

Arduino Workshop

University of Moratuwa, Sri Lanka

Dr. Andreas Könsgen
University of Bremen
Bremen, Germany
ajk@comnets.uni-bremen.de

What is Arduino?

- “Arduino is an open-source prototyping platform based on easy-to-use hardware and software.”
- “Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments.”
- “Arduino was born at the *Ivrea Interaction Design Institute* as an easy tool for fast prototyping, aimed at students without a background in electronics and programming.”

[www.arduino.cc/en/Guide/Introduction]

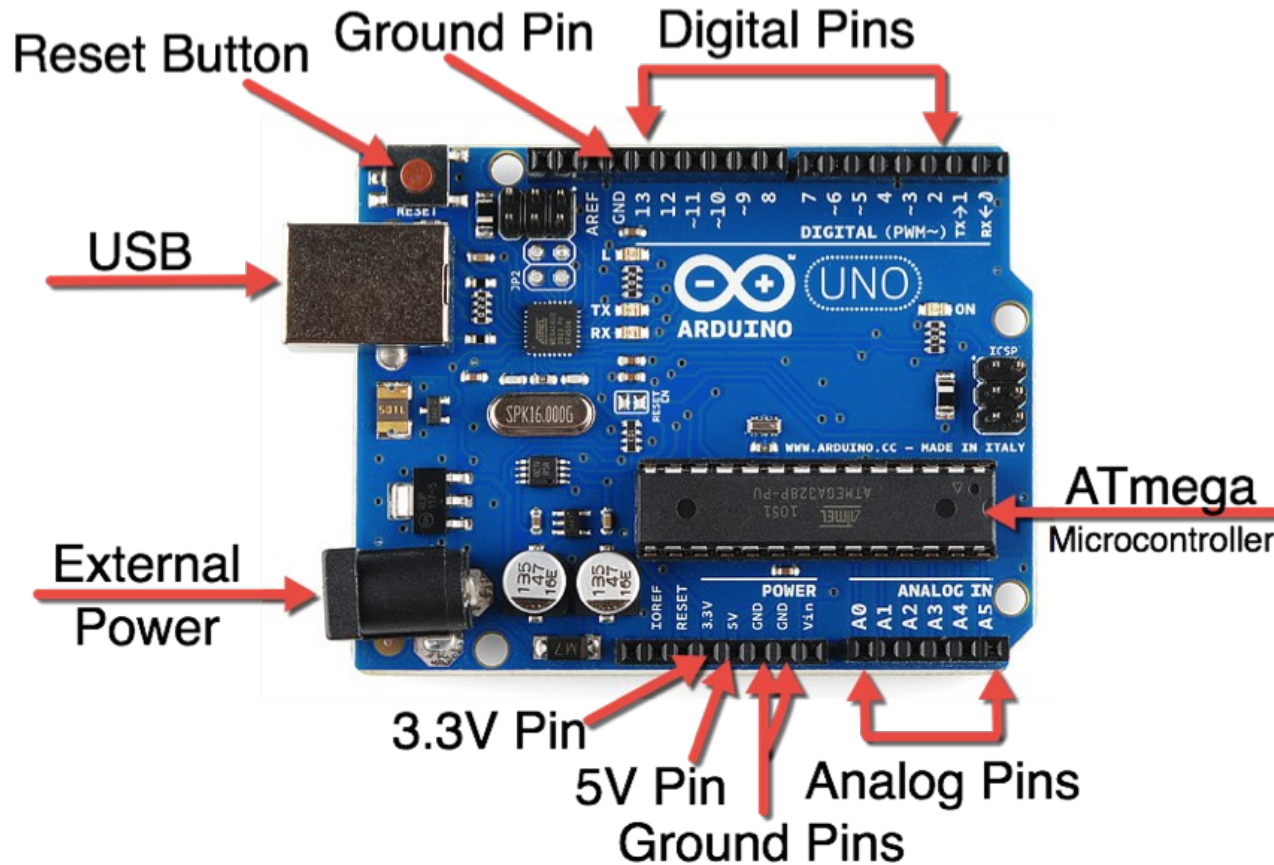
What is Arduino?



The Arduino Team

[makezine.com]

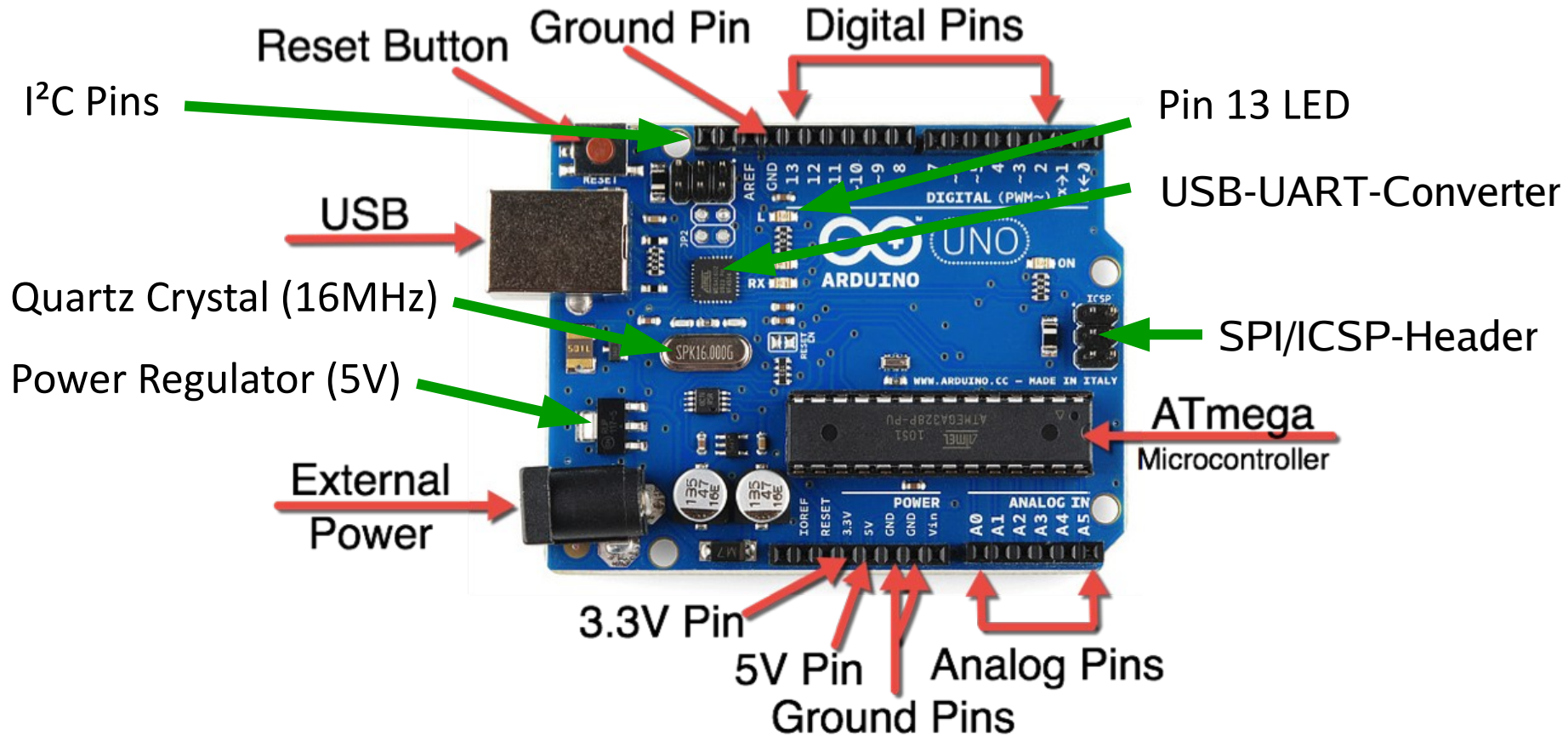
What is Arduino? One Example!



