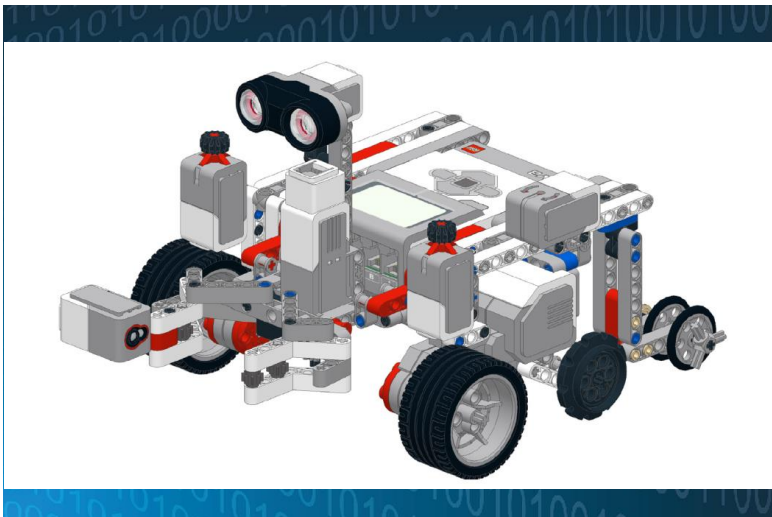




Hochschule Reutlingen
Reutlingen University



Phase 2 – EV3 Hardware and Programming

Computer Science for Engineers

Michael Danner
Markus Wachter

International Project Engineering
Hochschule Reutlingen

Lego Mindstorms EV3 Education Set

- 1 basic education set Lego Mindstorms EV3
- 1 charger and rechargeable battery
- 1 box
- 1 USB cable
- 1 SD card
- 1 Wifi dongle



<https://education.lego.com/de-de/product/mindstorms-ev3>

Homework 1

✚ Phase 2 - Robot ✎



✚ Handing-Out Procedure EV3 Sets ✎

- Please follow the handing-out procedure in Relax
- Due: see Relax course.
- Please upload one document per group only

Handing-Out Procedure EV3 Sets

After you have received the EV3 sets, please follow this procedure:

1. Make an inventory and fill in the inventory template in Excel
2. Upload the filled in template here
3. Collect replacement for any missing parts from Mr. Wachter
4. Please fill in the confirmation from in Word, sign it and upload it here



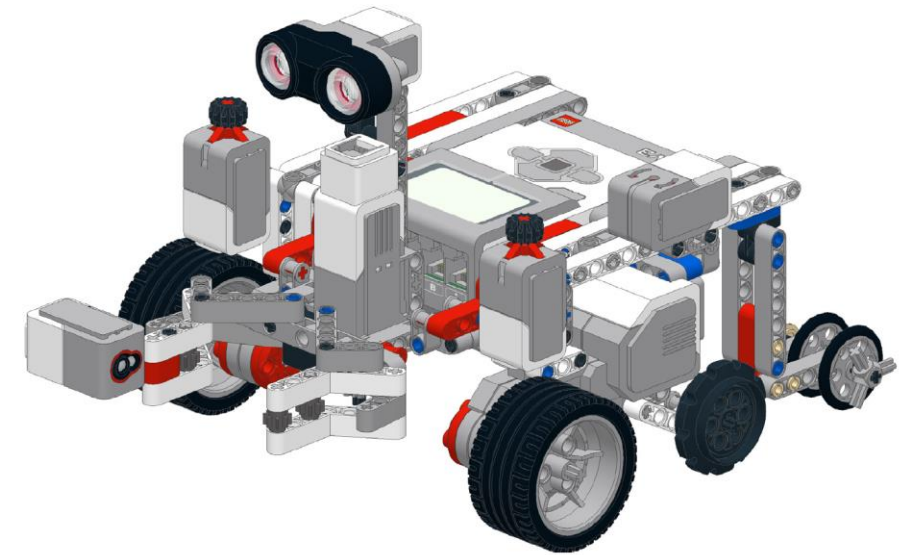
Confirmation_Mindstorm.docx










Inventory_mindstorms_template.xlsx

Homework 2

- Please construct the MarsRov3r
- Assembly instructions see Relax.
- Please note that these instructions are free to use in the classroom only.
Do not distribute the document!



Robot Phase

-  Scrum Recap
-  Project Teams
-  Handing out procedure of the robots
-  How To: Construct the MarsRover 
-  How To: Connect Brick via WiFi (Windows 10)
-  How to set up PyCharm Professional with EV3



Please note:

The building instructions calls for using 7 of the red L-shaped parts. In your set, there are only 6 of these parts available. Please improvise!

- Bring the Mars Rover to the next lecture.

Learning Goals Phase 2

In the second phase you will apply your knowledge of software design and implementation to interact with the sensors and actors of the robot.

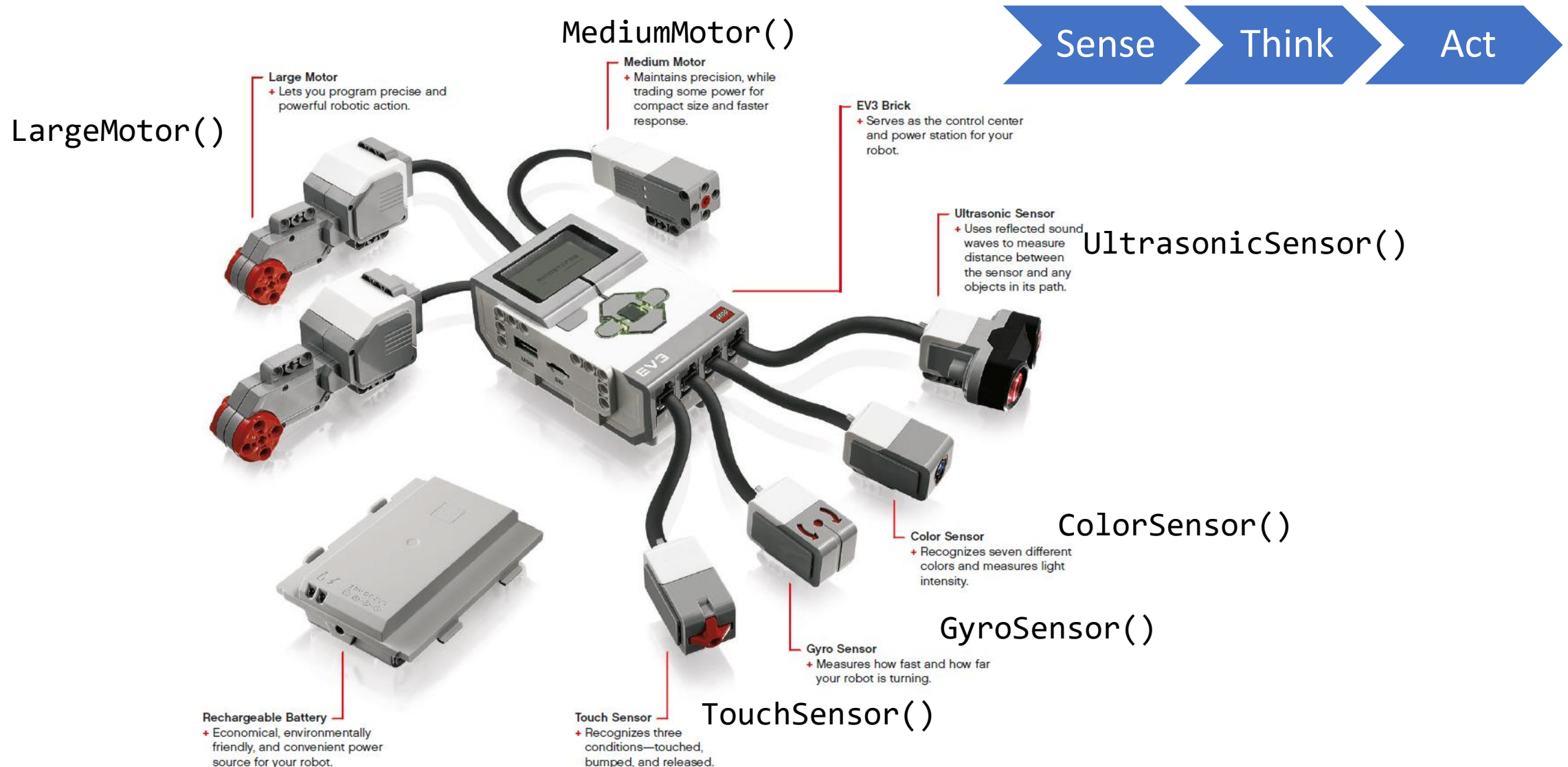
At the end of this phase you are able to

- explain the basic function of the utilized sensors
- interact with the sensors and actors of the robot in Python
- design and implement small programs, transfer and execute them on the robot
- identify typical error types and apply test methods
- describe the data flow within the processor and the memory access of a micro controller

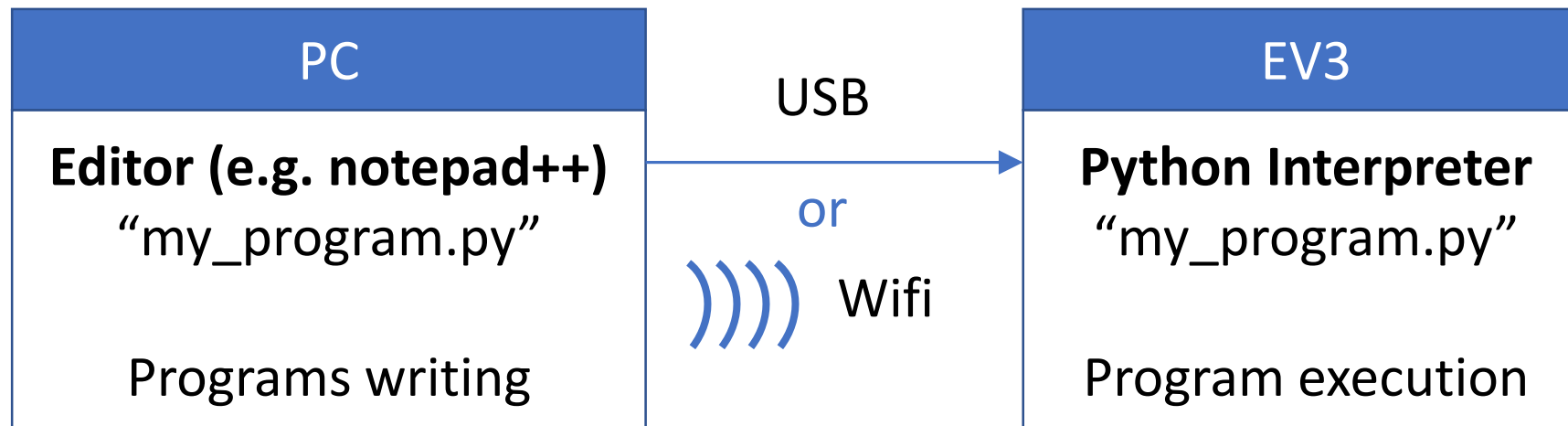
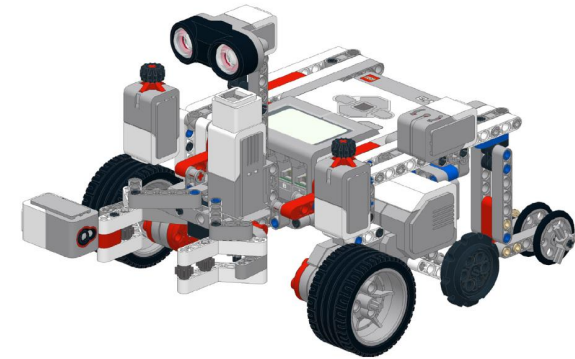
Contents: Python and software design

1. Connecting the robot with the lab computers
2. The EV3 actuators
3. The EV3 sensors
4. Testing and errors
5. Microcontroller architecture

Overview: Addressing EV3 Hardware in ev3dev

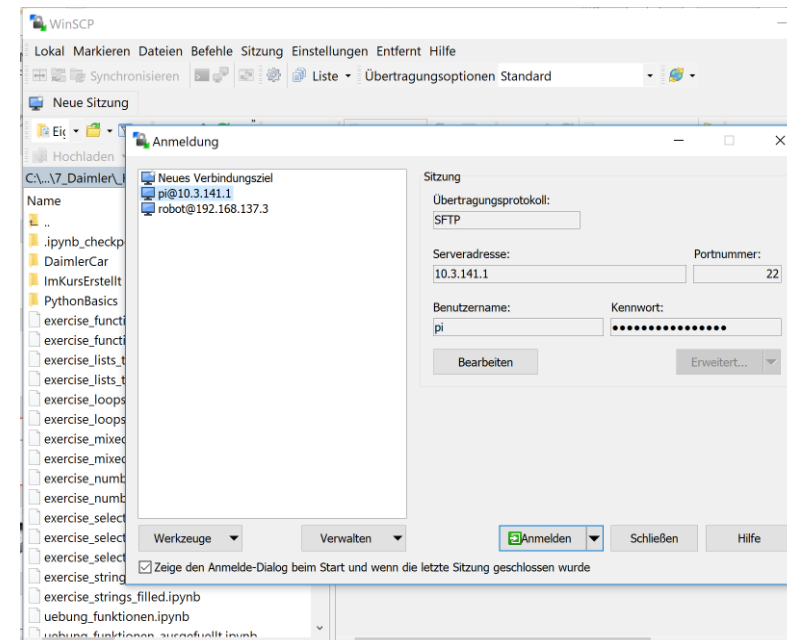
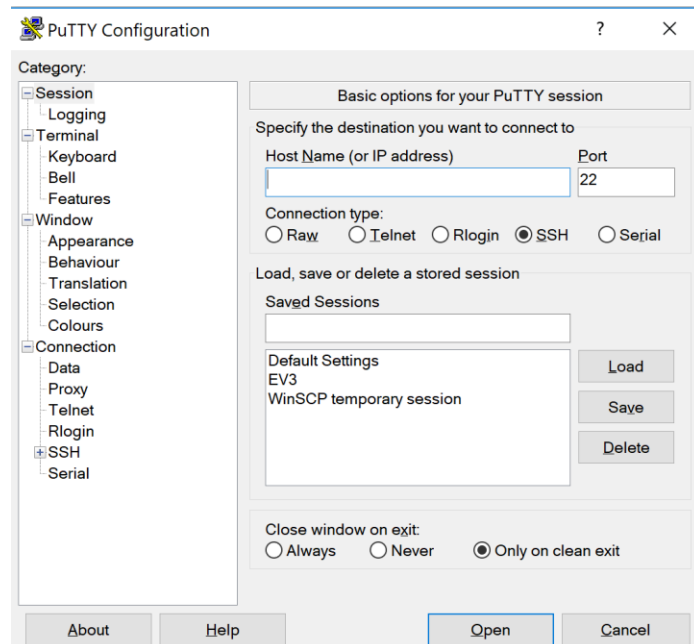


Programming and program execution on the EV3



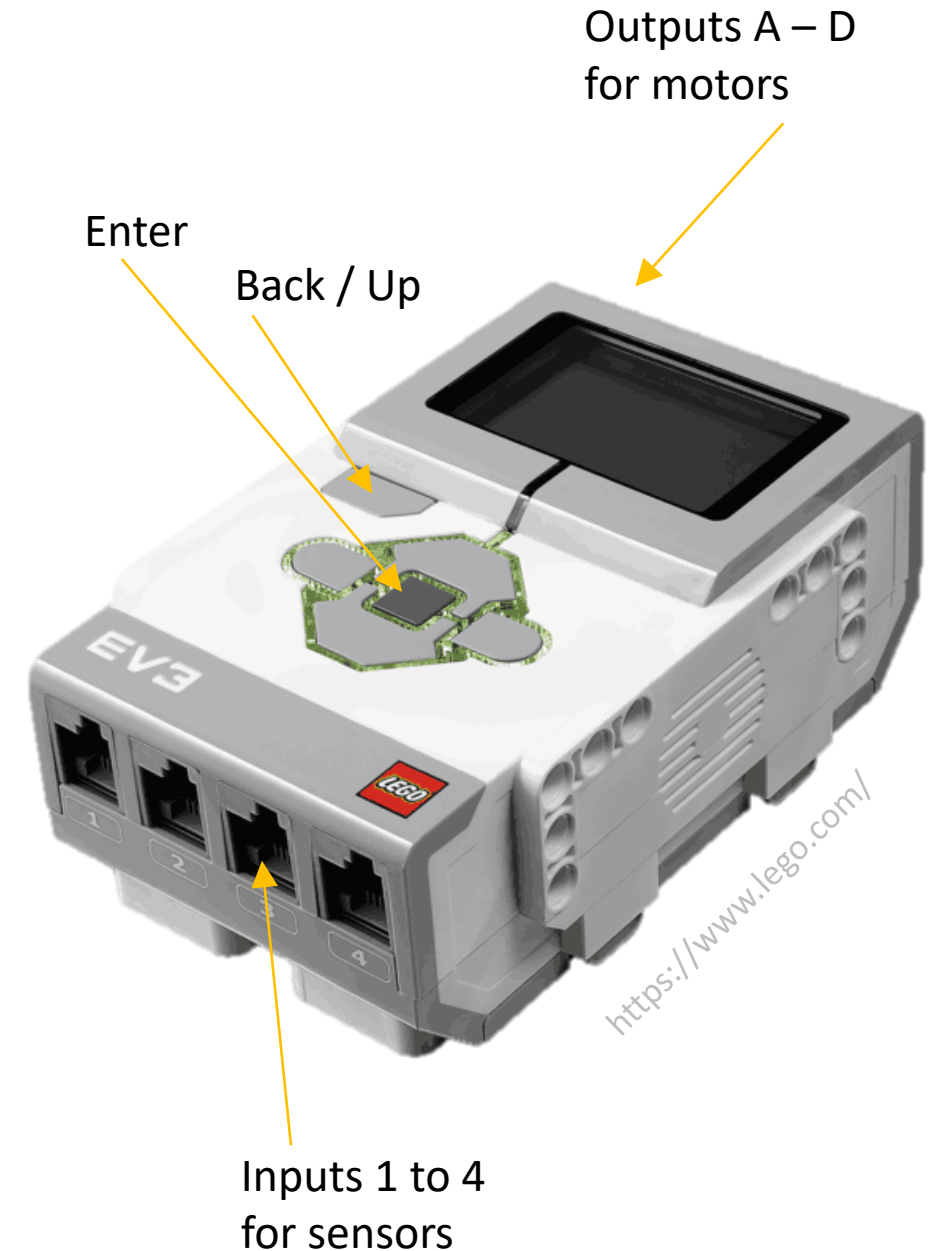
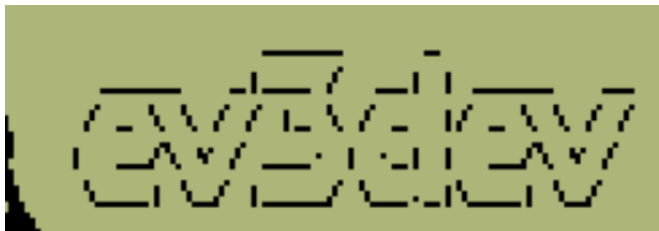
Required Software Installation on own PC

- Putty <https://putty.org/>
- WinSCP <https://winscp.net/eng/index.php> (FTP Client for Windows)
- notepad++ (option) <https://notepad-plus-plus.org/downloads/>



Connect brick with PC

- Connect the PC and the brick via USB cable
- Start the brick by pressing the button (Enter) in the middle of the brick (press the button until the LEDs light up)
- The EV3 boots from SD card with an alternative Operating System that allows programming the robot in Python. Wait until the green LEDs light up)

