard with ATtiny13, LED and 1kOhm resistor
Breadboard

• Quick prototyping
  – Changing/adding components is easy

• Can get confusing soon ("spaghetti wires")
Using the command line (not Xcode)

```bash
bash$ ls
Makefile main.c
bash$ make
avr-gcc -Wall -Os -DF_CPU=9600000 -mmcu=attiny13 -c main.c -o main.o
avr-gcc -Wall -Os -DF_CPU=9600000 -mmcu=attiny13 -o main.elf main.o
rm -f main.hex
avr-objcopy -j .text -j .data -O ihex main.elf main.hex
bash$ make flash
avrdude -c USBasp -p attiny13 -U flash:w:main.hex:i
```

avrdude: AVR device initialized and ready to accept instructions

```
... 
avrdude: writing flash (132 bytes):
Writing | ################################# | 100% 1.13s 
... 
avrdude: 132 bytes of flash verified
```

`with mySmartUSB:`

```bash
avrdude -p attiny13 -c stk500v2
-P /dev/cu.SLAB_USBtoUART
-U flash:w:main.hex:i
```

avrdude: safemode: Fuses OK
avrdude done. Thank you.
Assembly Language

• ATtiny have relatively simple instruction sets and are reasonably simple to program
  – ATtiny13: 120 instructions

• http://avra.sourceforge.net/index.html

• make

• V-USB
Development Process

Great idea → Research → Develop idea, create todo list → Get hardware components

Software development → Testing

Hardware development → PCB design → Testing, fabrication

Testing, fabrication

Source: Gadre, Malhotra: tinyAVR projects
Reading Data Sheets

• Extremely important to read carefully
  – Easy to find online

• Example: 7805 +5V voltage regulator
  – Operate according to “electrical characteristics”

4 Electrical characteristics

Table 3. Electrical characteristics of L7805 (refer to the test circuits, \( T_J = -55 \) to \( 150^\circ\text{C} \), \( V_I = 10\text{V} \), \( I_O = 500\text{mA} \), \( C_I = 0.33\mu\text{F} \), \( C_O = 0.1\mu\text{F} \) unless otherwise specified)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Test conditions</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V_O )</td>
<td>Output voltage</td>
<td>( T_J = 25^\circ\text{C} )</td>
<td>4.8</td>
<td>5</td>
<td>5.2</td>
<td>V</td>
</tr>
<tr>
<td>( V_O )</td>
<td>Output voltage</td>
<td>( I_O = 5\text{mA} ) to ( 1\text{A} ), ( P_O \leq 15\text{W} ) ( V_I = 8 ) to ( 20\text{V} )</td>
<td>4.65</td>
<td>5</td>
<td>5.35</td>
<td>V</td>
</tr>
</tbody>
</table>

– “Application Circuits” show typical usage
ATtiny13 Data Sheet

- 176 pages! (22 pages per pin!)
  - for next time:
    have a look at the data sheet
Hands-On

- Install AVR GCC
- Create stable 5V power supply on breadboard
- Program “µC Hello World” (blinking an LED) onto a ATtiny13
- Store your components into a sealed bag
- For next time: have a look into ATtiny13 datasheet