Arduino UNO

[www.ardusat.com]

What is Arduino? One Example!



Arduino UNO

[www.ardusat.com]

What is Arduino? Just a Name, ...

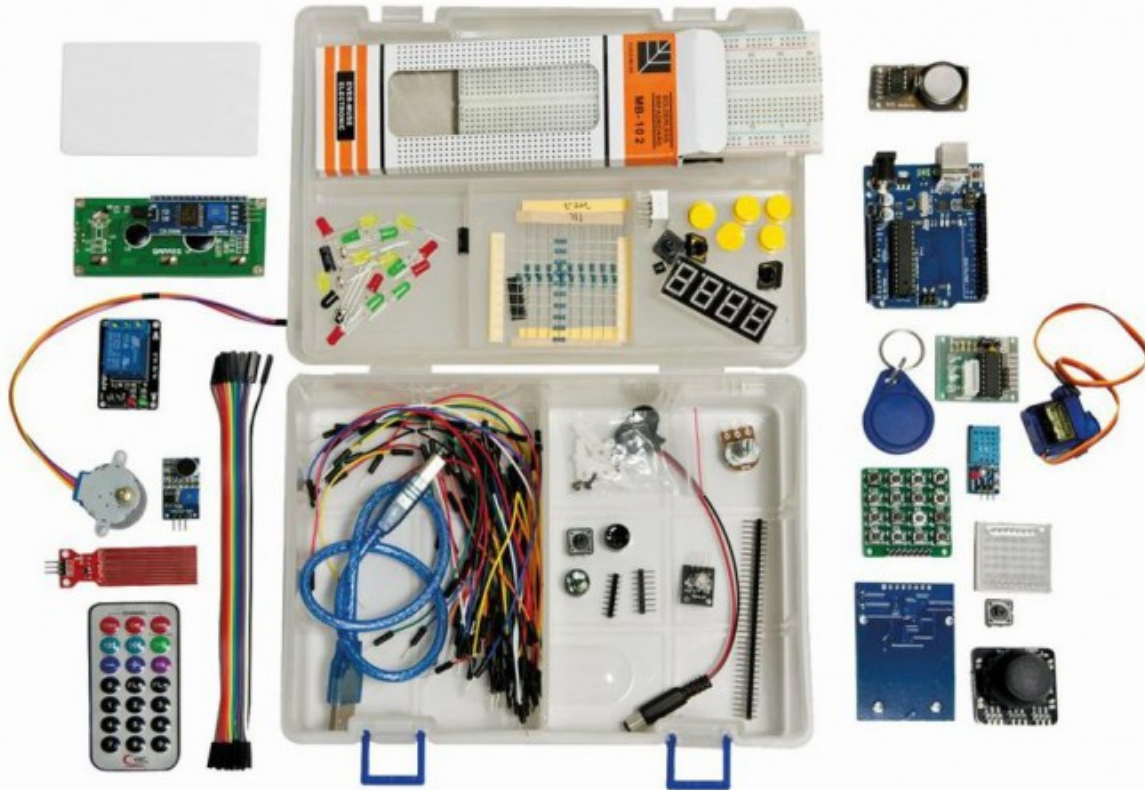
- Arduino means open-source software and hardware
- Arduino brand, logo, design of the website and of the boards are copyrighted (arduino.cc, arduino.org)
- Various boards are available

ENTRY LEVEL	ARDUINO UNO	ARDUINO 101	ARDUINO PRO	ARDUINO PRO MINI	ARDUINO MICRO	
	ARDUINO NANO	ARDUINO STARTER KIT	ARDUINO BASIC KIT	ARDUINO MOTOR SHIELD		
ENHANCED FEATURES	ARDUINO MEGA	ARDUINO ZERO	ARDUINO DUE	ARDUINO PROTO SHIELD		
INTERNET OF THINGS	ARDUINO YUN	ARDUINO ETHERNET SHIELD	ARDUINO GSM SHIELD	ARDUINO WIFI SHIELD 101		
WEARABLE	ARDUINO GEMMA	LILYPAD ARDUINO USB	LILYPAD ARDUINO MAIN BOARD			
	LILYPAD ARDUINO SIMPLE	LILYPAD ARDUINO SIMPLE SNAP				
3D PRINTING	MATERIA 101					

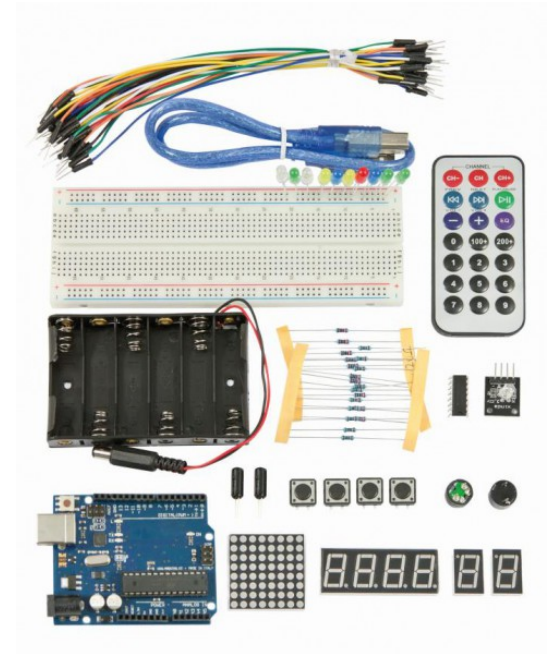
[www.arduino.cc/en/Main/Products]

- Many “clones” are sold as _duino, e.g. 4duino.

The 4duino Starter Kits



Standard



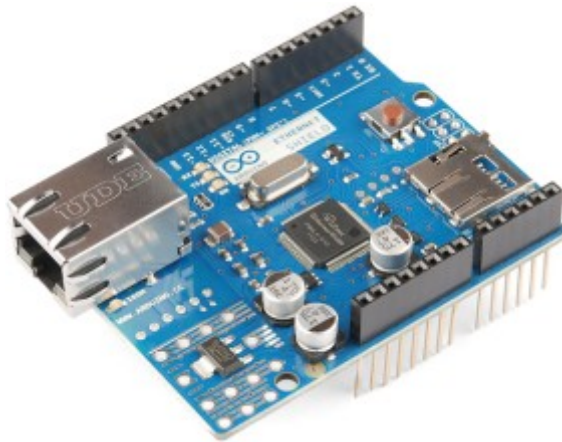
Light

Arduino UNO R3 compatible [www.allnet.de]

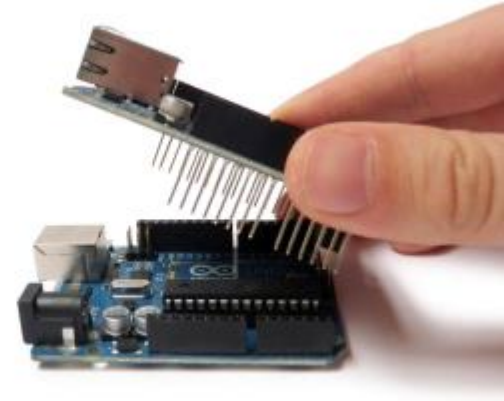
Arduino Shields

- “Arduino shields are modular circuit boards that piggyback onto your Arduino to instill it with extra functionality.”