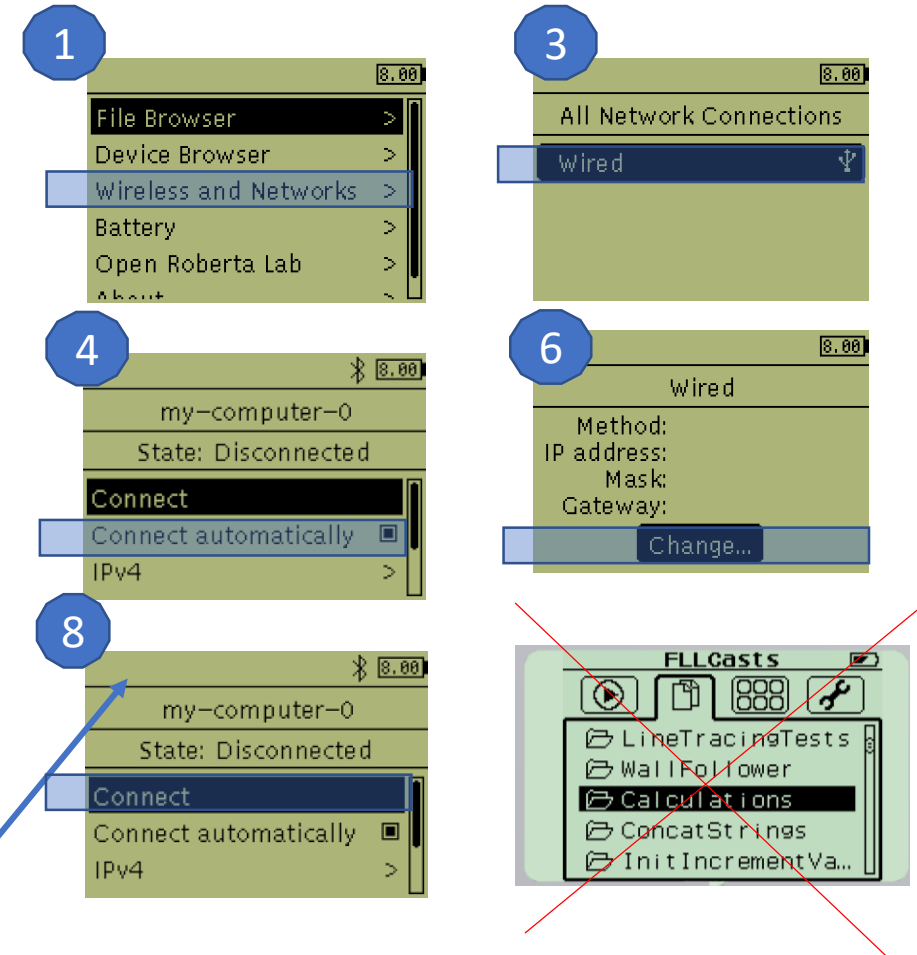


Own PC: Setting up the USB connection – Brick details

In the brick menu:

1. choose **Wireless and Networks**
2. choose **All Network Connections**
3. choose **Wired**
4. Tick the box „connect automatically“
5. choose **IPv4**
6. choose **Change**
7. choose **Load Windows Defaults** (or other OS)
8. Go back one screen and click „**Connect**“
9. Your IP adress is set to a random IP Adress

The connection is established when an IP adress is shown in the top of your brick display.



Task: Setup the network settings of the brick.

Own PC: Setting up the USB connection – PC details

Source:

https://de.wikipedia.org/wiki/Datei:USB-Logo_generic.svg

Follow the instructions for your operating system given on this side:

<https://www.ev3dev.org/docs/tutorials/connecting-to-the-internet-via-usb/>

Task: Setup the network settings of the PC.

Tip: On a German Windows installation, you find the control panel in this way:

- Hit the Windows Key on your Keypad and type „sys“
- Open the „Systemsteuerung“
- Klick on „Geräte und Drucker“

Lab: Setting up the connection – Brick

In the robot menu, assign

- a fixed IP-adress: 169.254.174.xxx, where xxx is a value between 2 and 254 (i.e. your lab computer number)
- Subnetzmaske: 255.255.255.0
- Standardgateway: 169.254.174.1

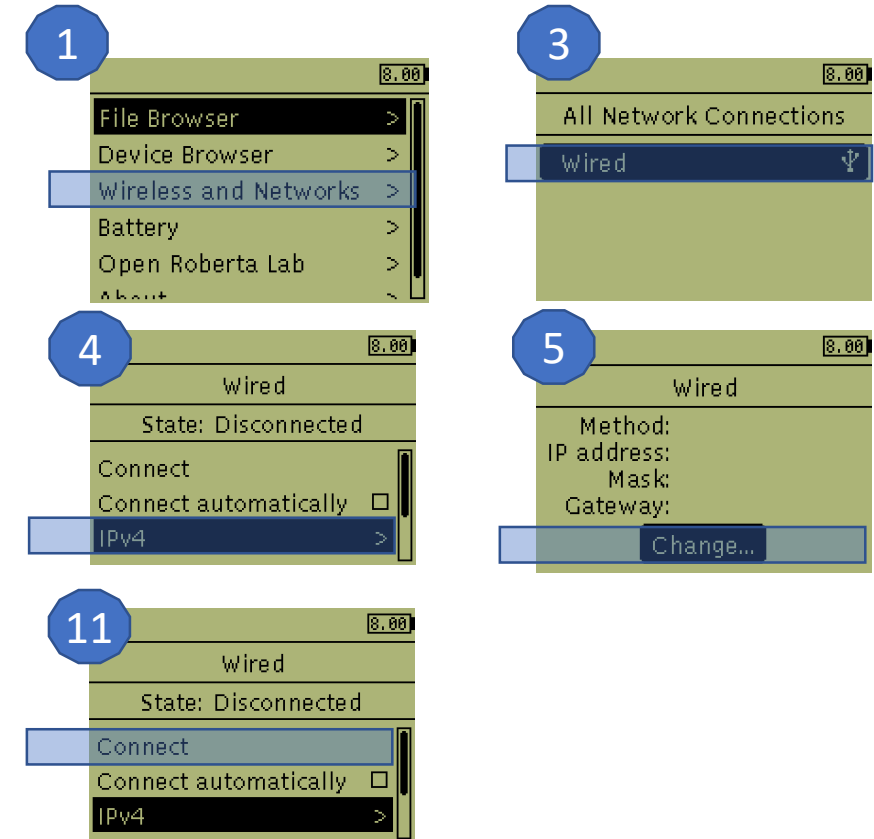
Details see next slide

Lab: Setting up the connection – Brick details

In the brick menu:

1. choose **Wireless and Networks**
2. choose **All Network Connections**
3. choose **Wired**
4. choose **IPv4**
5. choose **Change**
6. choose **Enter custom values**
7. Enter the IP adresse 169.254.174.xxx, where xxx is a value between 2 and 254, i.e. you lab computer number.
8. Network Mask: 255.255.255.0, Gateway: 169.254.174.1
9. Choose Apply
10. Go back one menu level
11. Choose **Connect**

The connection is established when the IP adress is shown.



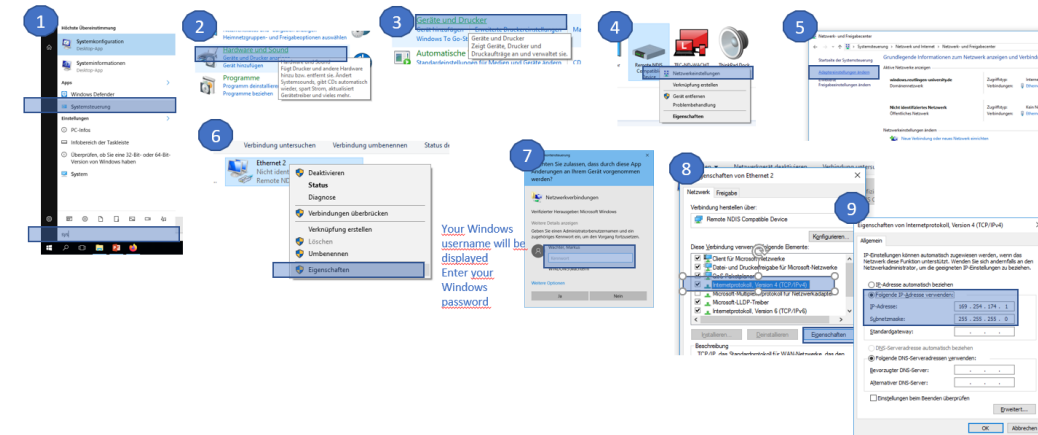
Source of instruction and pictures:

<http://www.ev3dev.org/docs/tutorials/connecting-to-the-internet-via-usb/>

Lab: Setting up the connection – PC details

- Assign the fixed IP address to the Ethernet 2 (Brick) Interface: 169.254.174.xxx/255.255.255.0 (see next slide).
- These IP settings will be stored permanently on the PC until the end of the semester.
- Keep this PC - when switching PCs, it may be necessary to redo the setup.

Task: Setup the network settings of the PC.



Lab: Setting up the connection – PC details

1 Höchste Übereinstimmung
Systemkonfiguration
Systeminformationen
Apps
Windows Defender
Systemsteuerung
Einstellungen
PC-Infos
Infobereich der Taskleiste
Überprüfen, ob Sie eine 32-Bit- oder 64-Bit-Version von Windows haben
System

2 Hardware und Sound
Geräte und Drucker anzeigen
Gerät hinzufügen
Programme
Programm deinstallieren
Programme beziehen

3 Geräte und Drucker
Gerät hinzufügen
Windows To Go-St
Automatische
Standardeinstellungen für Medien und Geräte ändern

4 Remote NDIS Compatible Device
Netzwerkeinstellungen
Verknüpfung erstellen
Gerät entfernen
Problembehandlung
Eigenschaften

5 Netzwerk- und Freigabecenter
Startseite der Systemsteuerung
Grundlegende Informationen zum Netzwerk anzeigen und Verbindung
Aktive Netzwerke anzeigen
windows.reutlingen-university.de
Domänennetzwerk
Zugriffstyp: Internet
Verbindungen: Ethernet
Nicht identifiziertes Netzwerk
Öffentliches Netzwerk
Zugriffstyp: Kein Netz
Verbindungen: Ethernet 2
Netzwerkeinstellungen ändern
Neue Verbindung oder neues Netzwerk einrichten

6 Verbindung untersuchen
Verbindung umbenennen
Status d
Ethernet 2
Nicht ident
Remote ND
Deaktivieren
Status
Diagnose
Verbindungen überbrücken
Verknüpfung erstellen
Löschen
Umbenennen
Eigenschaften

7 Sie erlauben, dass durch diese App Änderungen an Ihrem Gerät vorgenommen werden?
Netzwerkverbindungen
Verifizierter Herausgeber: Microsoft Windows
Weitere Details anzeigen
Geben Sie einen Administratorbenutzernamen und ein zugehöriges Kennwort ein, um den Vorgang fortzusetzen.
Wächter, Markus
Kennwort
WINDOWSwachtem
Weitere Optionen
Ja
Nein

8 Netzwerk
Freigabe
Verbindung herstellen über:
Remote NDIS Compatible Device
Diese Verbindung verwendet folgende Elemente:
☒ Client für Microsoft-Netzwerke
☒ Datei- und Druckerfreigabe für Microsoft-Netzwerke
☒ QoS-Paketplaner
☒ Internetprotokoll, Version 4 (TCP/IPv4)
☐ Microsoft-Multiplexorprotokoll für Netzwerkadapter
☒ Microsoft-LLDP-Treiber
☒ Internetprotokoll, Version 6 (TCP/IPv6)
Installieren...
Deinstallieren
Eigenschaften
Beschreibung
TCP/IP, das Standardprotokoll für WAN-Netzwerke, das den

9 Eigenschaften von Internetprotokoll, Version 4 (TCP/IPv4)
Allgemein
IP-Einstellungen können automatisch zugewiesen werden, wenn das Netzwerk diese Funktion unterstützt. Wenden Sie sich andernfalls an den Netzwerkadministrator, um die geeigneten IP-Einstellungen zu beziehen.
☐ IP-Adresse automatisch beziehen
☒ Folgende IP-Adresse verwenden:
IP-Adresse: 169 . 254 . 174 . 1
Subnetzmaske: 255 . 255 . 255 . 0
Standardgateway: . . .
☐ DNS-Serveradresse automatisch beziehen
☒ Folgende DNS-Serveradressen verwenden:
Bevorzugter DNS-Server: . . .
Alternativer DNS-Server: . . .
☐ Einstellungen beim Beenden überprüfen
Erweitert...
OK
Abbrechen

Your Windows username will be displayed
Enter your Windows password



Own PC: Setting up the WiFi connection

Source:

https://de.wikipedia.org/wiki/Datei:WiFi_Logo.svg

Follow the instructions using the document on Relax:

Robot Phase

Scrum Recap

Project Teams

Handing out procedure of the robots

How To: Construct the MarsRover

How To: Connect Brick via WiFi (Windows 10)

How to set up PyCharm Professional with EV3

You may configure PyCharm to connect to the EV3 brick:

- SFTP Remote Server: Edit code in PyCharm with all features and upload
In addition you should set "Override default permissions for files" in Set

Task: Setup the network settings of the PC.

Hint: You have to set the network type to 2.4 GHz

Mobiler Hotspot

Meine Internetverbindung für andere Geräte freigeben

☐ Aus

Eigene Internetverbindung freigeben von

windows.reutlingen-university.de

Meine Internetverbindung freigeben über

☒ WLAN

☐ Bluetooth

Netzwerkname: myComputer

Netzwerkpassword: 00000000

Netzfrequenzbereich: 2,4 GHz

Bearbeiten

Netzwerkinfos bearbeiten

Ändern Sie den Netzwerknamen und das Kennwort, die andere Benutzer für Ihre geteilte Verbindung verwenden.

Netzwerkname

myComputer

Netzwerkpassword (mind. acht Zeichen)

00000000

Netzfrequenzbereich

2,4 GHz

Speichern

Abbrechen

Own & Lab PC: Accessing the EV3 via WinSCP

Settings

- Protocol (Übertragungsprotokoll): SCP
- Server/Host (Rechnername): IP-Adress displayed on the brick
- User name (Benutzername): robot
- Password (Kennwort): maker
- Port: 22

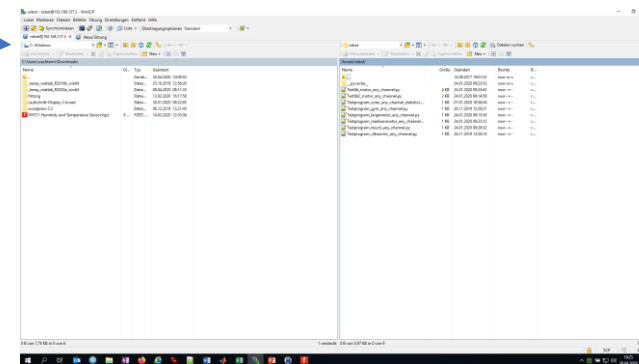
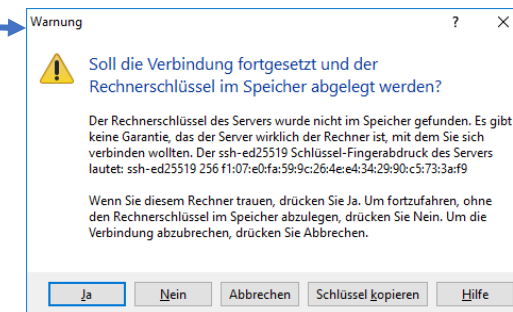
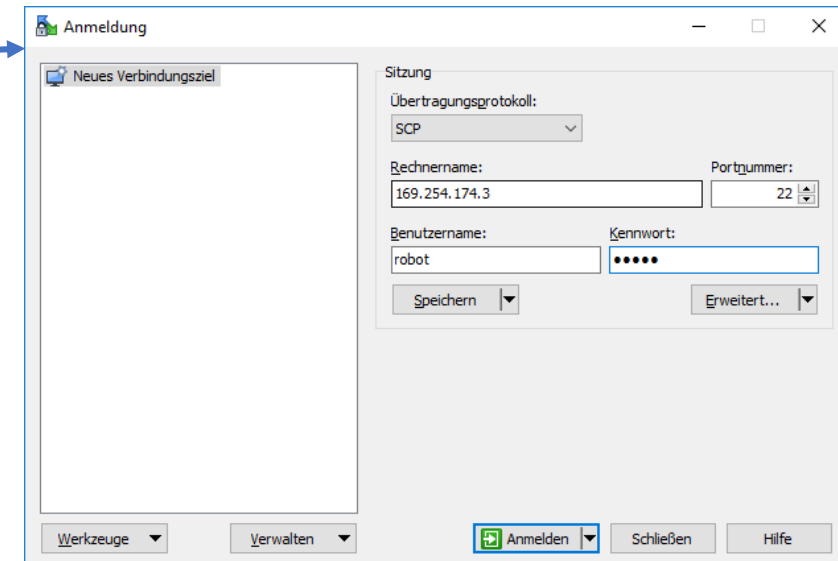
Hint: You can save (speichern) the connection incl. Password

- Connect (Anmelden)
- The first time you connect, an information window needs to be confirmed

Data transmission

- Left window: file structure of the PC
- Right window: file structure of the brick
- Transmit data by drag & drop

Task: Establish a connection between the brick and WinSCP.



Transfer and execution of example program

Transfer the program to the brick

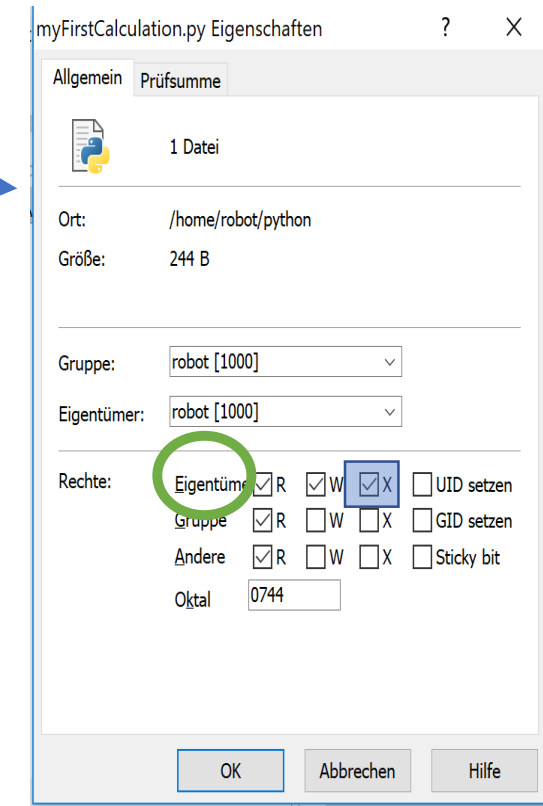
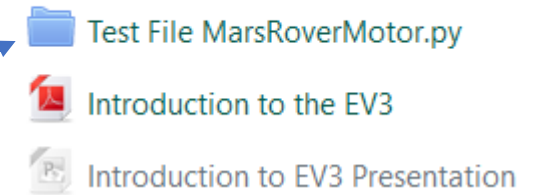
1. Check, if a motor is connected to Output A of the brick.
2. Copy the program „MarsRoverMotor.py“ from Relax to your PC.
3. Move the program in WinSCP onto the brick
4. On the robot, mark the program as executable:
 1. right click on the file and select properties (Eigenschaften)
 2. Rights (Rechte) set the x for the owner (Eigentümer)
 3. On the brick an executable program is marked by a *

Start the program from the brick directly:

- In the brick navigate to the program in the „File Browser“
- The program MarsRoverMotor.py* is highlighted
- Press enter (middle button) to execute the program

Task: Start the program on the brick directly

Pros: Debugging in PyCharm is possible
Cons: Program may run slower.