[source: learn.sparkfun.com]



Ethernet Shield



Connecting the Shield

Tutorial Outline

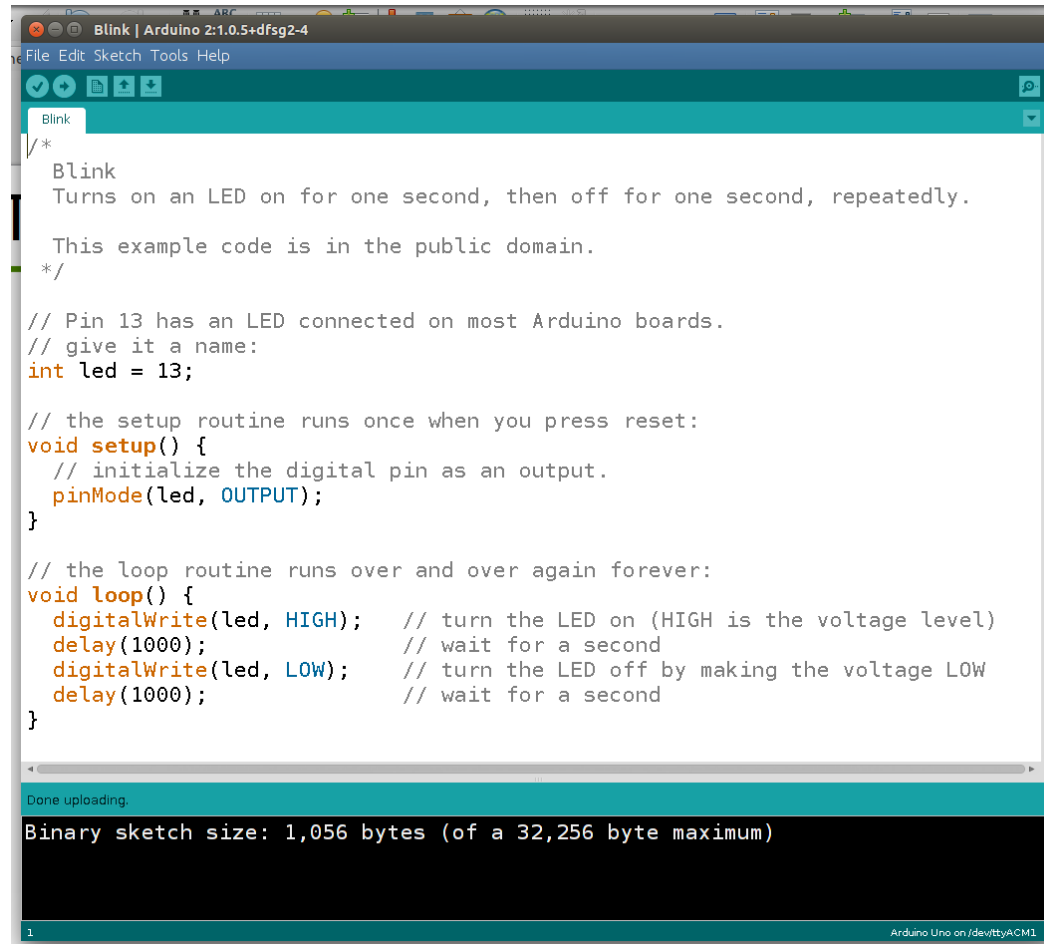
- Part 1 – Introduction to Arduino
 - Structure of an Arduino program
 - Analog and digital input and output
 - Interrupts and timers
 - Additional exercises
- Part 2 – Introduction to networking with Arduino
 - Prerequisites: netcat, wireshark, serial communication
 - The ESP8266 chip and its AT command language
 - Setting up serial communication between Arduino and ESP8266 chip
 - Sending and receiving data
 - Additional exercises

How to Program an Arduino

- The “Arduino Software” is an Integrated Development Environment (IDE). Download from www.arduino.cc/Main/Software
Runs on Win, Mac, and Linux.
- The Arduino programming language looks like C and programs are translated into C behind the scene. Language Reference: www.arduino.cc/en/Reference/HomePage
- An Arduino can be programmed virtually in an online simulator. Login at circuits.io/lab

Firefox is recommended.

The Arduino Software (IDE)



The screenshot shows the Arduino IDE interface. The main window displays the 'Blink' sketch code. The code is as follows:

```
/*
 * Blink
 * Turns on an LED on for one second, then off for one second, repeatedly.
 *
 * This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

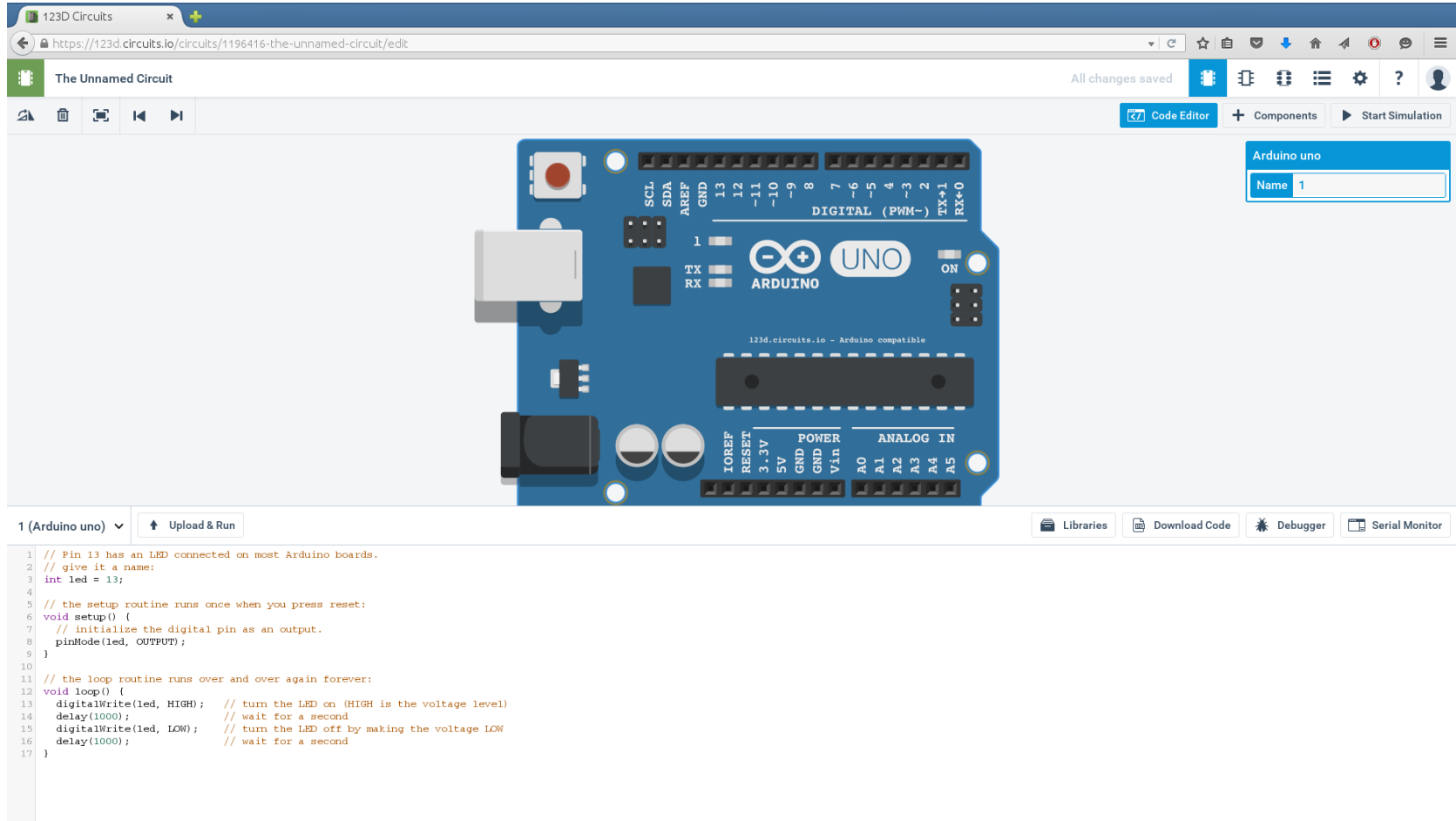
// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);             // wait for a second
  digitalWrite(led, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);             // wait for a second
}
```

Below the code editor, a status bar indicates 'Done uploading.' and 'Binary sketch size: 1,056 bytes (of a 32,256 byte maximum)'. The bottom status bar shows '1' and 'Arduino Uno on /dev/ttyACM1'.

A builtin language reference is available under Help → Reference

Arduino Simulator (circuits.io/lab)



The screenshot displays the Arduino Simulator interface on a web browser. The main window shows a virtual Arduino Uno board with various components like a USB Type-C port, a DC power jack, and a reset button. The board is labeled "123d.circuits.io - Arduino compatible".

At the top, the browser address bar shows the URL: <https://123d.circuits.io/circuits/1196416-the-unnamed-circuit/edit>. The page title is "The Unnamed Circuit".

On the right side, there are buttons for "Code Editor", "Components", and "Start Simulation". Below these, a dropdown menu shows "Arduino uno" with a "Name" field containing the value "1".

At the bottom, there is a code editor with the following code:

```
1 // Pin 13 has an LED connected on most Arduino boards.
2 // give it a name:
3 int led = 13;
4
5 // the setup routine runs once when you press reset:
6 void setup() {
7   // initialize the digital pin as an output.
8   pinMode(led, OUTPUT);
9 }
10
11 // the loop routine runs over and over again forever:
12 void loop() {
13   digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
14   delay(1000);             // wait for a second
15   digitalWrite(led, LOW);  // turn the LED off by making the voltage LOW
16   delay(1000);             // wait for a second
17 }
```

Below the code editor, there are buttons for "Libraries", "Download Code", "Debugger", and "Serial Monitor".

“Hello world” for Arduino

```
int ledPin = 13; // Pin 13 has an LED connected
                  // on most Arduino boards

// the setup routine runs once when you press reset:
void setup()
{
    pinMode(ledPin, OUTPUT); // initialize the digital pin as an
                             // output
                             // the keyword OUTPUT is predefined

// the loop routine runs over and over again forever:
void loop()
{
    digitalWrite(ledPin, HIGH); // turn the LED on
    delay(1000);                // wait for a second
    digitalWrite(ledPin, LOW);  // turn the LED off
    delay(1000);                // wait for a second
}
```


Arduino language vs. standard C

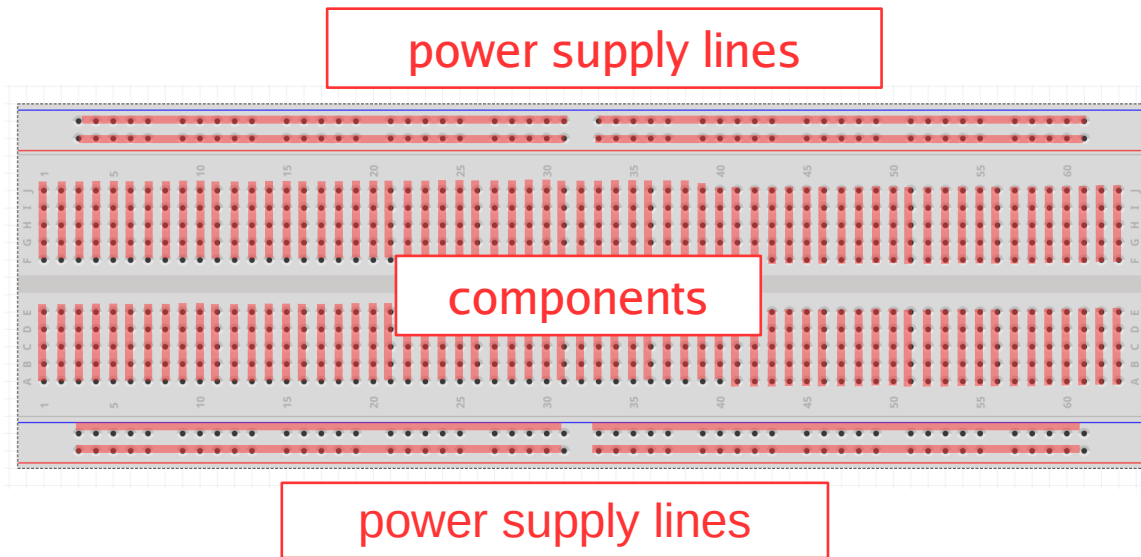
- No `main()` function; instead:
 - `setup()` – executed once at start of program
 - `loop()` – executed infinitely often after `setup()` is completed
- Variables can be declared
 - globally (at the beginning of program outside `setup()` or `loop()`)
 - locally in `setup()` or `loop()`
 - more locally inside `{}` bodies of `if()` decisions, `for()` loops etc. as in standard C
- “not equal” operator
 - In Arduino: “`!=`” in contrast to C's “`<>`”
- Additional data types
 - `boolean`: takes the values `true` and `false`
 - `byte`: synonym for unsigned `char`
 - `word`: synonym for unsigned `short`

About Arduino Pins

- Arduino can input and output both digital and analog data
- Some of the pins serve all purposes, others are restricted to e.g. only analog input
- Some of them are accessible externally, but also connected internally (e.g. to control on-board LED) – can be used if internal function is not needed for your project
- Details depend on the particular Arduino board model