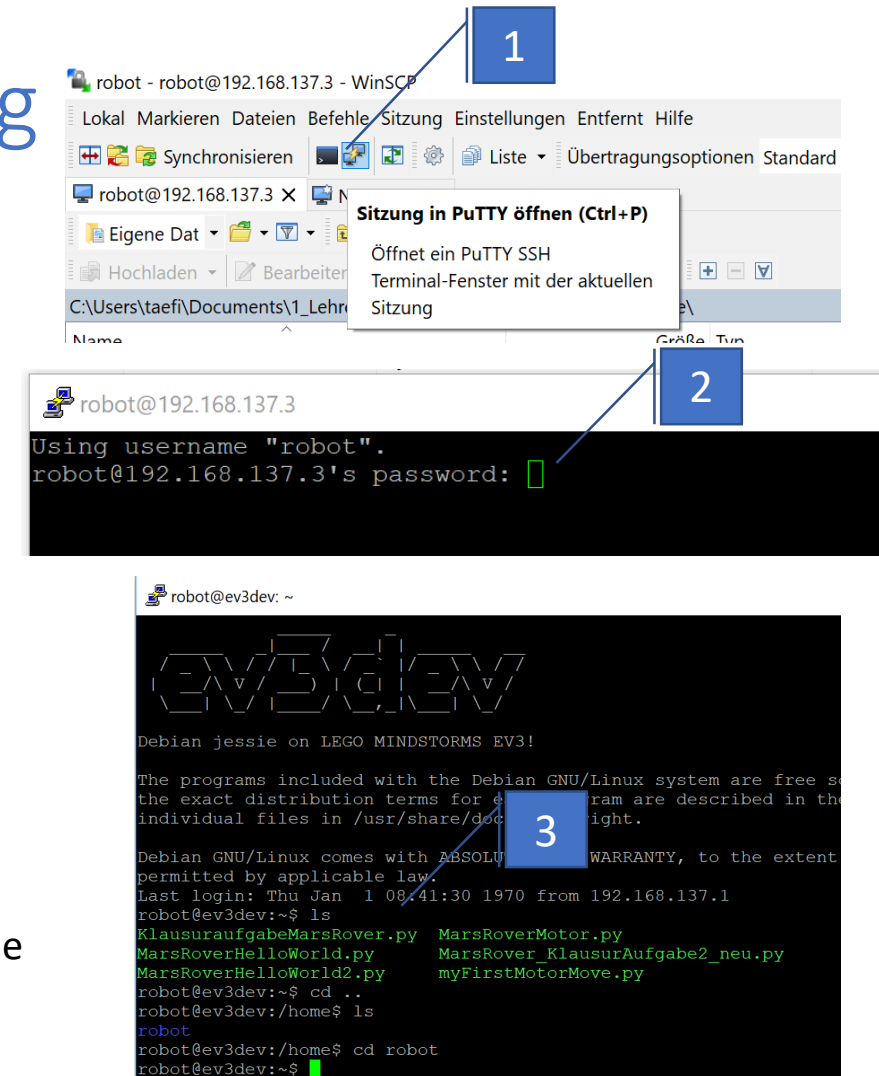
Executing programs via putty to read log

Starting the program via WinSCP – Putty

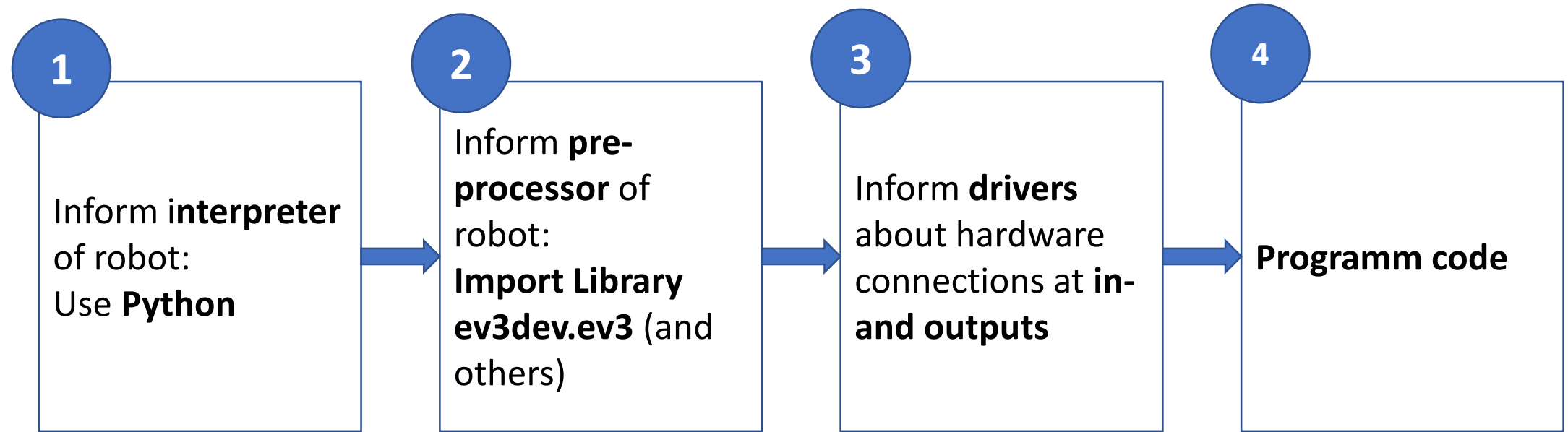
- 1 In WinSCP open a putty session
- 2 In putty enter the password (maker) and hit ENTER key
- 3 EV3dev starts up. Navigate to the directory home/robot
 1. “ls” (el-s) shows you the list of recent files/folders
 2. “cd ..” brings you one level up
 3. “cd <name of directory>” brings you into the directory
4. Start the program:
 1. robot@ev3dev:~\$ python3 my-file.py # will run your prog.
5. Other useful commands:
 1. robot@ev3dev:~\$ chmod +x my-file.py # make the script executable
 2. Force program to stop by typing Ctrl+C ad command line
 3. “Arrow up” revokes your latest command

Advantage: You are able to read error messages!

Task: Start the program MarsRoverMotor via WinSCP & Putty!



Structure of Python code to control the robot



The first two lines of a program

Your program must start with these two lines:

1

```
#!/usr/bin/env python3
```

2

```
import ev3dev.ev3 as ev3
```

! (Shebang) in the first line of the source code informs the interpreter on the brick to use Python3

The second line includes the library ev3dev.ev3 as “ev3”

Important note!

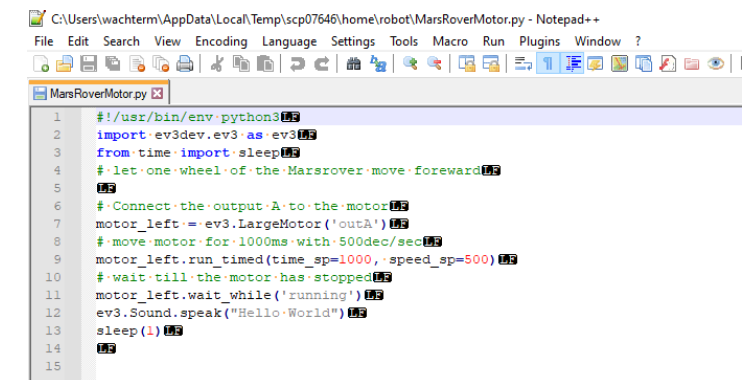
The operating system of the brick is a linux derivate.

Hence it can only read python source files that are written

- with the linux standard line endings (LF)
- Coded in UTF-8

This means:

- always start coding with a copy of the given program MarsRoverMotor.py
- If you are creating a new file, the editors need to format according to these requirements, **or your program will not run!**



```
1 #!/usr/bin/env python3
2 import ev3dev.ev3 as ev3
3 from time import sleep
4 # let one wheel of the Marsrover move forward
5
6 # Connect the output A to the motor
7 motor_left = ev3.LargeMotor('outA')
8 # move motor for 1000ms with 500dec/sec
9 motor_left.run_timed(time_sp=1000, speed_sp=500)
10 # wait till the motor has stopped
11 motor_left.wait_while('running')
12 ev3.Sound.speak("Hello World")
13 sleep(1)
14
15
```

Line endings e.g. in notepad++ (LF) – can be changed using menu „Edit-EOL conversion-Unix (LF)“

Contents: Python and software design

1. Connecting the robot with the lab computers
2. The EV3 actuators
3. The EV3 sensors
4. Microcontroller architecture
5. Testing and errors

3

Connecting actuators

In case that only one actor of a type is connected to an output, it is not necessary to specify the output port in the argument of the function:

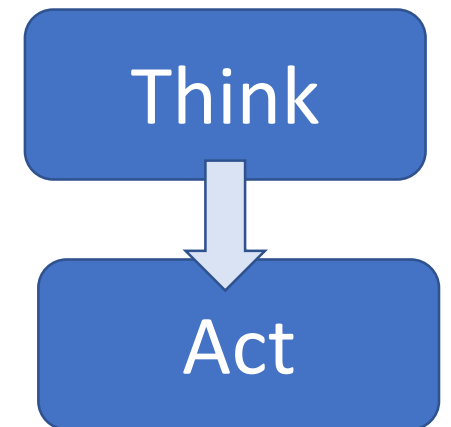
Example:

- `motor_left = ev3.LargeMotor()`

In case that two or more actors of the same type are connected to outputs, it is necessary to specify the output ports:

Example:

- `motor_left = ev3.LargeMotor('outA')`
- `motor_right = ev3.LargeMotor('outD')`



The EV3 Motors



Introduction to the EV3



Introduction to EV3 Presentation

Für Teilnehmer/innen verborgen



EV3 Python Library



Tasks

For the full list of parameters and functions please consult the [ev3 python documentation](#)



Examples motor functions

- `run_forever()`: the motor runs with speed defined in `speed_sp` until another command is sent to it.
- `run_timed()`: the motor runs for the time defined in register `time_sp`.
- `run_to_abs_pos()`: the motor runs until it reaches the absolute position defined in `position_sp`.
- `run_to_rel_pos()`: the motor runs until it reaches a position relative to the current position. The new position is `current position + position_sp` (in degrees – a value of 360 in `position_sp` represents one turn of the motor).

Examples for parameter that can be set

Motor parameters can either be set

- before executing a motor command (like a variable),
- or can be passed to the motor functions like parameters.
 - `stop_action = 'coast', 'brake', 'hold'`
 - `speed_sp = 500` (degrees/sec)
 - `time_sp = 1000`

```
# Connect the output A to the motor
motor_left = ev3.LargeMotor('outA')

#set a default speed of 100
motor_left.speed_sp = 100
# move motor for 1000ms
motor_left.run_timed(time_sp=1000)
```

```
# Connect the output A to the motor
motor_left = ev3.LargeMotor('outA')

# move motor for 1000ms
motor_left.run_timed(time_sp=1000, speed_sp = 100)
```

EV 3: Example code to run a motor

- 1 `#!/usr/bin/env python3`
- 2 `import ev3dev.ev3 as ev3`
- 3 `# assign Motor to output A`
`motor_left = ev3.LargeMotor('outA')`
- 4 `# Run the motor_left for 1000 ms`
`# with an angular speed of 500°/s, hence about 1.5 turns.`
`motor_left.run_timed(time_sp=1000, speed_sp=500)`

`# wait until the motor has stopped`
`motor_left.wait_while('running')`

`# say: Hello world!`
`ev3.Sound.speak("Hello world!")`

Summary: Run the test program

1. Load the test **test program** "MarsRoverMotor.py" from Relax to your PC
2. Transfer the program via WinSCP to the brick into the folder home/robot
3. Make the file executable
4. Connect a large motor with output A
5. Start the program from the brick or via Putty



```
#!/usr/bin/env python3
import ev3dev.ev3 as ev3

# assign Motor to output A
motor_left = ev3.LargeMotor('outA')

# Run the motor_left for 1000 ms
# with an angular speed of 500°/s, hence about 1.5 turns.
motor_left.run_timed(time_sp=1000, speed_sp=500)

# wait until the motor has stopped
motor_left.wait_while('running')

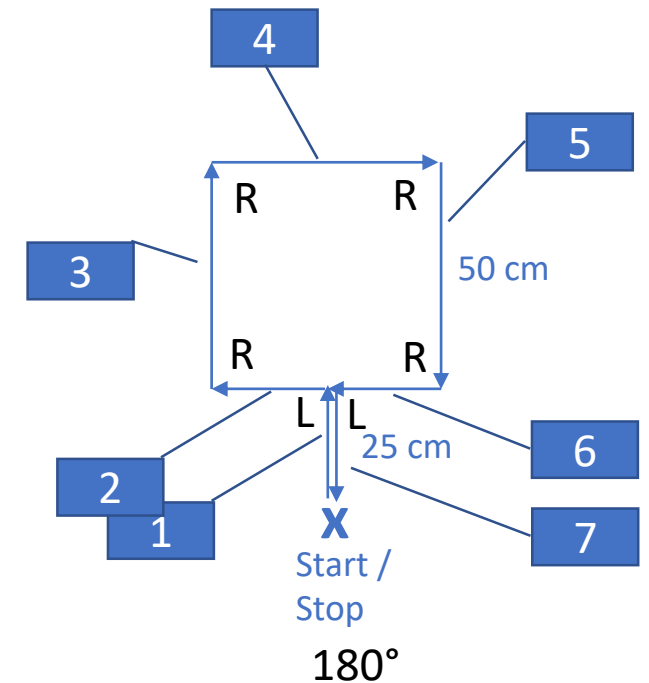
# say: Hello world!
ev3.Sound.speak("Hello world!")
```

Tasks: motor control

1. Connect both motors to the output ports and assign them in the program accordingly to make the robot move forward.
2. Adapt the program, so that the robot drives forward for a while, then backwards (use positive and negative speeds)
3. Make the robot turn 180° by spinning one motor forward and one backwards at the same time

Task: Drive a square

Let the robot drive a square according to the graphic on the right. When arriving at the stop point, let the robot finish with a 180° pivot turn, so that he stands exactly in the same position as he started.



Tips:

- Don't just start to program in an unorganized way. Plan your code with an activity diagram.
- Check in the documentation on motor commands that you do not know yet, but which will make the robot to drive the exact distance (File *python-ev3dev.pdf*)
- It is important to let the motors finish executing one command, before starting the next command. This is possible with a `motor_right.wait_while('running')`, for example.
- This page has good tutorials, tips and tricks (note, that we will work with Version 1): <https://sites.google.com/site/ev3devpython/>

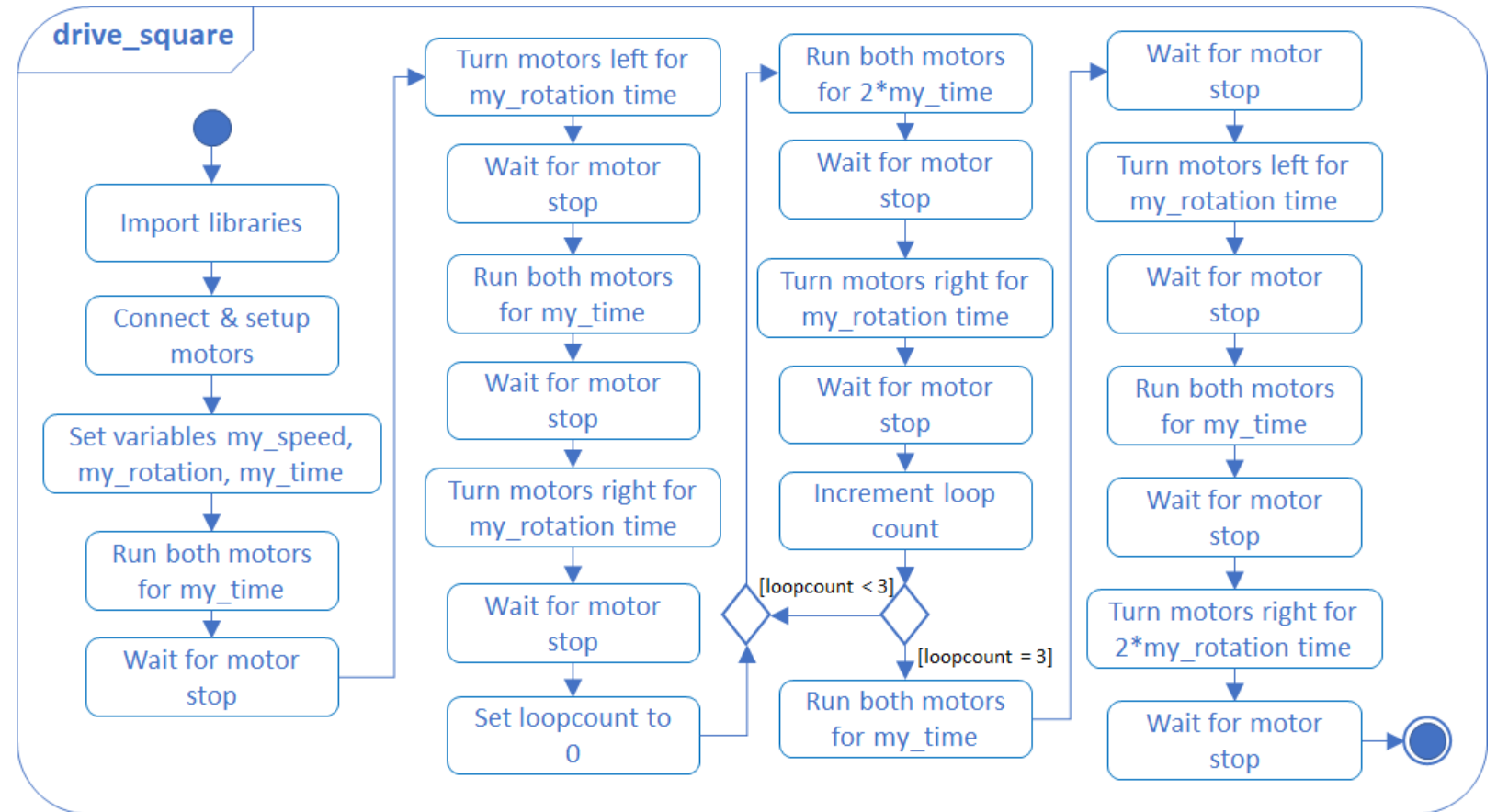
Square: one solution possibility

Idea:

- Using the known functions `motor_left.run_timed(time_sp=my_time, speed_sp=my_speed)`
- Find the values for the variables speed and time by trial and error for 25cm and 90° turn

Disadvantages: not precise...

- Due to construction
- Due to motors
- In case of changes in environment



Square: Ideas for improvement

Implement the following ideas line by line, to catch errors and problems

Ideas:

1. **Distance:** instead letting the motor run for a certain time, measure the rotations and calculate how many rotations are needed to drive a certain distance
2. **Turn:** instead letting the motor run for a certain time, use the gyro to detect when a turn is finished.
3. **Functions** for “Drive” and “Turn” to be defined, to be able to reuse the code for starting motors and waiting until the action is finished.
 - `def drive(m_left, m_right, length):`

Motor: Use distance instead of time

- To make the robot move a certain distance, the number of turns needs to be calculated.
- Therefor the diameter of the wheels needs to be known.
 - Either measure the diameter > Calculate the perimeter
 - Or read the diameter on the wheels (55x23 >> 55 mm)
- The robot moves for the calulated perimeter per turn of the wheel
- The argument of the function is in degrees
- The motor can be prevented from coasting, by using the brake command. `stop_action = brake` to improve the accuracy.

Example – turns the motor by one rotation (360°)

- `Motor.run_to_rel_pos(position_sp = 360, stop_action = 'brake')`

Contents: Python and software design

1. Connecting the robot with the lab computers
2. The EV3 actuators
3. The EV3 sensors
4. Microcontroller architecture
5. Testing and errors

Connecting sensors

In case only one type of a certain sensor is connected to the input ports, it is not necessary to specify the exact port.