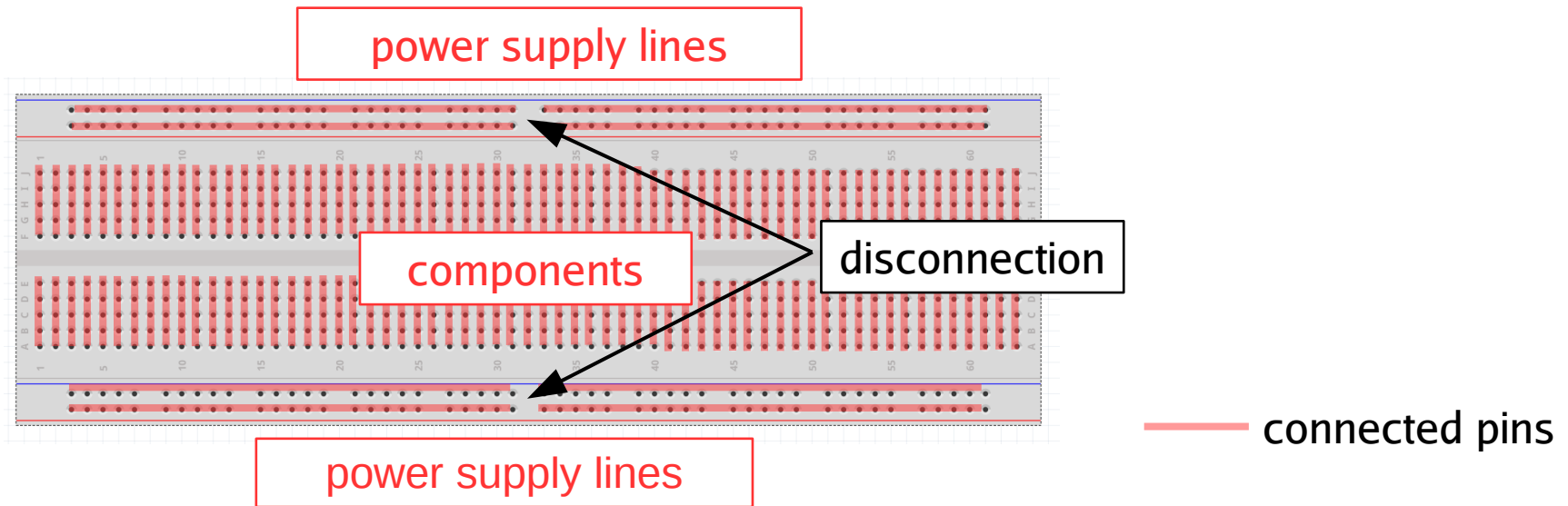
Built-in LED / External LED

- You can also use an external LED instead of the built-in one.
- A resistor (220Ω – $1k\Omega$) is required, otherwise the LED will be destroyed.
- Check correct polarity of the LED: the anode is identified by the long pin.



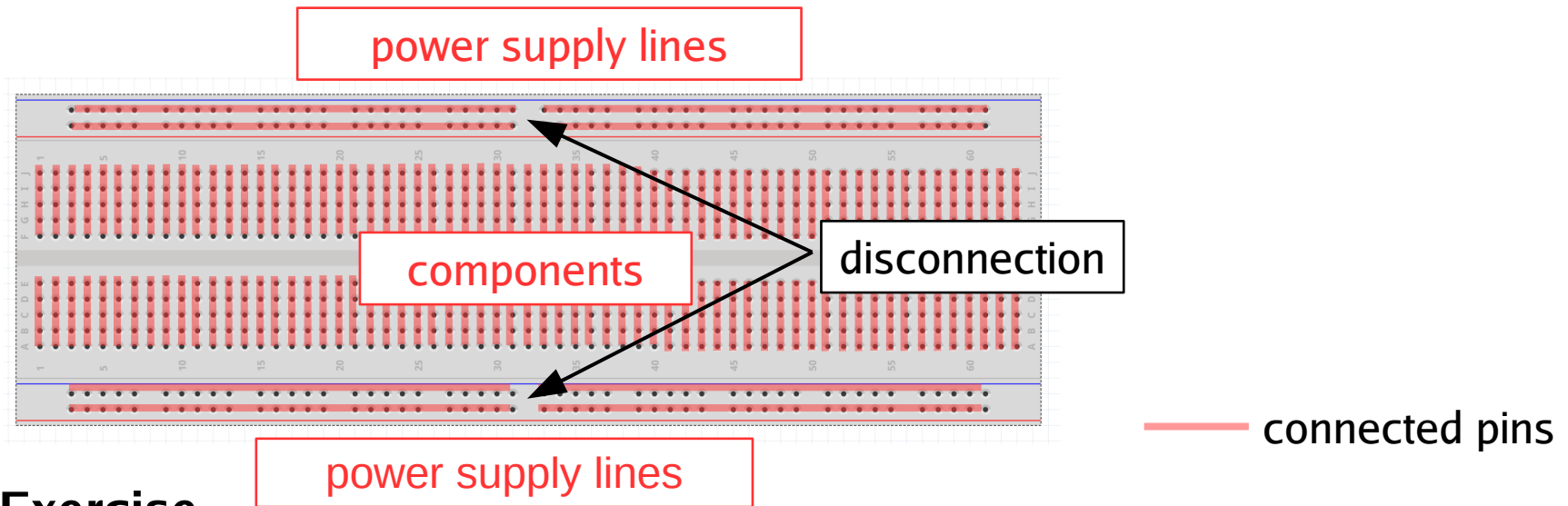
Built-in LED / External LED

- You can also use an external LED instead of the built-in one.
- A resistor (220Ω – $1k\Omega$) is required, otherwise the LED will be destroyed.
- Check correct polarity of the LED: the anode is identified by the long pin.



Built-in LED / External LED

- You can also use an external LED instead of the built-in one.
- A resistor (220 Ω –1k Ω) is required, otherwise the LED will be destroyed.
- Check correct polarity of the LED: the anode is identified by the long pin.



Exercise

Connect the external LED to pin 14 and modify the “Hello World” program to use the external LED.

Digital Input (1)

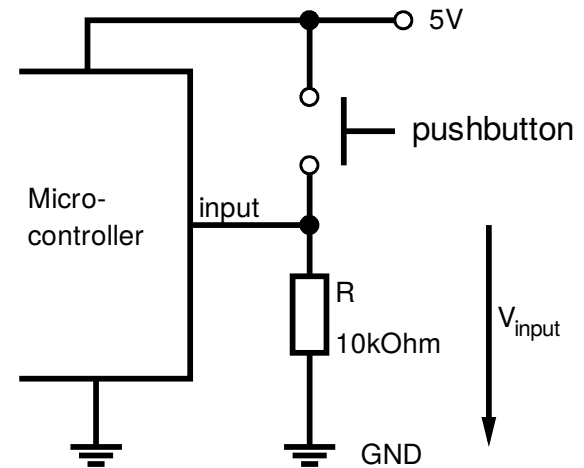
- As seen in the “Hello World” example, the `pinMode(pin_number, mode)` function is used to configure whether a pin should work as input or output
 - For input, *mode* is set to `INPUT`.
- Digital input is read from a pin using the `digitalRead()` function:

```
/* global variables */  
int inputPin = 15;  
int value;  
  
/* put this into the loop() part */  
value = digitalRead(inputPin);
```

Digital Input (2)

Exercise

- Connect the pushbutton and the 10kΩ resistor to pin 15 as shown in the circuit diagram.
 - The input pin of the Arduino has a high impedance so that the pulldown resistor is used to eliminate noise on the input.
- Modify the “hello world” program so that the LED is on when the pushbutton is pressed and vice versa.



Avoiding external pull-up resistor

- The Arduino has builtin resistors which can be attached to any input pin as pull-up resistors by the `pinMode()` function.
- To enable, set the second argument of the `pinMode()` function call to `INPUT_PULLUP`, respectively.

Exercise

- Modify the code so that the internal pullup resistor is used.
- Remove the external pulldown resistor from the previous experiment and connect the pushbutton in a suitable way so that the LED status changes when the button is pressed.