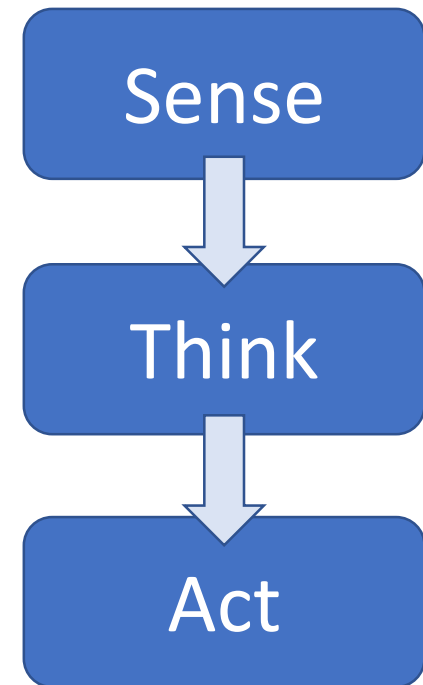
Example:

- `ts = ev3.TouchSensor()`
- `gy = ev3.GyroSensor()`

In case more than one sensor of a certain type (i.e. push button) is connected to the brick, the respective port numbers must be specified.

Example

- `ts1 = ev3.TouchSensor('in1')`
- `ts2 = ev3.TouchSensor('in2')`



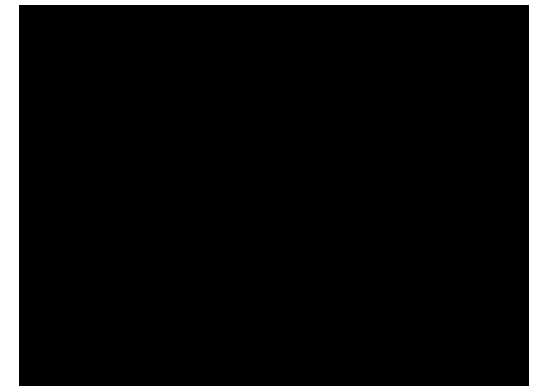
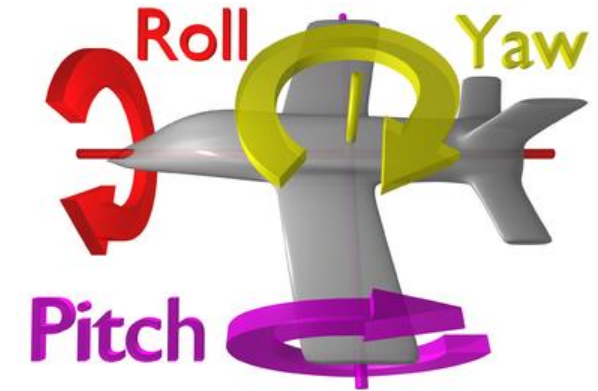
Gyroscope

Detects and measures the orientation or angular velocity in relation to a certain axis

Implementation:

- As a spinning disk that can move freely in a frame
- The spinning disk keeps its orientation, even if the frame moves, tilts or rotates. It conserves the angular momentum (Drehimpulserhaltung)
- Prof. von Bohnenberger from University Tübingen was the first to use a gyroscope for measurements
- Other operating principles: MEMS (Micro-Electro-Mechanical Systems). Further details on MEMS Gyros:

<https://www.st.com/en/mems-and-sensors/gyroscopes.html>



Video: http://www.globalspec.com/learnmore/sensors_transducers_detectors/tilt_sensing/gyroscopes

Gyroscope - Applications



Automotive/ Aviation / Space:

Measurement of
change in orientation



**As part of an Inertial
Measurement Unit
(IMU) = Gyroscopes
and acceleration
sensors in 3 axes**



Smartphone:
Navigation by dead
reckoning (koppeln), if
no GPS signal available,
i.e. in a tunnel



Segway:
Measurement of angle
to control the velocity

Interesting video about gyroscopes on the ISS (in German): <https://www.youtube.com/watch?v=8fnRUEsjpNc>

The EV3 gyroscope: connection

The EV3 gyro measures the rotation around one axis

- **Information how far the roboter has turned.**
Accuracy: $\pm 3^\circ$ for a 90° turn.
- **Information how fast the roboter tilts.**
Measureable angular velocity is maximally 440 degrees per second.

Connecting the gyro:

```
# connect the gyro sensor anywhere  
gy = ev3.GyroSensor()
```



The EV3 gyroscope - functions

1. **Measuring an angle in degrees:** how far did the robot turn?
 - Switch gyro to angular measurement mode: **gy.mode = 'GYRO-ANG'** (the robot must be still, during start-up and calibration phase).
 - **gy.value()** returns the current angle in degrees since roboter start-up.
 - **gy.value()** delivers positive values for a clockwise turn.
2. **Measuring the angular velocity per second:** how fast is the robot tilting?
 - Switch gyro to angular velocity mode: **gy.mode = 'GYRO-RATE'**
 - **gy.value()** returns the current angular velocity.

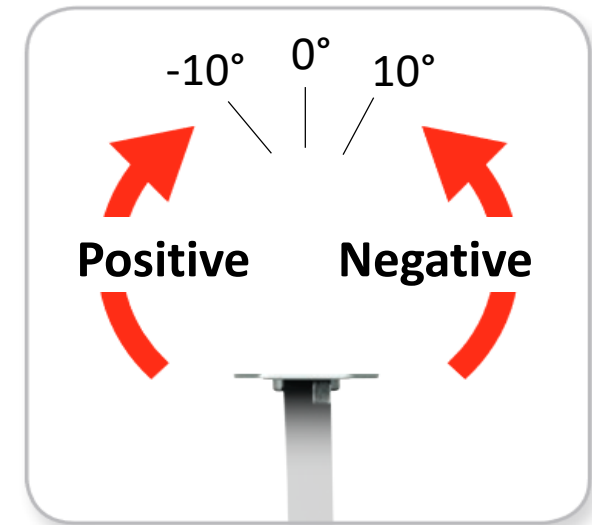
Tips

During program runtime the sensor can be recalibrated:

- switch the gy.modes back and forth or using recalibrate function.

If gyro (and other sensors) deliver irregular values, reset the sensor:

- disconnect and reconnect the sensor cable *after* starting the brick.



Rotation around a single axis

Build-in functions that help with debugging

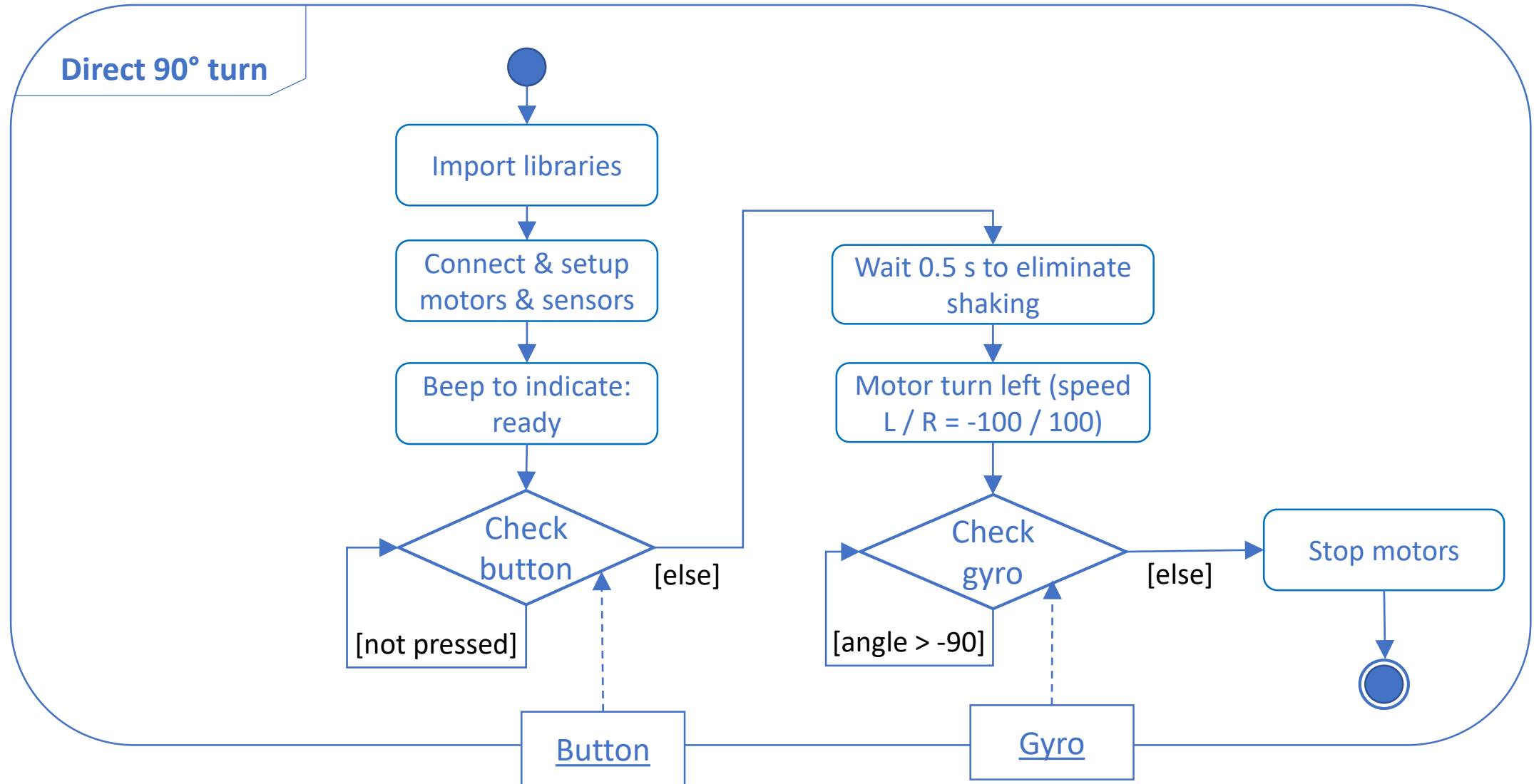
Use prints or beeps to find out in which part of the program the robot currently executes.

- `ev3.Sound.beep()`
- `print(gy.value())`

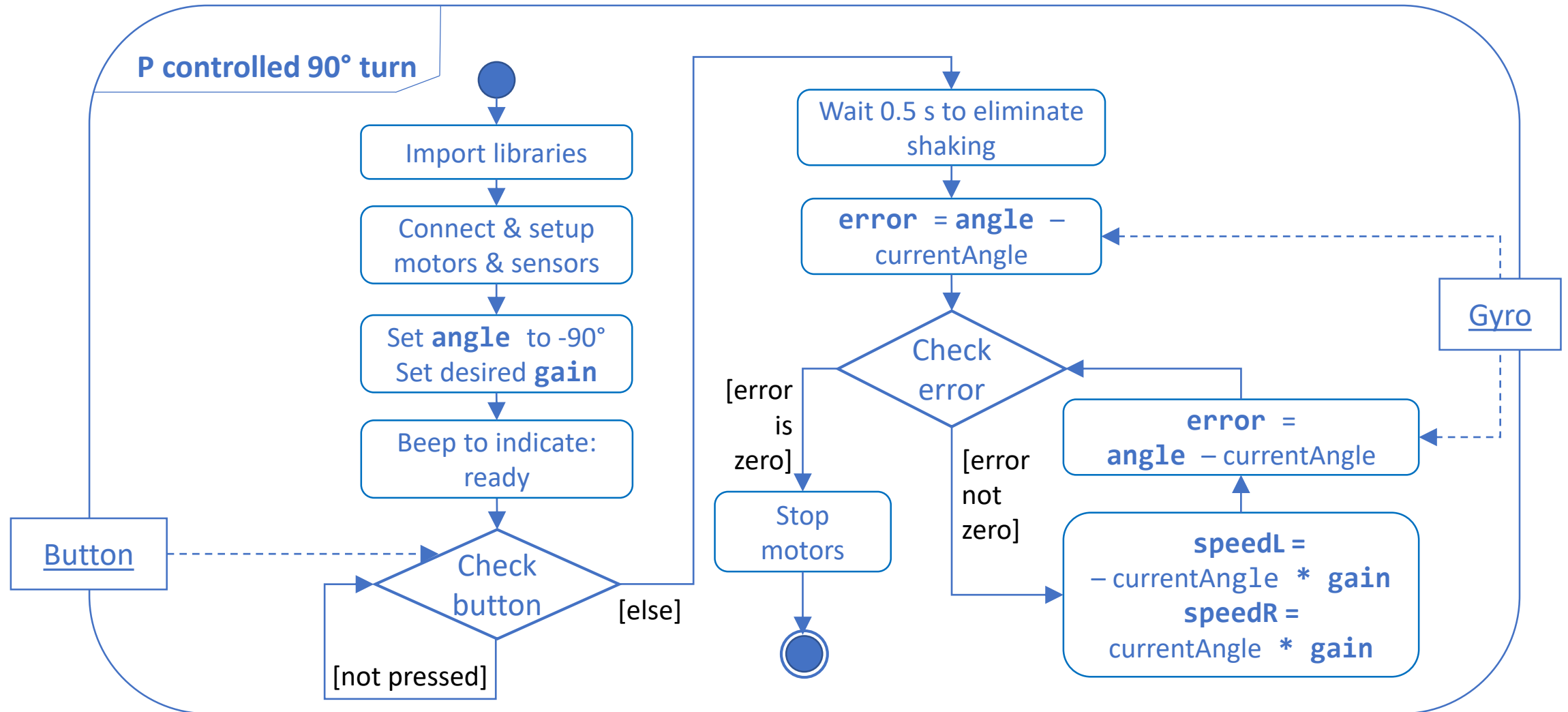
Example: This code will execute the while loop and beep annoyingly, as long as the angle is smaller than 90 degrees. Once outside the loop, the robot will say so.

```
while gy.value() < 90:
    ev3.Sound.beep()
ev3.Sound.speak("Outside loop")
```

Using gyroscope for left 90° turn with direct control

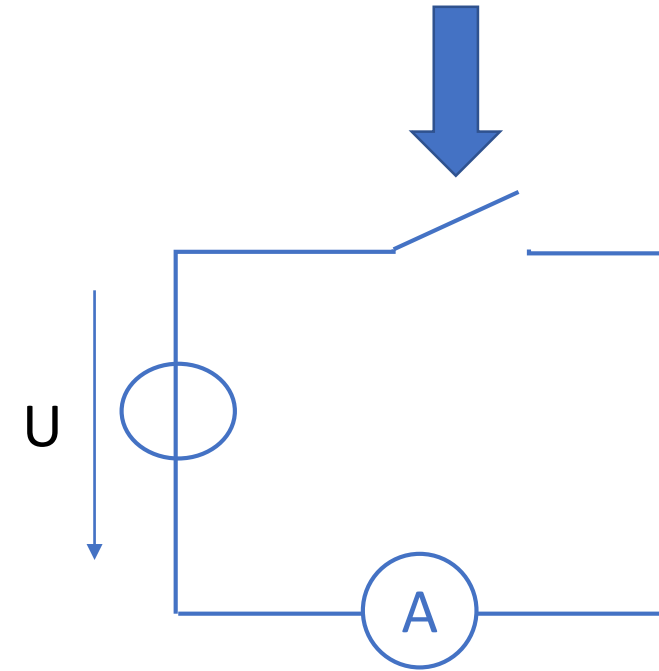


Using gyroscope for left turn 90° with P-control



Touch sensors

- **Touch sensors** can be implemented, i.e. with a simple switch that closes an electrical circuit when pressed down.
- Other touch sensors use the change in capacity to detect a touch (Example: Touch sensor on a ceran stove or smartphone screen)



The touch sensor of the EV3



The EV3 touch sensor is a simple, but precise device.

It indicates when the button is pressed or released.

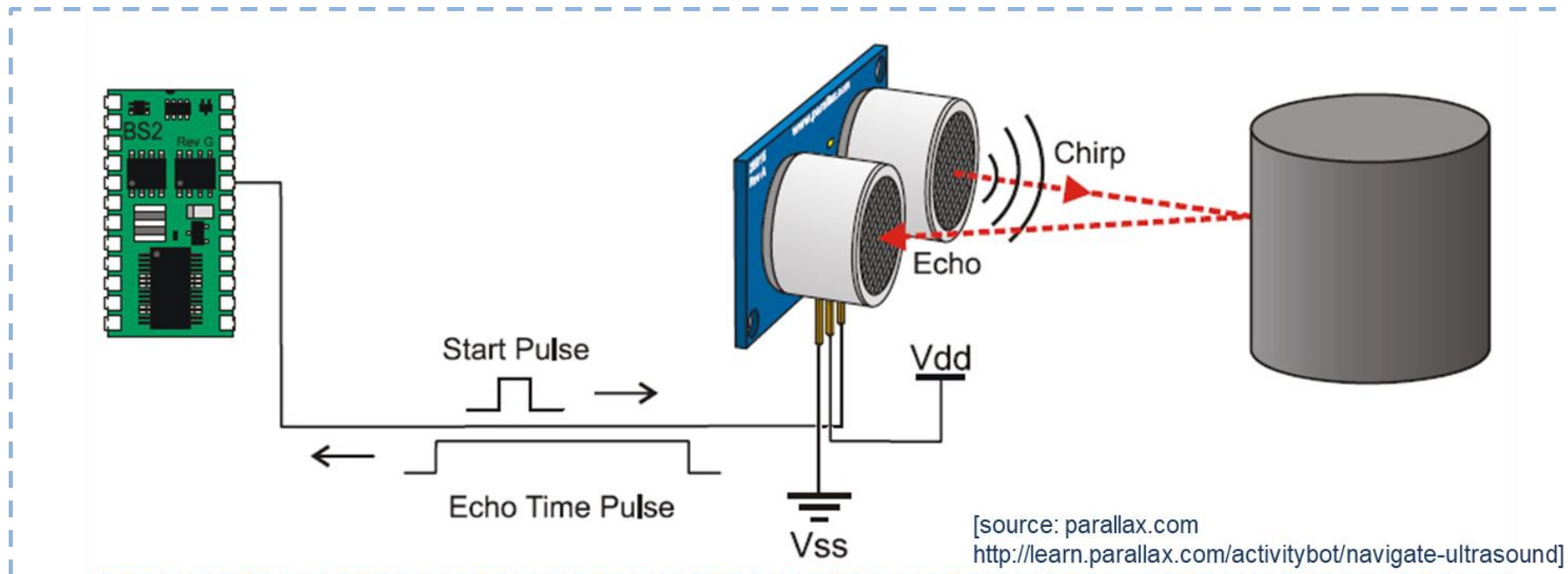
The function `value()` delivers the current status of the button.