Analog Input (1)

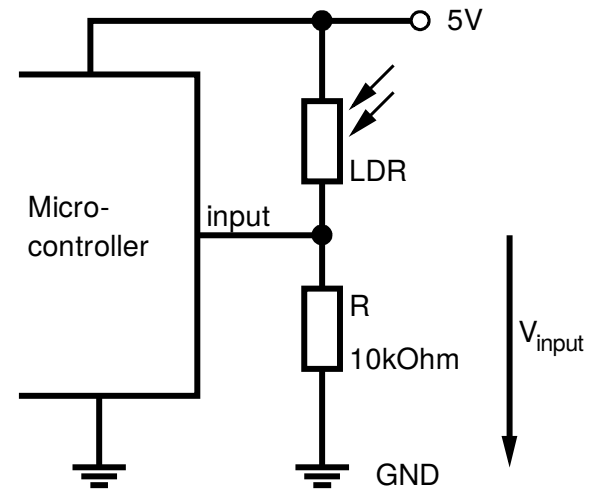
- The arduino includes 10-bit A/D converters which measure the voltage between an input pin and GND
- 0V is indicated as 0 and 5V is indicated as 1023
- Initialising the pin: in the same way as digital inputs using `pinMode(pin_number, INPUT)`
- Analog input is read from a pin using the `analogRead()` function:

```
/* global variables */  
int inputPin = 15;  
int value;  
  
/* put this into the loop() part */  
value = analogRead(inputPin);
```

Analog Input (2)

Exercise

- Modify the circuit by replacing the pushbutton by the LDR as given in the circuit diagram.
- Modify the “hello world” program so that the LED is on when the light exposure exceeds a certain value.
 - Hint: you might have to try with different threshold values until you find one where the experiment works well.



(Pseudo)-Analog Output

- The Arduino can produce pseudo-analog a.k.a. PWM output on a pin using the `analogWrite()` function:
`analogWrite(pin, duty_cycle);`
duty_cycle is a value between 0 and 255 where 0 means permanently off and 255 means permanently on

Exercise

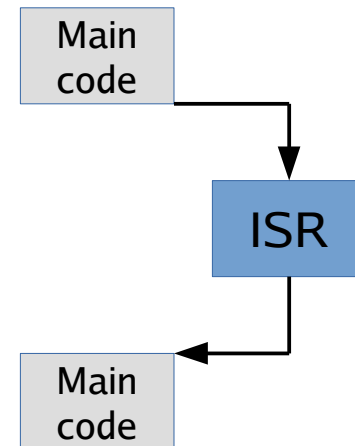
- Extend the previous program so that the voltage reading from the LDR is output through the external LED as light intensity.
- Save a separate copy of the code before continuing. It will be needed for the experiments of the WLAN part of this workshop.

Multitasking

- If you want to run multiple tasks on the Arduino, avoid the `delay()` function
- `delay()` internally runs a timing loop and restrains the processor from doing any other task
- Multiple applications using the `delay()` function can run into problems
- Solve this problem by using timers and interrupts

Interrupts

- State change of a hardware unit which results in the execution of certain program code called “interrupt service routine (ISR)”
- Procedure
 - Main code execution is suspended
 - ISR is executed
 - Execution returns to main code
- Allows for responding to events or timing without blocking the main program execution



Interrupt Sources

- External Interrupts

- A pin changing state
- Serial communication unit when information has been received
- Analog to Digital converter having established conversion

- Internal Interrupts

- Timers

External interrupts (1)

- External interrupts are triggered by external events (e.g. pressing a pushbutton) on an interrupt pin of the Arduino
- Arduino has two digital pins which can be configured to receive external interrupts
 - INT0 (pin D2)
 - INT1 (pin D3)
- Configure the pin to be used as input as seen earlier using the `pinMode()` function

External interrupts (2)

- Assign interrupt to pin
 - `AttachInterrupt(pin, ISR, mode)` ;
 - *pin* could be specified directly, however for compatibility check with different Arduino flavors it is better to give the function call `digitalPinToInterrupt(pin)` as the first argument to `AttachInterrupt()`
 - *ISR* is the name of the interrupt service routine – the function which serves the interrupt – without arguments
 - *Mode* can be:
CHANGING – trigger interrupt whenever pin changes the state
RISING resp. FALLING – trigger on rising resp. falling edge
LOW – trigger interrupt whenever pin is low

External interrupts (3)

“Hello world” example using interrupts

```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, FALLING);
}

void loop() {
  digitalWrite(ledPin, state);
}

void blink() {
  state = !state;
}
```

Exercise

Run this example on your Arduino and check whether the output is as expected. Also check the effect when setting the last argument of `attachInterrupt()` is set to `RISING`

Timer Interrupts (1)

- Timer: internal clock used to measure time events
- An Arduino has three timers
 - Timer0, Timer2 (8 bit)
 - Timer1 (16 bit)
- Timers are controlled by programming Arduino's hardware registers
 - Difficult in particular for beginners

Timer Interrupts (2)

- For the control of e.g. Timer1, a library is available
 - Also supports pulse width modulation (PWM)
 - `Timer1.Initialise(duration_in_microseconds) ;`
 - `Timer1.attachInterrupt(function_to_be_called) ;`
 - `Timer1.pwm(Pin_with_PWM_output, duty_cycle) ;`
 - whenever the timer has expired, the configured pin is set to HIGH for a duration of $\textit{duration_in_microseconds} * \textit{duty_cycle}$

Timers and PWM example (1)

```
// the additional library "timerone" is needed to compile this code
#include "TimerOne.h"

#define period 5000000 // sets the pwm period in microseconds
#define duty 512 // sets the duty cycle of pwm to 50%
#define ledPin 13
#define pwmPin 9 // check this with external LED

void setup()
{
    pinMode(ledPin, OUTPUT);
    Timer1.initialize(period);
    Timer1.pwm(pwmPin, duty);
    Timer1.attachInterrupt(callback); // attaches callback() as a timer overflow interrupt
}

void callback()
{
    digitalWrite(ledPin, digitalRead(ledPin) ^ 1); // toggles the led at timer overflow
}

void loop()
{
}
```

Timers and PWM example (2)

Exercise

- In order to run the code from the previous page, an additional library is needed which is not part of the Arduino IDE's standard installation
 - In the main menu, open Sketch → Include Library → Manage Libraries
 - The Library Manager window opens. Enter the search term "TimerOne" into the search field on the right.
 - Click onto the "TimerOne" entry which then appears, press the "install" button. The library is now ready to use.
 - Use the external LED to check the function of the PWM output.
- Modify the PWM output example from slide 24 by replacing the `analogOut()` function by a digital output and implementing the PWM yourself as shown in the code of the previous slide

Further exercises (1)

- Write a program which periodically dims the LED up and down smoothly once per second.
- Extend the previous program so that whenever the pushbutton is pressed, the dimming period is increased by one second, until 5 seconds. After that, when the pushbutton is pressed further, the period is reduced by one second, until the starting state is reached.
- Extend the LDR experiment from slide 24 using the pushbutton in the following way:
 - Connect the pushbutton to a free digital input pin and configure the internal pullup/pulldown resistor.
 - When the pushbutton is pressed the first time, the LDR voltage reading is stored, assuming that the LDR is fully covered.
 - While the system waits for the second pressing of the pushbutton, the LED blinks
 - When the pushbutton is pressed the second time, the LDR voltage reading is stored, assuming that the LDR is fully exposed to the environmental light.

Further exercises (2)

- Write a program which measures the duration for which the pushbutton is pressed and light the LED for the same time once the pushbutton is released
 - Hint: you can use the function `millis()` which returns the number of milliseconds since start of the program as an `unsigned long`
- Write a program which outputs the LDR voltage reading as a binary number using the LED
 - You can e.g. use long light pulses for a '1' digit and short pulses for '0'
 - Alternatively, you can output the digits of the decimal number using morse code

Other examples

- Try out further examples yourself from
 - File → Examples in Arduino IDE
- How to add external libraries to Arduino IDE (if not available in the IDE's library manager)
 - Download the library in zip format
 - In Arduino IDE go to Sketch → Include Library → Add .zip Library

Please download for tomorrow

- netcat

- Windows: <https://eternallybored.org/misc/netcat>
- Linux/Mac: included in any standard OS installation

- wireshark

- Windows/Mac: www.wireshark.org
- Linux: select the respective package in your package manager

End of Part 1

Tutorial outline

- Part 1 – Introduction to Arduino
 - Structure of an Arduino program
 - Analog and digital input and output
 - Interrupts and timers
 - Additional exercises
- Part 2 – Introduction to networking with Arduino
 - Software tools: netcat, wireshark, serial monitor
 - The ESP8266 chip and its AT command language
 - Setting up serial communication between Arduino and ESP8266 chip
 - Sending and receiving data using the ESP8266
 - Additional exercises

Tools – netcat

- “swiss army knife” for networking diagnostics
- Opens TCP or UDP connections on arbitrary ports
- In our workshop used as a UDP server to communicate with the Arduino
- Start inside a terminal as a listener with:
`netcat -u -l -p portnumber`
portnumber is a number between 1024 and 65535
- Any data sent to netcat will be displayed inside the terminal window
- Any data typed into the terminal window will be sent to the client

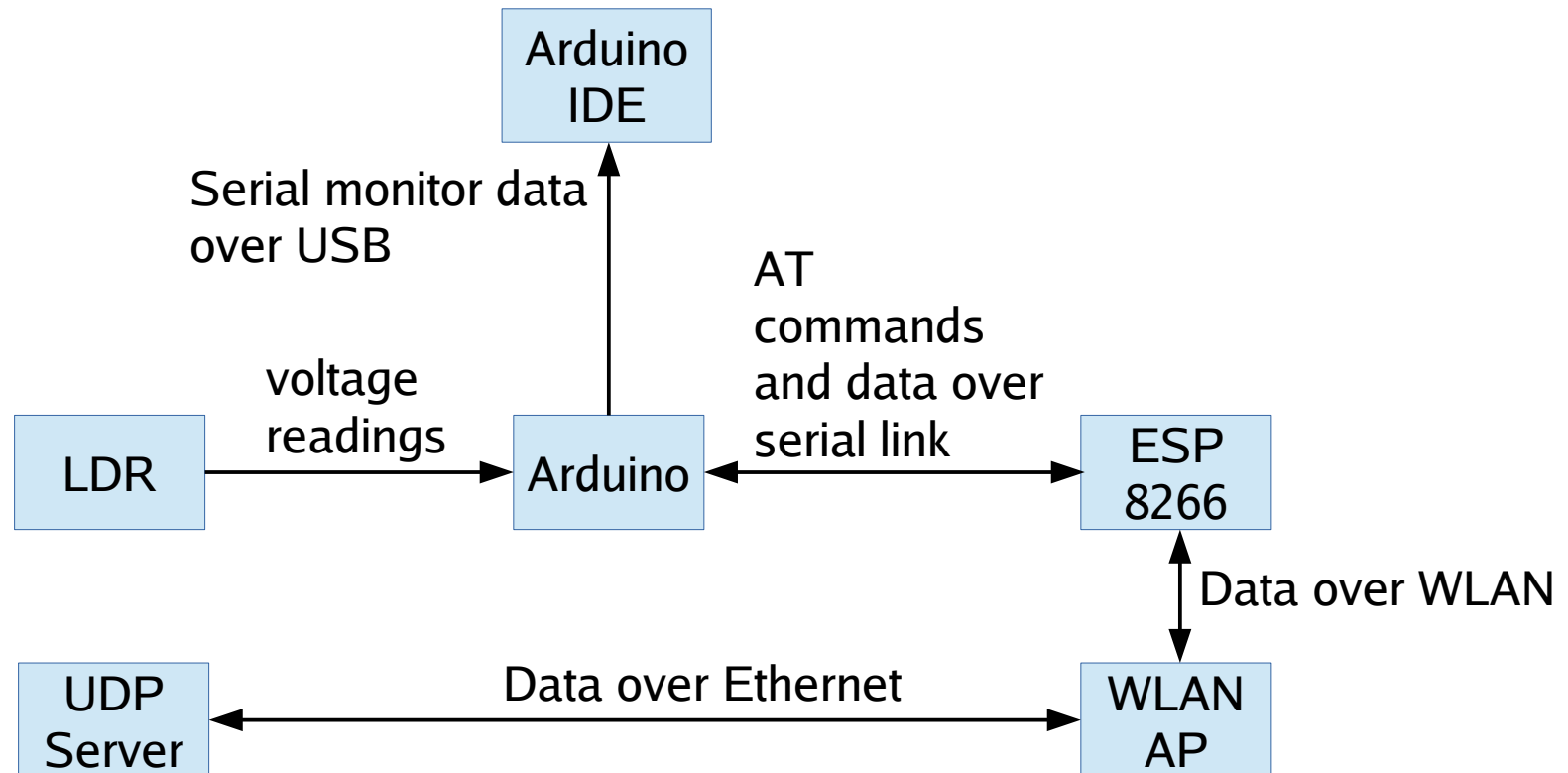
Tools – wireshark

- Graphical network monitoring tool, very useful for troubleshooting
- Grabs all packets which are sent or received over a network interface and displays all protocol headers and the payload
- Protocol headers are interpreted and displayed in a human-readable format
- Usage: see demonstration

Tools – Serial Monitor

- USB connection between computer and Arduino can emulate a serial link
- Serial monitor is part of Arduino development IDE
 - Displays data which is sent onto the serial link
- Accessible through Tools → Serial Monitor
 - Selection of serial device may be required
- Useful for development (avoids the need to operate a display directly by Arduino) and debugging

Experimental Setup and Data Flow



Using Serial Communication

- Initialise the serial link in the `setup()` part using the command `Serial.begin(bitrate)`;
where *bitrate* is 9600 in this experiment.
- In the `loop()` part, use the command
 - `Serial.print(arg)`; to print a line without line feed
 - `Serial.println(arg)`; to print a line with line feed
 - *arg* is a string constant in quotation marks or a string variable

Exercise

- Take your copy of the code for the LDR exercise in slide 24
- Extend the code so that each LDR voltage reading is printed to the serial output, prepended by a sequence number

ESP 8266 chip

- WLAN controller including TCP/IP stack with supporting functions such as DHCP
- Communicates with Arduino through serial interface
- ESP 8266 is controlled by “AT commands”
 - Human-readable string starting with the sequence “AT” for “attention” followed by an abbreviated command and optional comma-separated arguments
 - e.g. AT+RST – perform a reset
 - e.g. AT+CWJAP=”mrt-workshop”, ”20170907” - connect to access point giving the credentials
 - AT commands originated in analog modems for telephone lines