Value	Description
0	Not pressed
1	Pressed

```
ts = ev3.TouchSensor() # connect a touch sensor anywhere
while not ts.value():   # loop if the button is not pressed
    sleep(0.01)         # wait for 0.01 seconds, then re-enter loop
```

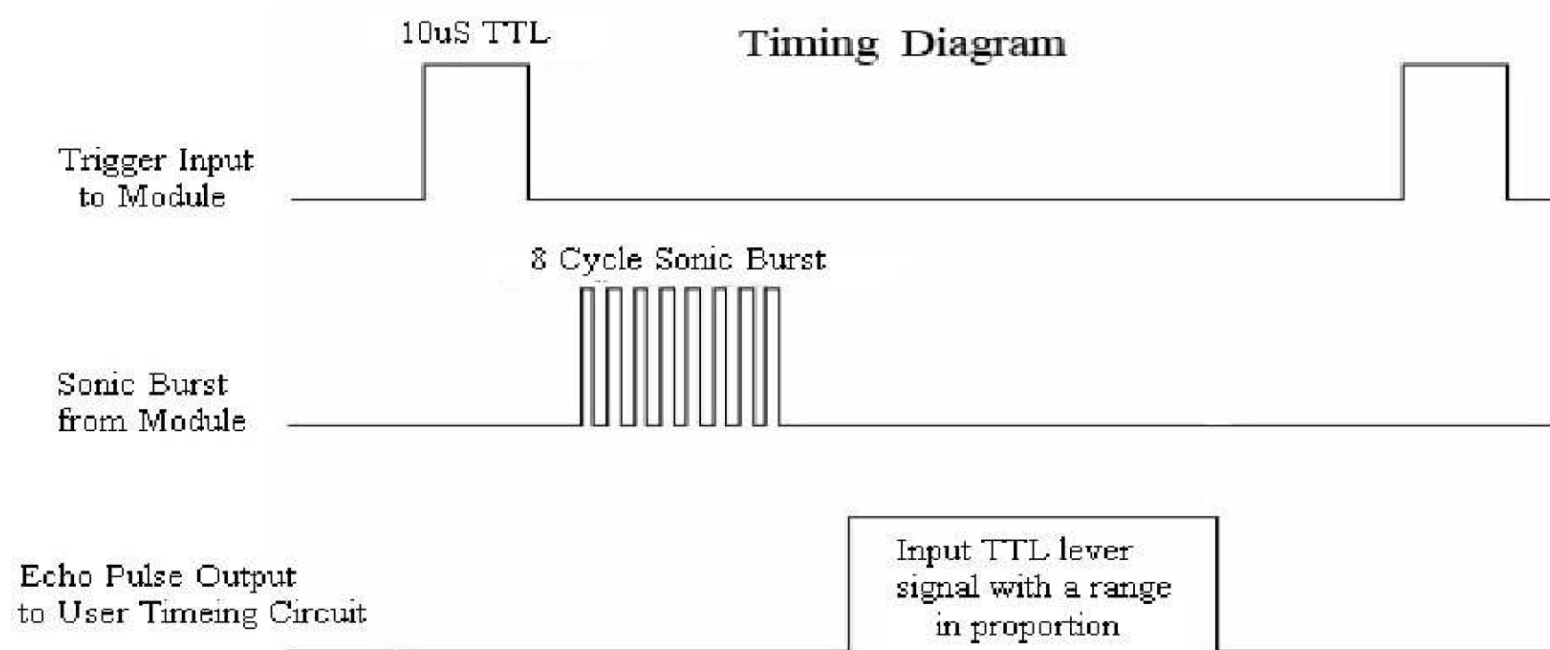
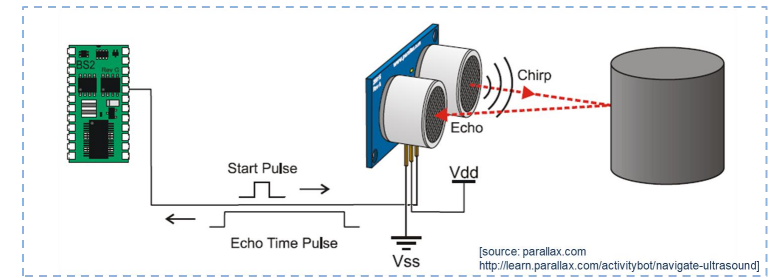
Ultra sonic sensors

- Ultra sonic sensors measure the time it takes for a sound impulse to travel from the sender to the object and back (echo time pulse).
- The duration of the echo time pulse is proportional to the distance from the object.

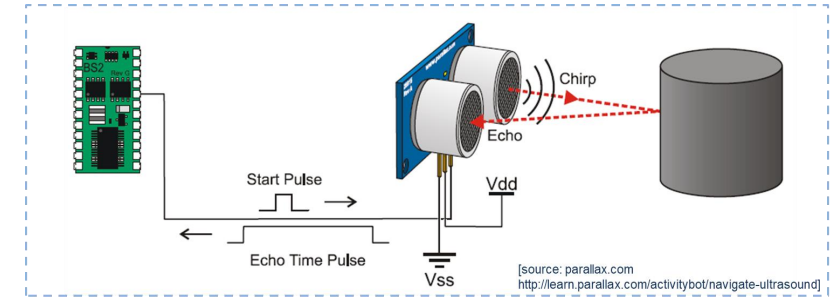


Ultra sonic sensor

Timing diagram from a data sheet



Exercise



- The duration of the echo time pulse is proportional to the distance from the object
- The speed of sound is 343 m/s
- How far is the distance between an object and the sensor, if the echo time pulse is 1 ms long?
- [Result: 17.5 cm]

The ultra sonic sensor of the EV3



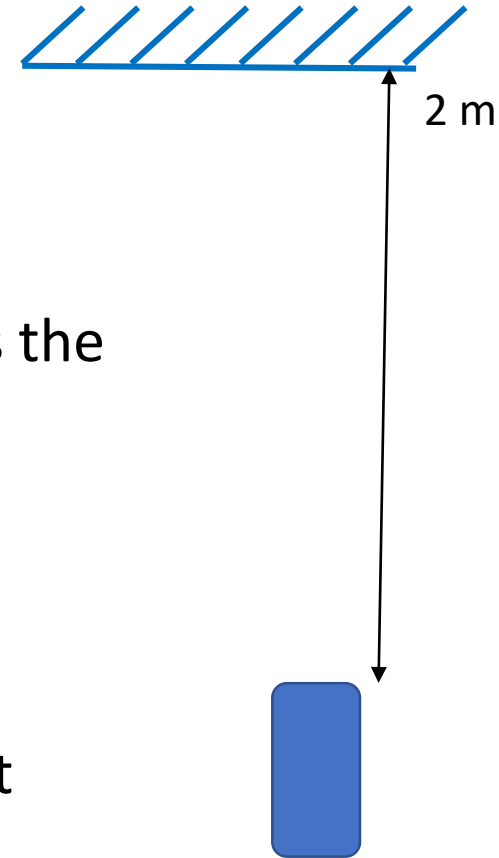
Connecting the ultra sonic sensor with the brick:
`us = ev3.UltrasonicSensor()`

The sensor has two modes:

1. Distance:
 - Switching to distance mode:
`us.mode='US-DIST-CM' # Put the US sensor into distance mode`
 - Wait for the distance to fall below a certain value:
`while us.value() > 150: # While the distance is smaller than 15.0cm
 sleep(0.01) # wait`
 - The measurable distance is between 3 and 250 cm (with a precision of +/- 1 cm)
 - The sensor might not work well, when mounted close to hard surfaces, such as the ground (due to interferences) or when it shall detect objects that do not reflect well (such as soft round or soft surfaces).
2. Identify other ultra sonic sensors
 - `us.mode = 'US-LISTEN'`
 - `other_sensor_present` is 1, if other sensors are detected

Race: NOT hitting the wall

- Position the roboter about 2 m in front of a wall, facing the wall.
- Upon button press, the robot should drive as fast as possible towards the wall.
- The goal is to stop as fast as possible in 1 cm distance to the wall (the robot should not touch the wall, but not further away than 1 cm).
- If a robot touches the wall, the race is lost.
- The team that can present a software design and develops the fastest robot wins.

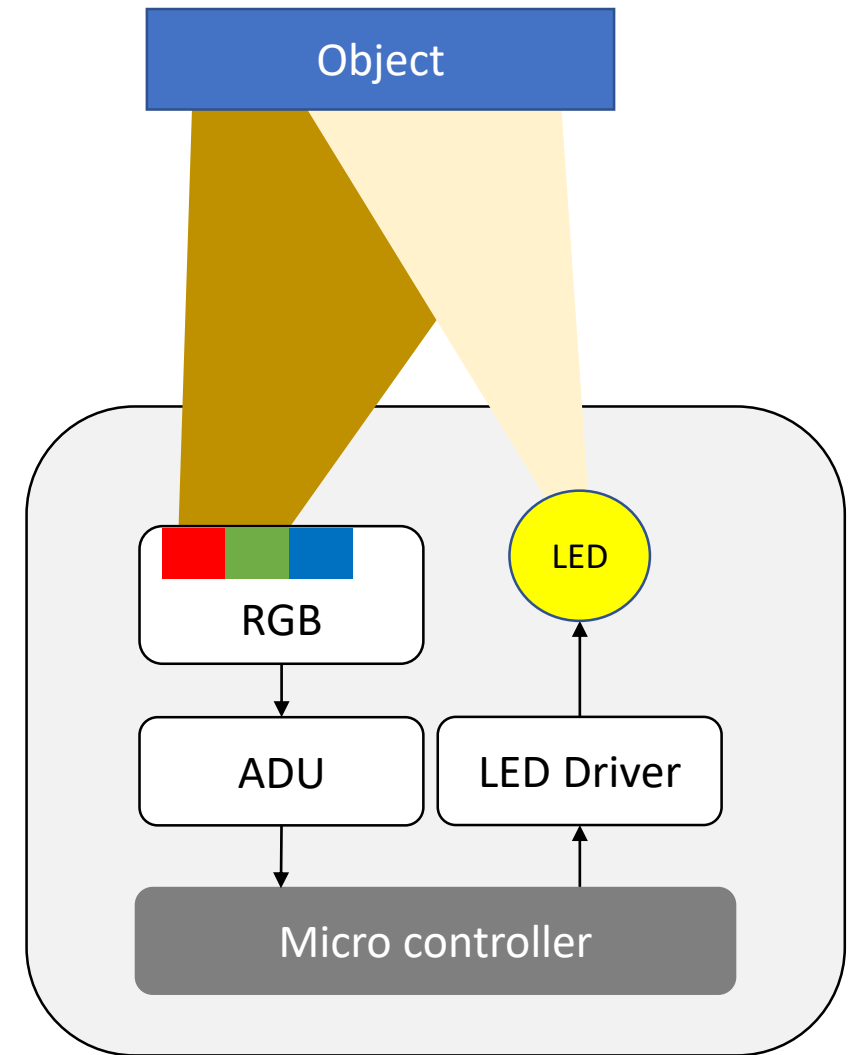


Color sensor

- A color sensor test a color against different known values
- A white LED shines on the object that shall be measured. The frequency and intensity of the reflected waves are evaluated.

Applications

- Control the presence of lids and caps
- Control the light intensity of LED lamps (lumen)
- Check the presence of pills in blister packs
- Quality check of paints and colors



The color sensor of the EV3

Digital sensor to detect the color or intensity of light, that enters through the little window at the front of the sensor. The sensor rate is 1kHz.

- Connecting the sensor: `cl = ev3.ColorSensor()`
- The sensor can be used in three modes
 1. **Color mode** (`cl.mode='COL-COLOR'`) The sensor detects seven colors: `cl.value()` = 0 to 7, see table on the right.
 2. **Reflect mode** (`cl.mode='COL-REFLECT'`) **The sensor measures the** intensity of the light. The light is emitted by a red led and reflected on an object. `cl.value()` returns a value between 0 (very dark) and 100 (very bright).
 3. **Ambient mode** (`cl.mode='COL-AMBIENT'`) The sensor measures the intensity of ambient light. It delivers a `cl.value()` on a scale from 0 (very dark) until 100 (very bright).

The precision in reflect mode is highest, when the sensor is mounted parallel to the surface of the object, without touching it.



Color mode



Reflect mode



Ambient mode



Value returned by <code>cl.value()</code>	Meaning
0	No color
1	Black
2	Blue
3	Green
4	Yellow
5	Red
6	White
7	Brown

Optimized class ColorSensor2

On each brick there is an optimized class that can be used for the Color Sensor. It detects the colors more reliable. The sensor rate with this class is 100Hz.

- Import that class with `import ColorSensor2`
- Connecting the sensor: `cl = ColorSensor2.ColorSensor2()`
- The sensor can be used like the base class but with some extra functions:
 1. `getCalibratedColor()` returns the number of the color
 2. `getCalibratedColorString()` returns the name of the color

Important: Do not delete the files `ColorSensor2.py` and `cal_values.txt` from the brick! The Python file contains the new class `ColorSensor2` and the other file contains the calibration data for the color sensor in your box.

If you get another color sensor you have to create the calibration data new. Therefore delete or rename `cal_values.txt` and run your program that uses `ColorSensor2`. You will be prompted once to hold the sensor over a white surface and press ENTER. Follow these instructions and `cal_values.txt` will be created new.

You can also start the recalibration at any time with function `recalibrate_sensor()`

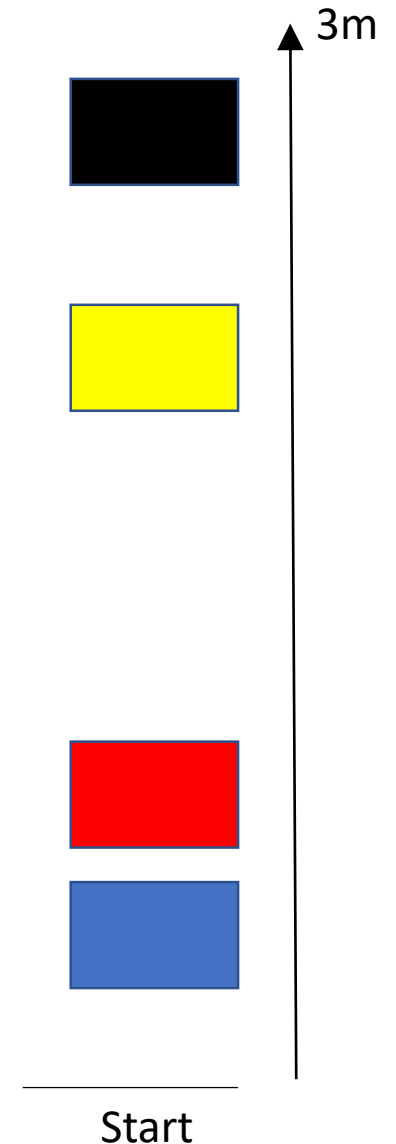


Value returned by cl.value()	Meaning
0	No color
1	Black
2	Blue
3	Green
4	Yellow
5	Red
6	White
7	Brown

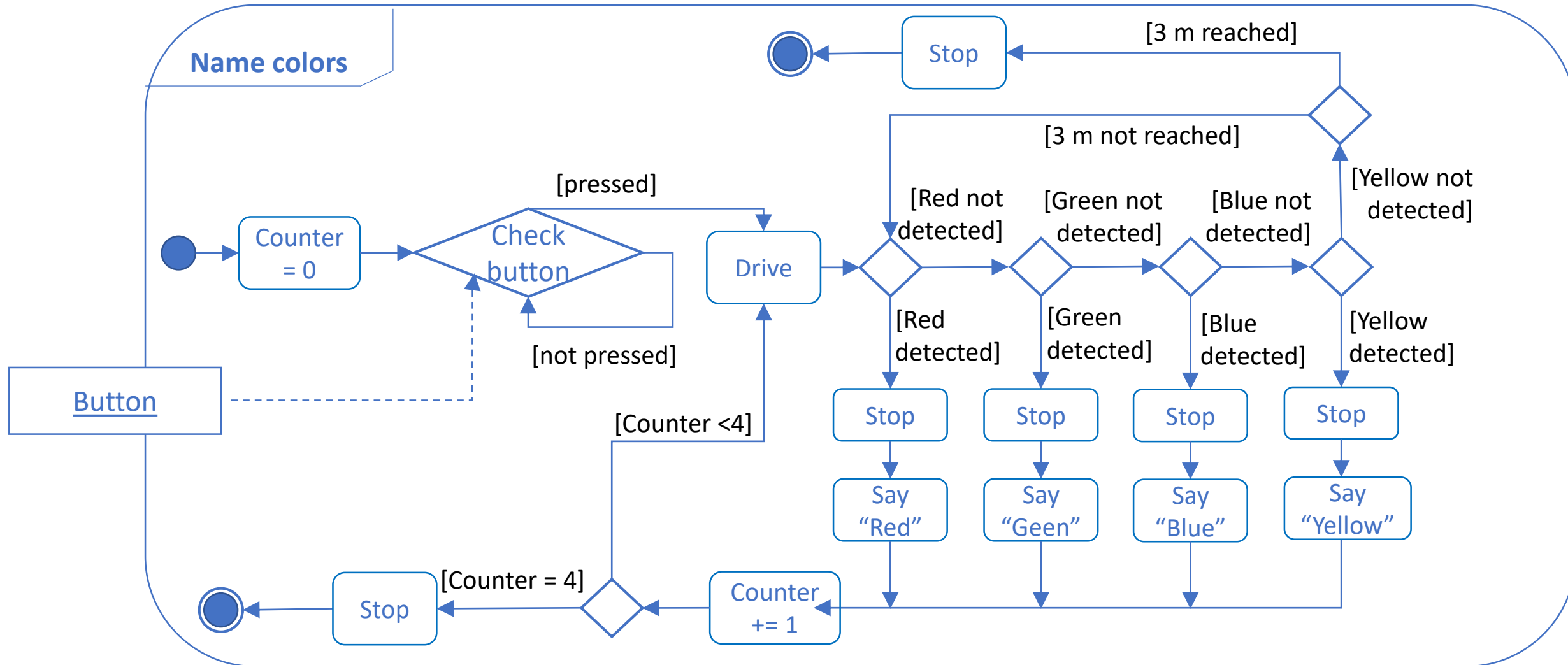
Exercise Color Sensor

- Distribute randomly four colored sheets of paper (20x30 cm each, red, blue, yellow, black) on a straight line of 3 m on the ground.
- The Mars Rover is positioned in front of the first paper. Upon power up it should drive 3 m straight forward (so that it will drive over the colored sheets).
- When identifying a color, the Mars Rover shall stop and name the color.
- On the fourth sheet, the rover shall stop.
- If no sheet is detected, the robot shall stop after 3 m automatically.

Design an activity diagram and implement the task on the robot.



Suggestion Activity Diagram: Exercise Color Sensor



Contents: Python and software design

1. Connecting the robot with the lab computers
2. The EV3 actuators
3. The EV3 sensors
4. Microcontroller architecture
5. Testing and errors

Numbering Systems

Digital numbers are often not only displayed in the decimal format.

After this block you should be able to:

- Understand the components of microcontrollers (μ C) and microprozessors (μ C, CPU)
- Read and calculate with numbers in the dual- and hexadecimal format

Definitions

- **Microprocessor (μ P)**

Central processing unit (CPU) with control unit, data processor, registers and data bus interface; integrated on a single chip.

- **Microcontroller (μ C or MCU)**

Microprocessor plus additional peripheral modules integrated on a single chip, for example: Input/Output (I/O) Module, Timer, Memory, Bus interface, etc.

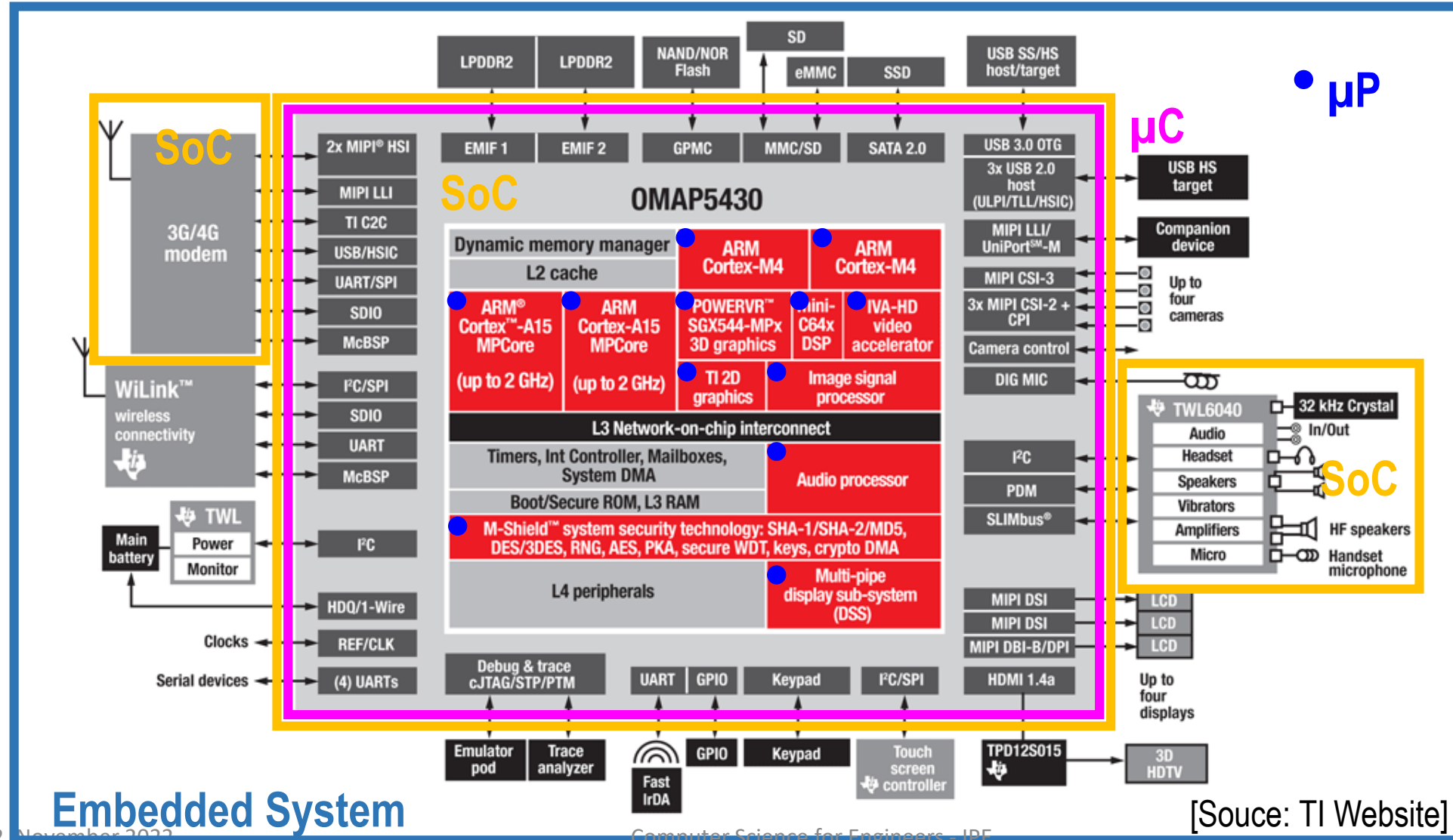
- **System-on-a-chip (SoC)**

Integration of several components, such as microcontrollers, (programmable) digital and/or analog components (converter, oscillator, sensors) on a single chip (“monolithic Integration”)

- **Embedded System**

Electronic system that is boxed in a housing and has a specific purpose. The electronic components are hidden (“embedded”). Example: Different SoC and other peripheral components on a chip, often with high requirements regarding real time capabilities and price

Example: “Embedded System” Solution from Texas Instruments for Mobile Phones (TI OMAP5430 SoC)



Application Examples

Data processing

- Vending machines
- Intelligent Cameras

Communication

- Networks, IoT, Smart City
- (Mobile) phones
- Wearables

Entertainment

- Digital cameras
- Game consoles & -equipment

Automotive

- Infotainment
- Safety and Driver Assistance

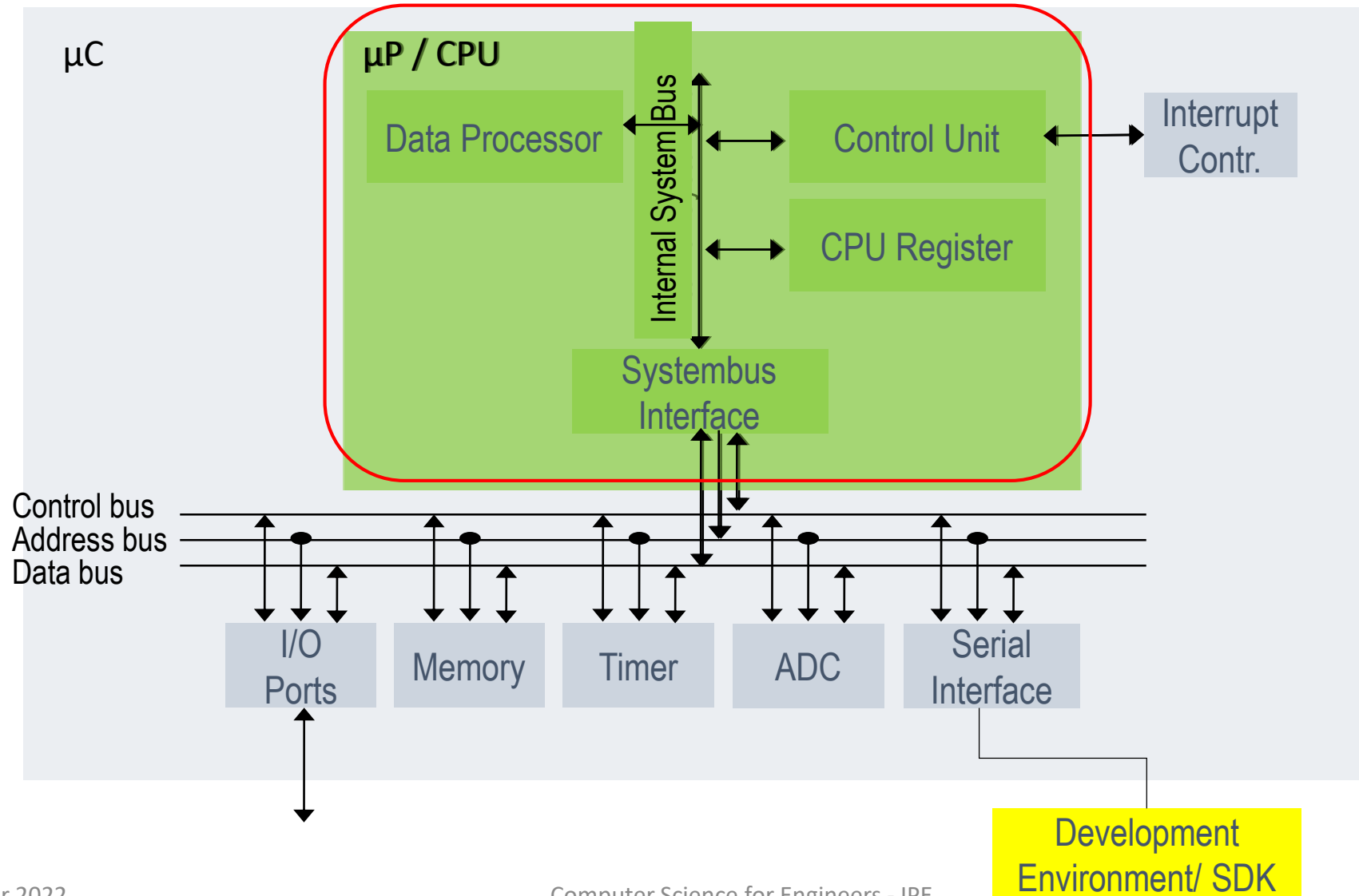
Industry

- Medical equipment
- Automation / Robotics etc.



Central part in nearly every modern technical product

In-general block diagram of a micro controller

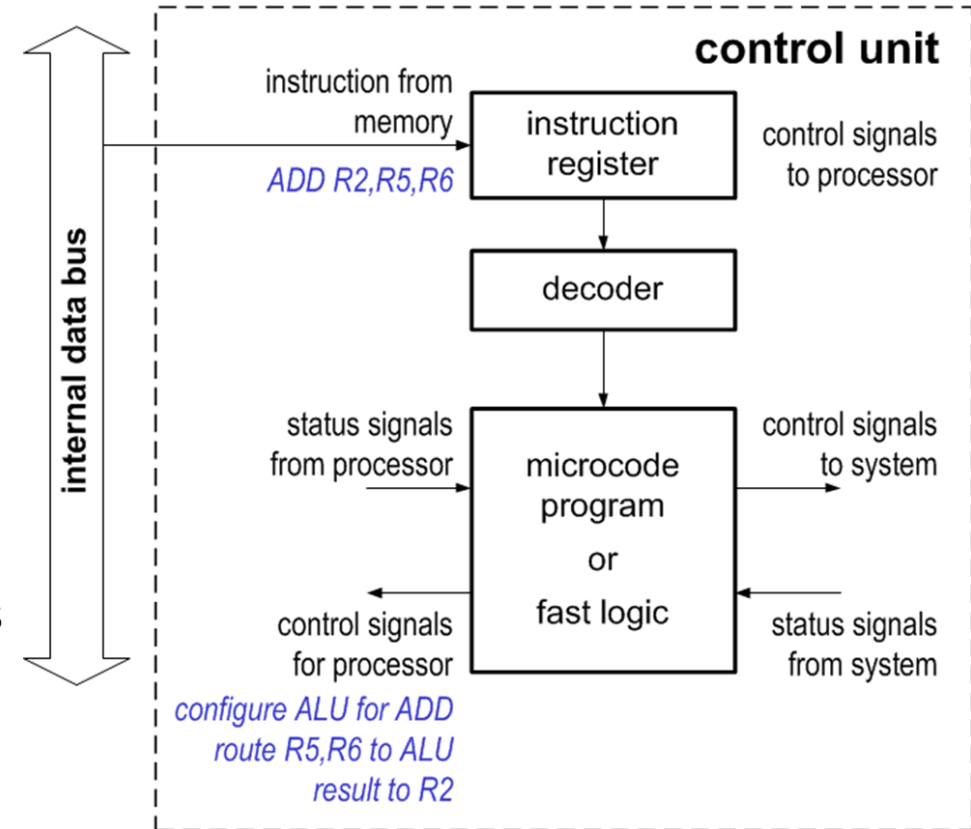


CPU: Control unit

- Reads and carries out the program code
- Controls the internal processes in the CPU
- Ensures the communication between CPU, memory and other peripherals

Typical phases when carrying out a line of code

1. **FETCH:** Load the opcode from the memory into the instruction register of the CPU
2. **DECODE:** interpret (analyze) the opcode
3. **EXECUTE:** produce control signals for internal CPU processes (f.e. configure the ALU, connect the registers containing the operands)
4. **WRITEBACK:** Write the results of the calculation into memory or register.



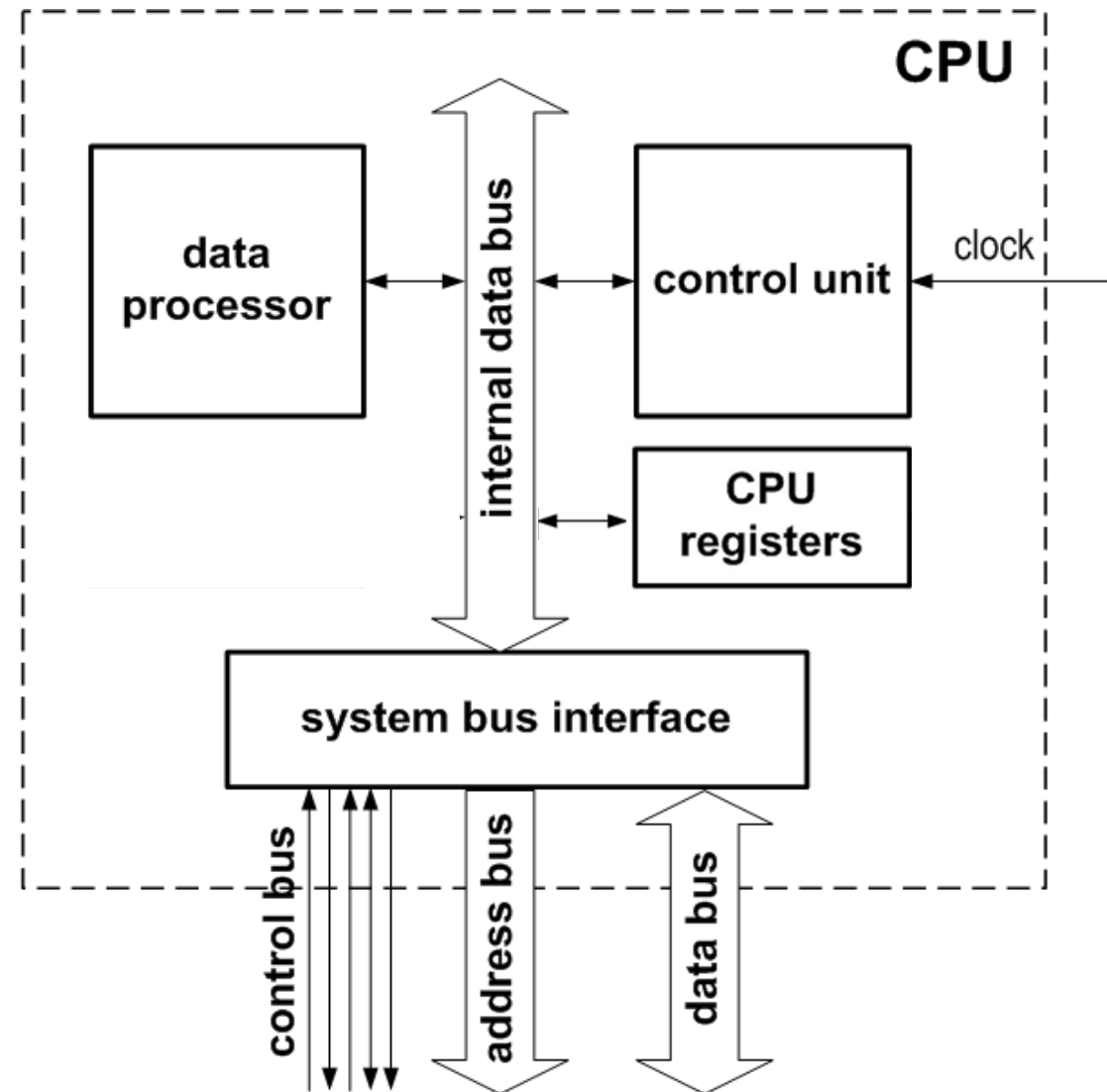
Central Processing Unit (CPU)

The Central Processing Unit (CPU) is

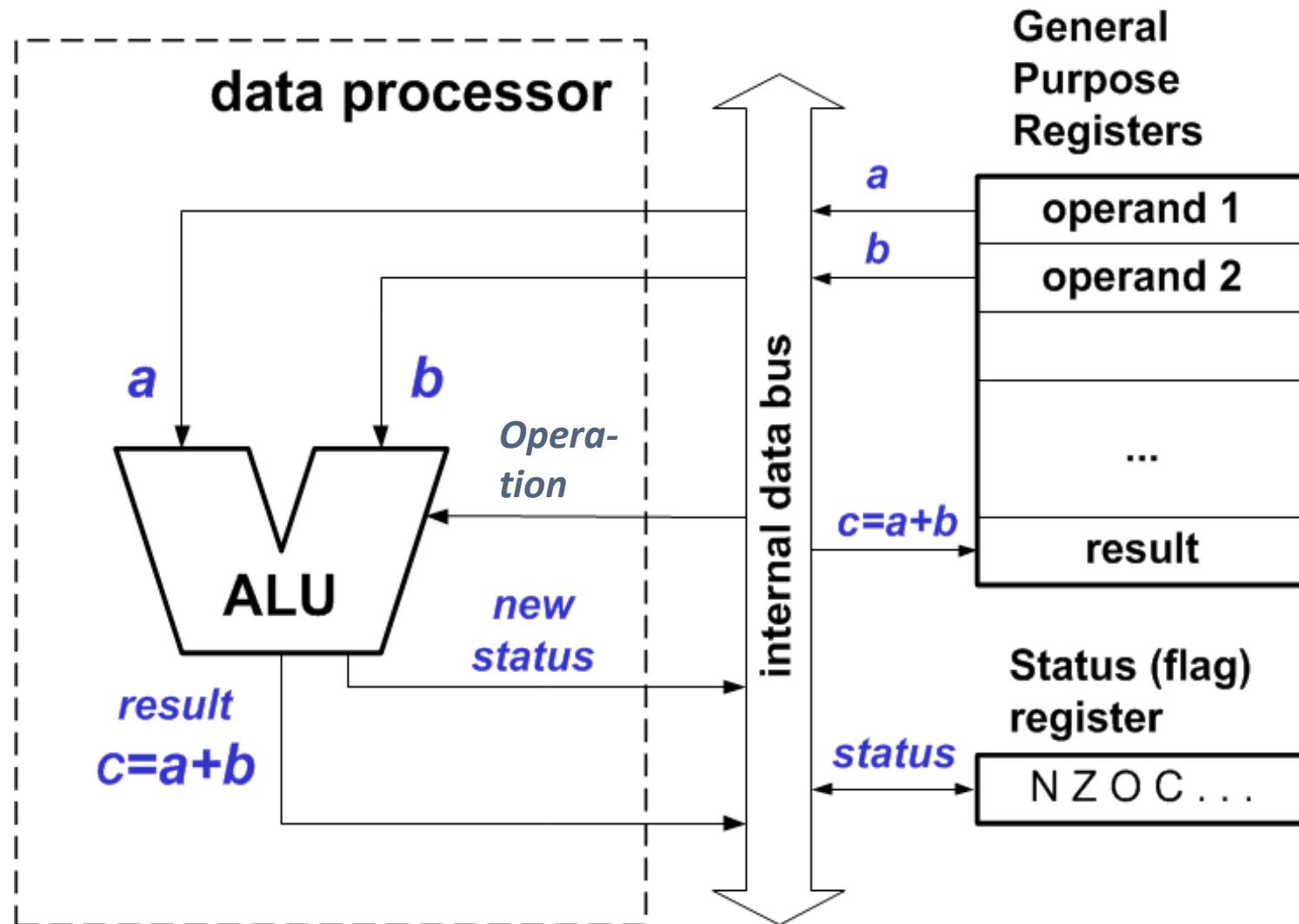
- The main unit to process and control the data of the μC ,
- Is the master of the system, which controls the digital blocks of a μC

Components der of the CPU:

- Data Processor
- Control Unit
- CPU register
- System bus interface

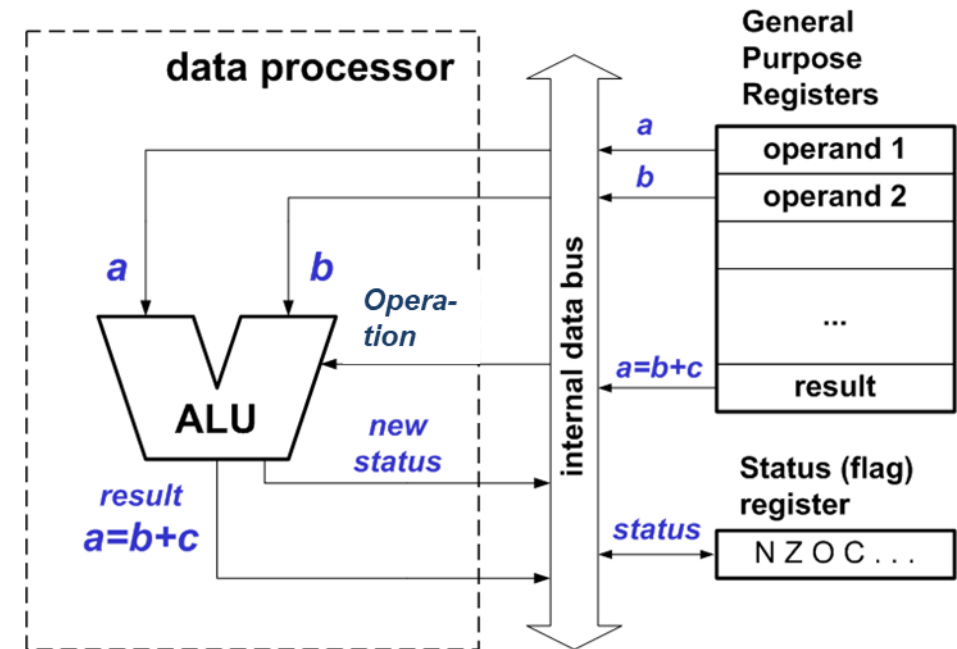


CPU: data processor



CPU: data processor

1. Arithmetic Logic Unit, ALU carries out
 - mathematical (+, -, *, ...)
 - logical (&, |, ^, ...)operations.
2. Two operands
 - Are loaded into the general purpose registers of the CPU
 - In some architectures also directly from memory (via caches)
3. The result of an operation is stored in the general purpose register.
4. The status of the operation is indicated in the status (or flag) register.
States are i.e.
 1. z=1: Result is zero;
 2. n=1: Result is negative;
 3. v=1: Result of signed operation cannot be presented properly (overflowed into sign bit)
 4. c=1: Result of unsigned operation cannot be presented properly.