Combining Arduino with ESP8266

- By WiFi shield which can be plugged together with an Arduino board

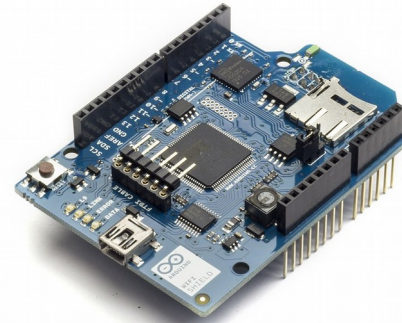


Figure source: store.arduino.cc/arduino-wifi-shield

- Pretzelboard: Arduino nano+ESP8266 integrated on the same board

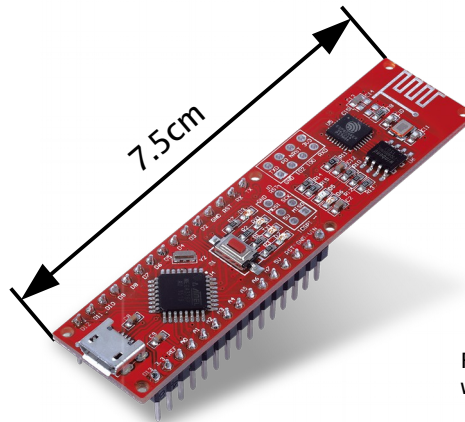


Figure source:
www.franzis.de/maker/raspberry-pi-arduino-und-mehr/pretzel-board-platine

Initialise serial link to ESP chip

- Similar to controlling serial link to Arduino IDE
- Smaller Arduinos only have one hardware serial interface which is already used for the communication with the IDE → further serial ports are provided using digital output pins using software emulation
- At top of program:
 - Declare software serial interface:

```
#include <SoftwareSerial.h>
SoftwareSerial esp8266(11, 12);
    // pin numbers for RX, TX
```
 - Initialise declared interface:

```
esp8266.begin(bitrate)
```

 where *bitrate* = 19200

Reset WLAN chip

- Sending AT commands to the 8266:
`esp8266.println("command");`
- Checking response of 8266:
`result=esp8266.findUntil("string1", "string2");`
parses data received through the serial interface until either of the strings is found. *result* is true if *string1* is found, otherwise false.
- Reset WLAN chip: AT+RST
The chip returns OK or ERROR as a response.

Exercise

- Initiate a reset and print out whether it was successful or not.

Set WLAN operation mode

- Set operation mode for WLAN interface:
`AT+CWMODE=mode`
 - *mode* may be set to:
 - 1 – station
 - 2 – access point
 - 3 – station and access point simultaneously
- The command returns OK or ERROR.
- General hint applicable to all AT commands
If an AT command takes one or more arguments, they are separated from the command by the “=” sign. No blanks should be placed before or after the “=” sign.

Set WLAN access credentials

- Set access credentials:
`AT+CWJAP="ssid", "password"`
The command returns OK or ERROR.
- General hints
 - Multiple arguments are separated by commas. No blanks should appear before or after the commas.
 - String arguments are enclosed in quotation marks.
- Caution: in order to send an AT command with string arguments using the `Serial.println` function, the quotation marks must be masked with backslashes because the argument for the `println` function is itself enclosed in quotation marks:
e.g. `Serial.println("AT+CWJAP=\"mrt-workshop\", \"20170907\")`;

Initialise UDP connection

- Obtain IP address from a DHCP server:

AT+CIFSR

The command takes no arguments, it returns OK or ERROR.

- Open a UDP connection to a server:

AT+CIPSTART="UDP", *dest_ip*, *dest_port*, *source_port*, 0

- Caution: the last three arguments are numbers, not strings, hence there are no quotation marks.

- Usage example:

```
Serial.println("AT+CIPSTART=\"UDP\", \"192.168.1.1\", 9999, 8888, 0");
```

The command returns OK or ERROR.

Exercise

Configure the WLAN as a station as described on the previous slide and prepare the UDP connection. Specify your laptop's IP as the destination. Check the correct execution of each command with the `findUntil` function and print the execution status to the serial monitor.

Send a fixed message (1)

- Send a data packet: `AT+CIPSEND=message_length`
- `message_length` is a variable integer value

Exercise

- At the beginning of program: Create a char array and assign a string to it.
- Manually count the number of characters.
- In the `loop()` section, send `AT+CIPSEND` along with the string size to the ESP8266.
- After that, send the message using `esp8266.write(string)`;
- Wait a second to allow the system to process the sending request using the function `delay(milliseconds)`.
- Prepare the UDP server on your laptop:
`netcat -u -l -p portnumber`
 - the portnumber must match the 3rd argument of the `AT+CIPSTART` call on the previous slide

Send a fixed message (2)

Example

- In `loop()` section:

```
esp8266.println("AT+CIPSEND=10");  
delay(500); // the ESP needs time to process the  
           // previous command  
esp8266.write("teststring");  
delay(1000);
```
- Enter the code *after* the voltage reading was taken from the LDR. We will later on modify the code so that the voltage readings are sent instead of the fixed message.
- When running the program, you should now see packets sent from the Arduino to the laptop. Since the laptop does not know how to handle them, it will respond by sending an ICMP “port not reachable” message.

Sending variable messages (1)

- Problem: up to now, we just sent a constant string as the message. How to send variable data such as the LDR voltage reading?
 - Put content to be sent into a string variable which is handed over to `esp8266.write()`
 - Prepare content for that string variable using `sprintf()`
 - Usage of Arduino's `sprintf()` like in standard C
 - Determine length of the string using the `strlen()` function
 - Also like in standard C
- at beginning of program:
- ```
char msg_buffer[64];
int msg_length;
```
- in loop() section:
- ```
... /* some code which fills msg_buffer with content */ ...  
msg_length=strlen(msg_buffer);
```

Sending variable messages (2)

Example

- at beginning of program:

```
char msg_buffer[64];  
char command_buffer[64];  
int msg_length;
```

- in loop() section:

```
// prepare message to be sent  
sprintf(msg_buffer, "LDR reading: %d", ldr_reading);  
// determine message length  
msg_length=strlen(msg_buffer);  
// prepare AT command string  
sprintf(command_buffer, "AT+CIPSEND=%d", msg_length);  
// send AT command  
esp8266.println(command_buffer);  
delay(1000);  
// send message  
esp8266.write(msg_buffer);  
delay(1000);
```

Sending variable messages (3)

Exercise

- Modify your message sending code to send out the LDR readings. Also add a sequence number when printing the message.
- Write information about the execution status of each command to the serial monitor.
- Check with netcat or wireshark whether the messages are visible and correctly formatted.

Receiving messages (1)

- The Arduino stores messages received over serial links in a buffer
 - Asynchronous communication
- Messages can be picked up whenever program has time
 - No messages are lost even if the Arduino is waiting/busy because of e.g. a `delay()` function call
- It cannot be predicted when messages will arrive in the buffer
 - Before reading the buffer, first check if there is any content:
`result=esp8266.available()`
`result` is true if there is data to be read, otherwise false.
 - Read one character from the buffer:
`input=esp8266.read();`

Receiving messages (2)

Exercise

- In order to generate messages to be received by ESP8266, simply type into the terminal window of your computer where netcat is running. The data is sent once you hit “enter”.
- Read the received data character by character using a `while()` loop. What should be the termination criterion? Also use the same `while()` loop to print each character to the serial monitor.
 - Hint: you can alternately send and receive data inside the `loop()` part because the input data is buffered if the Arduino is busy with other code as mentioned earlier.

Further exercises

- When receiving messages, the 8266 chip returns a lot of control information along with the messages. Filter all 8266 output which does not belong to the received message and only print the message itself to the serial output.
- Set up the 8266 as a UDP echo server. Use netcat on your laptop as the client which sends data to the ESP8266. The Arduino should read the data and send it back to the laptop where it should appear on the netcat window.

End of Part 2

Acknowledgment

Parts of the material presented have been adapted from lectures given by the Communication Networks working group at University of Bremen, Germany