Complement on two with 3 bit number

Unsigned:

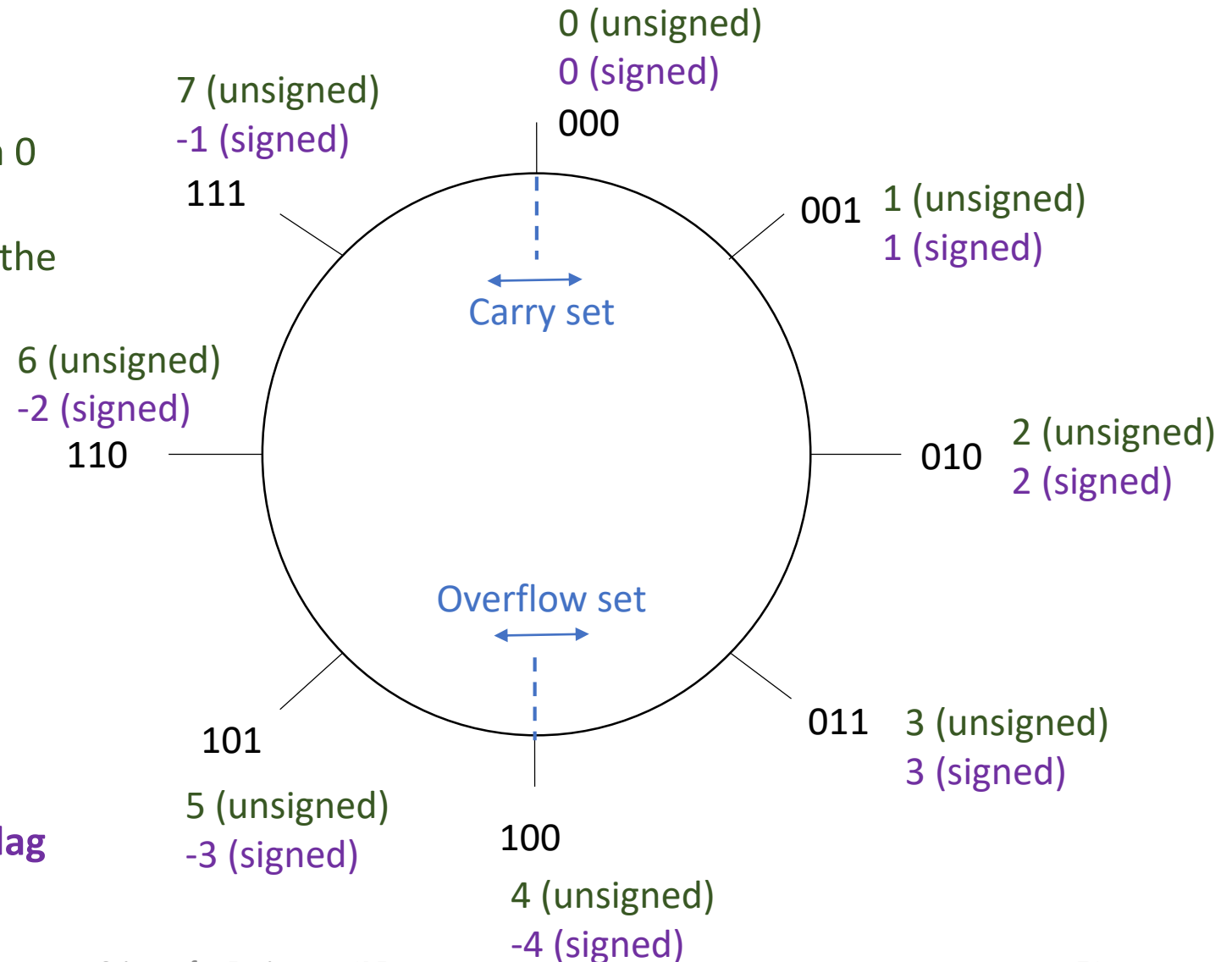
- 3 bits can represent $2^3 = 8$ values (from 0 to 7)
- Adding 7+1 to an unsigned number sets the carry flag:

$$\begin{array}{r} 111 \quad (7) \\ + 001 \quad (1) \\ \hline 1\ 000 \quad (0) \end{array} \Rightarrow 000 \text{ plus } \textbf{carry flag}$$

Signed:

- 3 bits can represent $2^3 = 8$ values (from 0 to 3 and -1 to -4)
- Adding -1 to -4 sets the overflow flag:

$$\begin{array}{r} 100 \quad (-4) \\ + 111 \quad (-1) \\ \hline 1\ 011 \quad (+3) \end{array} \Rightarrow 011 \text{ plus } \textbf{overflow flag}$$



CPU: Summary

Data Processor

- Arithmetic and logic operations with two operands
- Returns: The result of the operation and a status

Control Unit

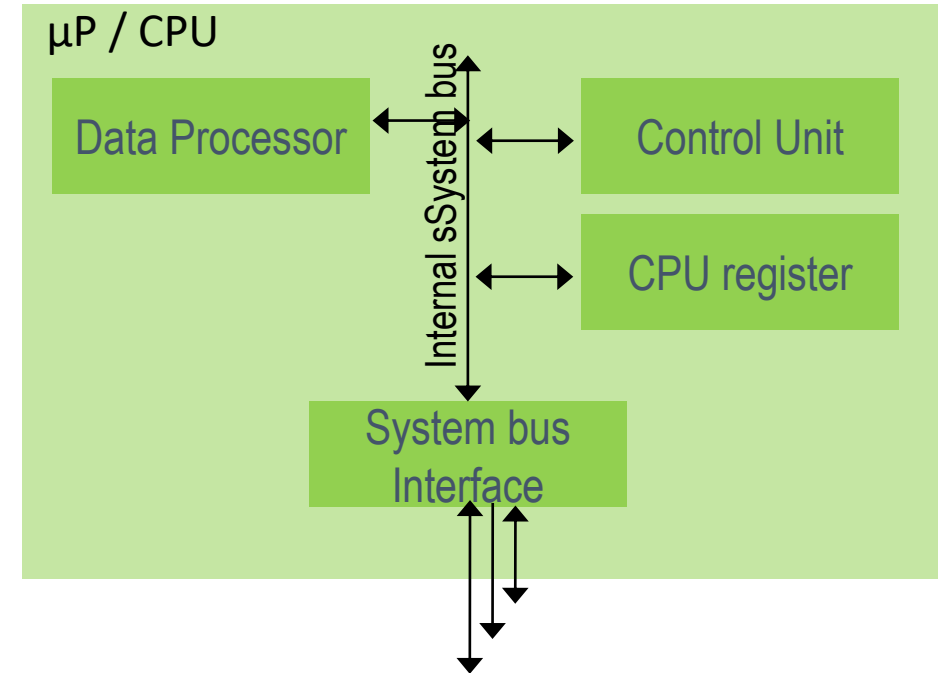
- Analyses the opcode and controls the processes in the CPU

CPU Register

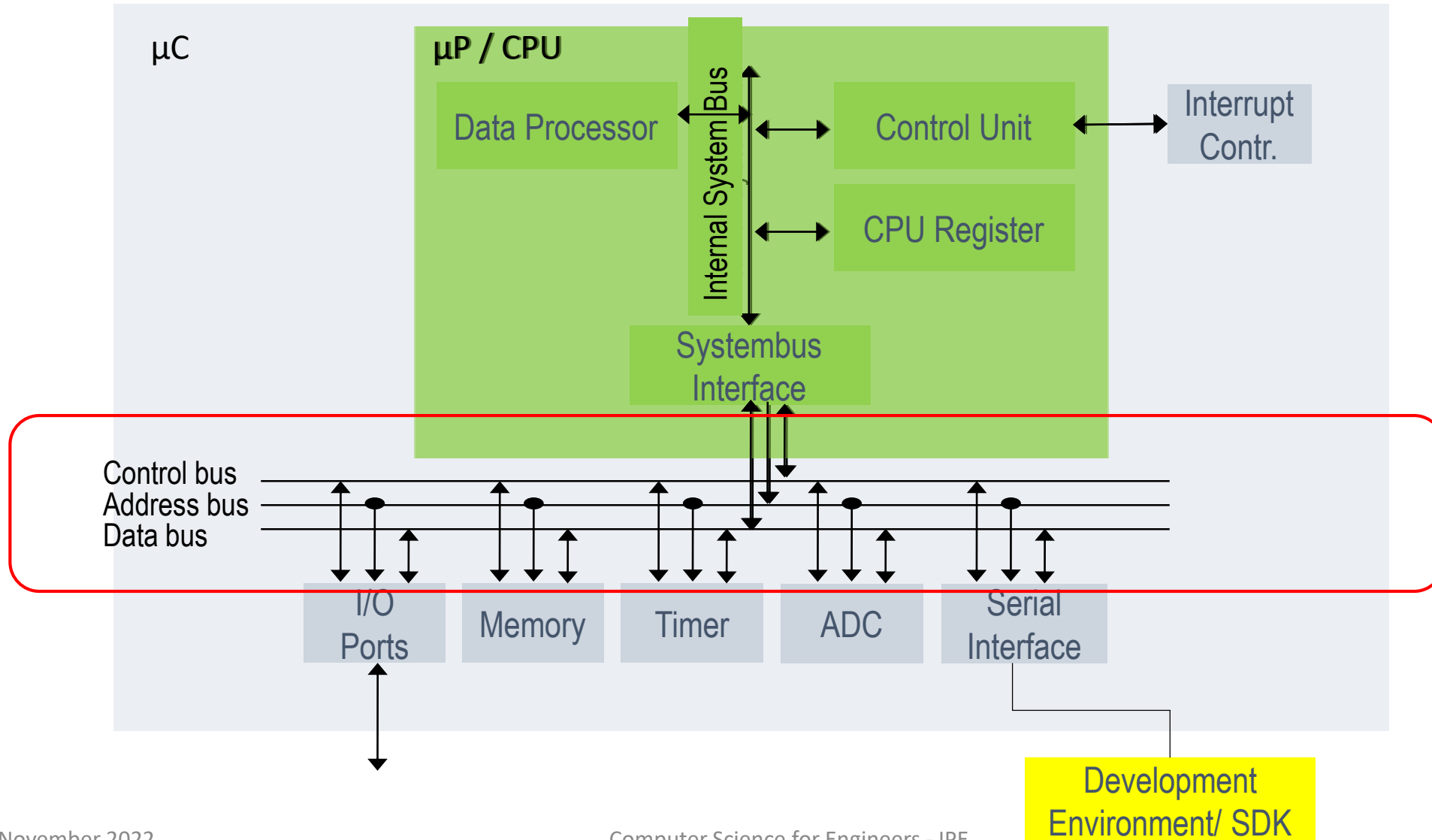
- Stores the operands and results
- Stores status of operation in status register (CPU flags)
- Pointer registers, point to the next operation (program counter, PC) or the the beginning of the stack (stack pointer, SP)

System Bus Interface

- Communicates with peripheral modules



In-general block diagram of a microcontroller

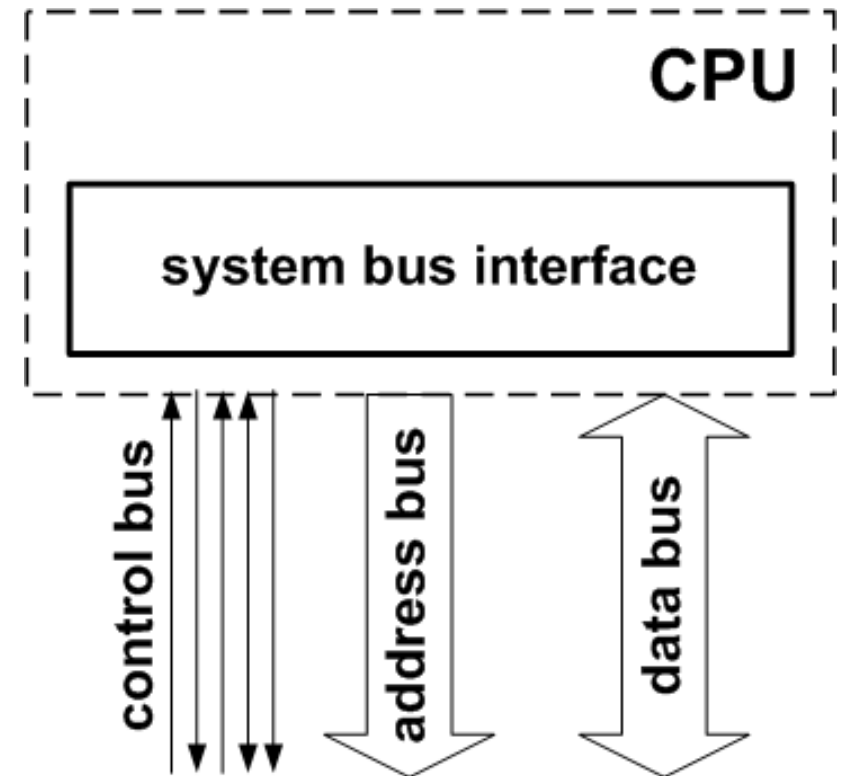


The system bus consist of three sub-busses

Data exchange

with memory (on-chip and μ C extern) and peripheral modules is initiated via the system bus interface with 3 sub bus systems

- **address bus**
(uni-directional, structured)
- **data bus**
(bi-directional, structured)
- **control bus**
(bi-directional, not structured)



The system bus consist of three sub-busses

Adress bus (uni-directional, structured)

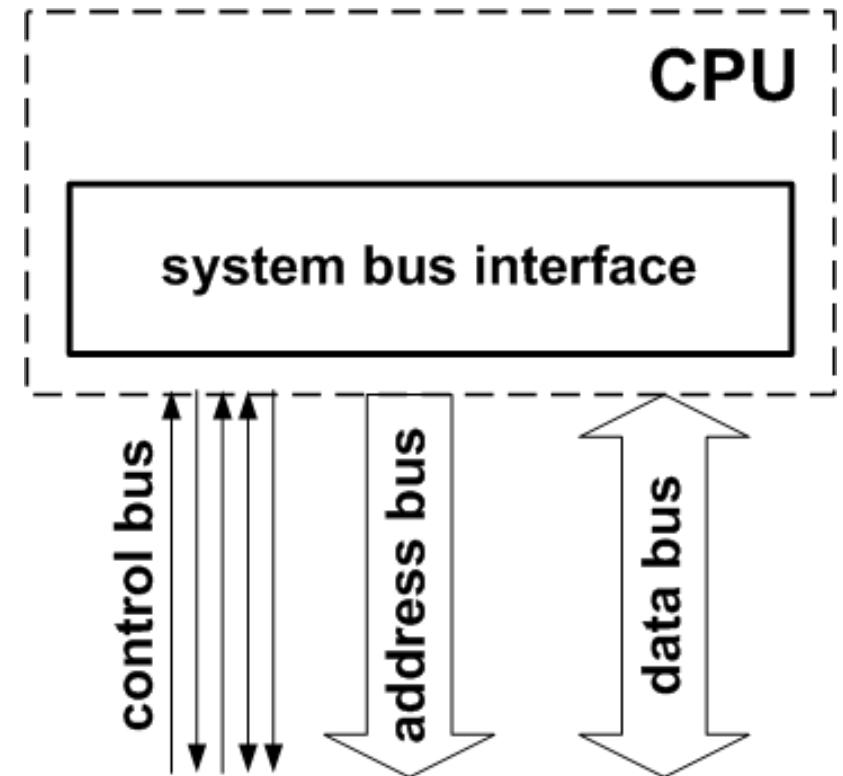
CPU as Master puts an adress (= bit pattern) on this bus. This identifies an unique „place“ in the system, which can provide or receive data (read / write). All slave modules (memory, peripherals) may read the bus.

Data bus (bi-directional, structured)

CPU and slave modules exchange data via this bus (f.e. instructions, variables, operands)

Control bus (bi-directional, unstructured)

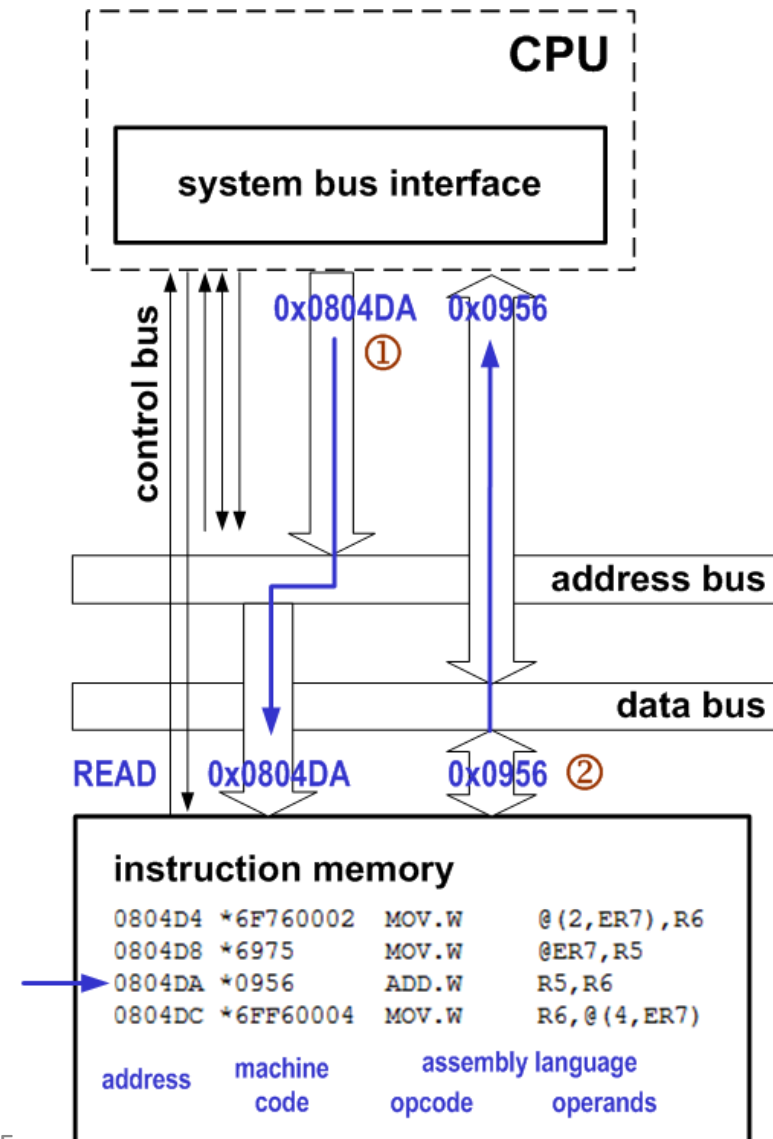
Unstructured bus with many single lines, that have a defined function. Example: define the direction of data transfer (read/write), stop an data transfer, mark a related data sequence (burst-transfer), etc.



Loading a command from memory into the CPU via system bus

1. The CPU puts the address of the next command on the **address bus**, f.e. 0x0804DA
2. Via the **control bus** the control indication ("READ") is requested from the memory
3. Memory reads the value of the requested address (on address 0x0804DA) and puts the value 0x0956 (= Instruction ADD.W R5,R6) to the data bus
4. CPU copies the value from the data bus to the instruction register

Python
code:
`c=a+b`



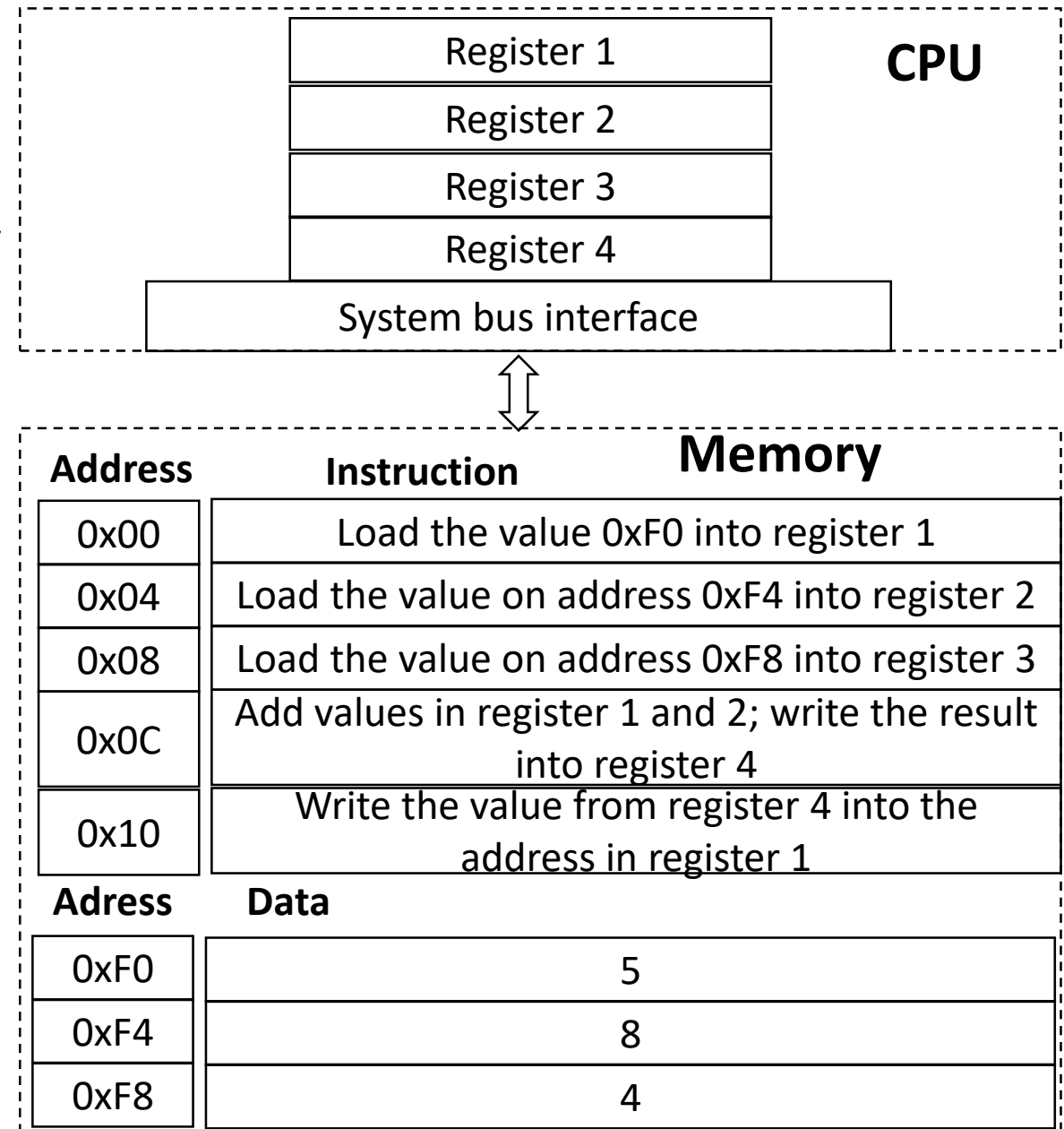
Exercise:

The pseudocode from instructions memory 0x00 to 0x10 are executed.

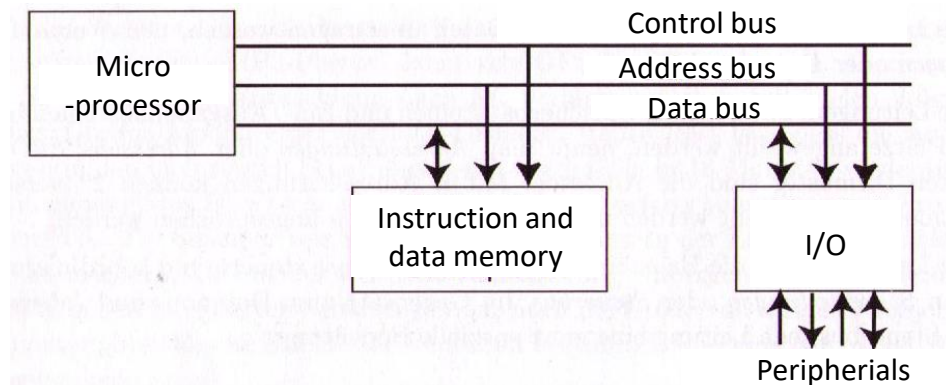
After the execution, which values are

1. In the CPU register 1
2. In the CPU register 2
3. In the CPU register 3
4. In the CPU register 4

1. In data memory 0xF0?
2. In data memory 0xF4?
3. In data memory 0xF8?

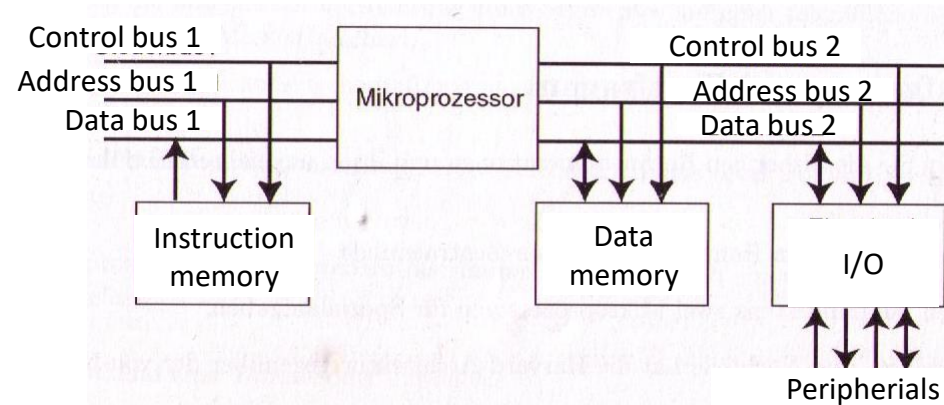


Von-Neumann Architecture



1. Data and program code **share a common memory**
2. Only a single **Data- and Address bus** is required
3. Example: PC, X86 Processors
4. **Advantage: Architecture is simple** (just one bus), and **flexible** (size of data & instruction memory)
5. **Disadvantage: Program execution is slower.** Instruction and data need to be read and written after each other. While data is written no new instruction can be read.

Harvard Architecture



1. **Separated Memory** for data and instruction code
2. **2 Data buses**, often 2 Address buses
3. Example: ARM Cortex M3, DSPs
4. **Advantage: increased speed** due to parallel reading/writing of data and instructions
5. **Disadvantage:** More hardware effort und and less flexible. The max. size of instruction code and data needs to be known in advance.

Contents: Python and software design

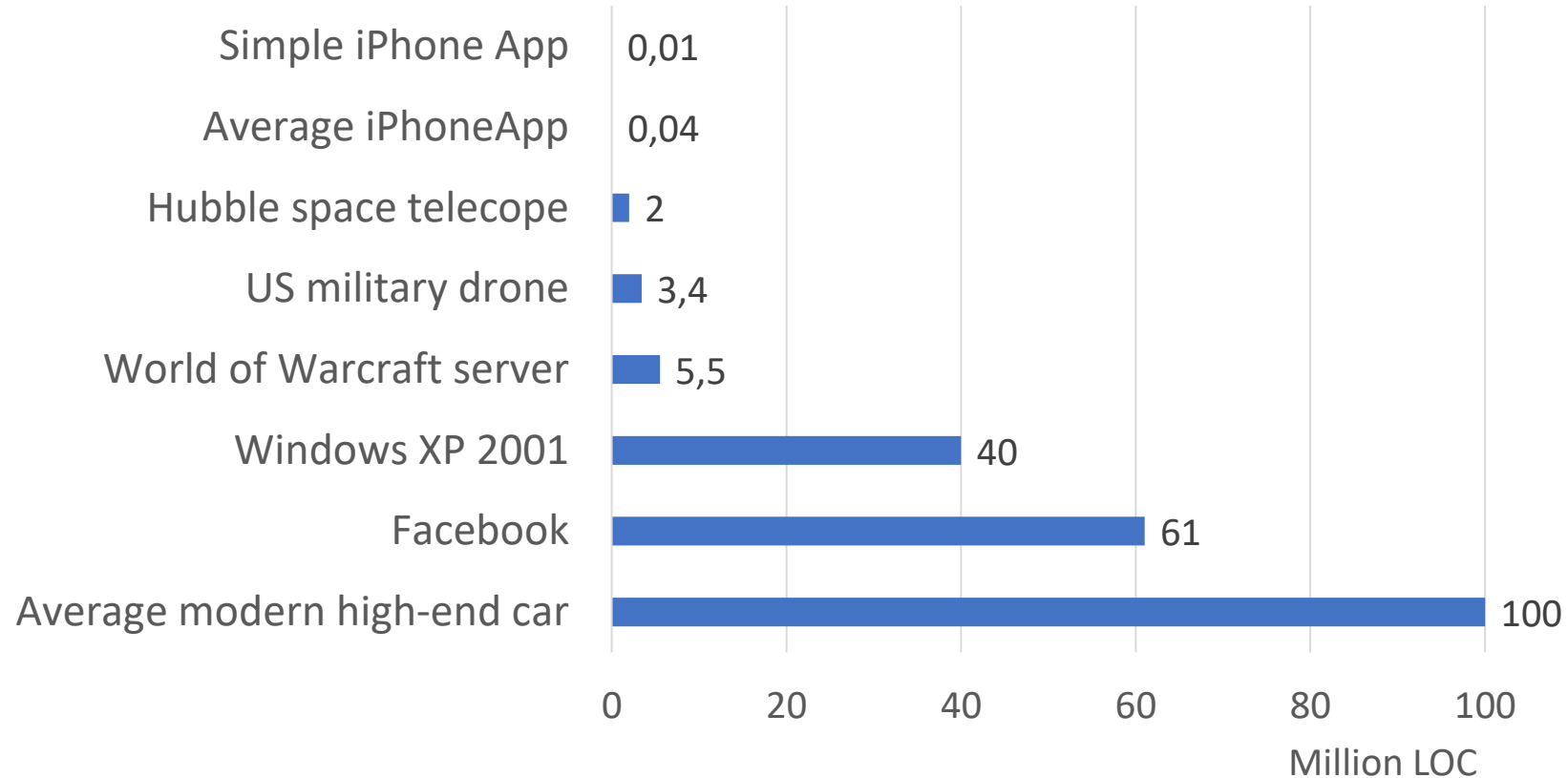
1. Connecting the robot with the lab computers
2. The EV3 actuators
3. The EV3 sensors
4. Microcontroller architecture
5. Testing and errors

Learning Goals

After this impuls you can

- Explain the difference between trial & error and testing of software
- Name the different ways to test and test types
- Classify errors into categories (error types)
- Determine and use equivalent classes for testing your software

Size of software based systems in Lines of Code (LOC)



⇒ Additionally the number of SW developers and the numbers of SW users increase

Source: <https://www.visualcapitalist.com/millions-lines-of-code/>

Errare humanum est

Why test?

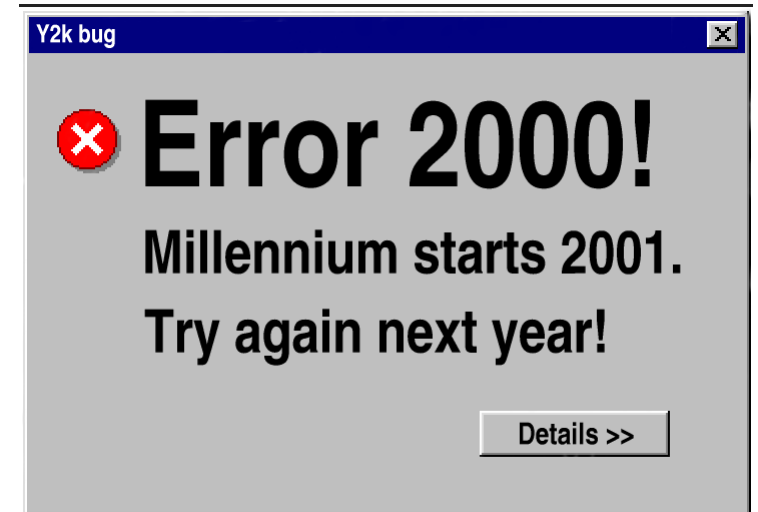
- Costs time on the short term (that is not available)
 - >> Banana software: It's not a bug – it's a feature
 - Rarely large software is 100% free of errors
- ⇒ **Safety critical software is tested according to norms**

Some severe bugs

- Ariane 5 crash (Cost: 370+ million US\$)
 - Radiotherapy overdose (3 dead)
 - Y2k Bug (Cost: „Up to 800 billion US\$“)
- ⇒ **Increasingly safety relevant software: energy, transport, finances, IoT, cloud infrastructure**

Long term advantages

- Stable code > Increases the development time
 - Large (growing) software projects can only be maintained with automated tests.
- ⇒ **But: Tests only detect errors they have been designed to find, no guarantee for error free code**



Grafik: <http://www.toptenz.net/top-10-most-famous-doomsday-prophecies-that-failed.php/y2k-bug>

Detecting errors: Trial & error vs. testing

Trial & error

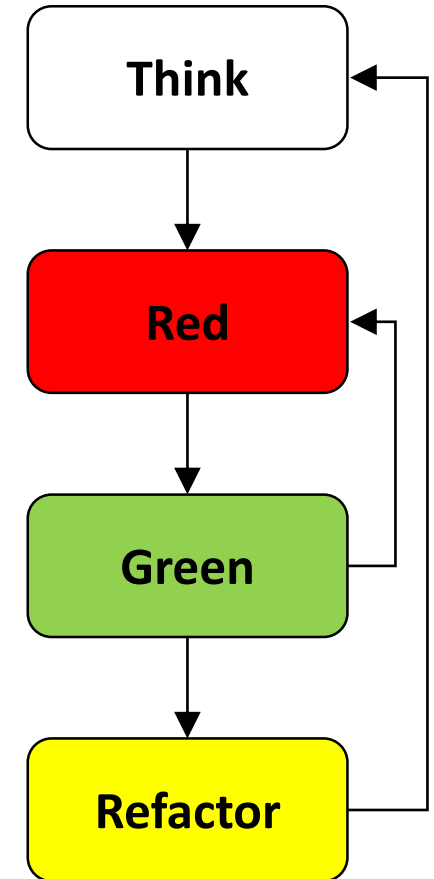
- Write code
- Run through all paths (durchhangeln)

Test

- Formal procedure
- First design and write tests, then code
- Automated testing (i.e. unit tests)

Test Driven Development

1. Think
 - Select requirements to be implemented
 - Specify test cases
2. Red: Implement and execute test cases
 - All tests will fail
3. Green: Implement code and execute test cases
 - Test fails > continue with step 3
 - Test successful > continue with step 4
4. Refactor: Optimize code without changing the functionality
 - Test cases must still execute successfully



Error types

1. Syntax errors

- Violation of syntax „grammatical rules“ of the programming languages, i.e. missing the colon after `else` :
- Code will not run, the interpreter or compiler will find and name the errors

2. Semantic errors

- Syntactically correct and executable code
- Poor implementation
- Example: `if (a = 10)` instead of `if (a == 10)`

3. Logic errors

- Loop repeats one time too less / much
- `<` instead of `≤` in loop

4. Design error

- Customer requirements interpreted wrongly
- Unstructured programming, f.e. code repetition and change only in one place.

The most expensive hyphen in history



Graphic: Public domain
Text <https://priceonomics.com/the-typo-that-destroyed-a-space-shuttle/>

"[The hyphen] gives a cue for the spacecraft to ignore the data the computer feeds it until radar contact is once again restored.

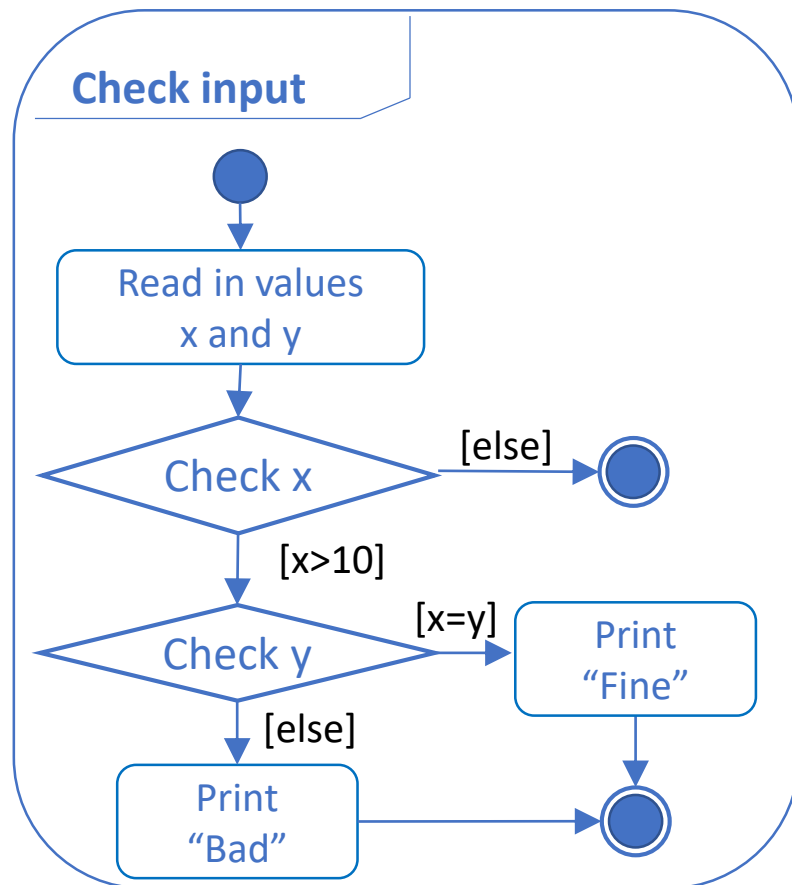
When that hyphen is left out, false information is fed into the spacecraft control systems.

In this case, the computer fed the rocket in hard left, nose down and the vehicle obeyed and crashed."

Richard B. Morrison, a NASA official

Exercise: error types

Find and characterize the 3 errors in the code implemented according to the activity diagram



Three errors

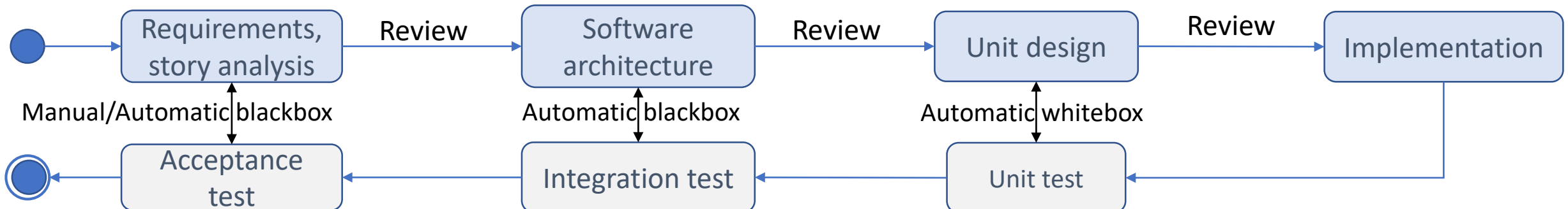
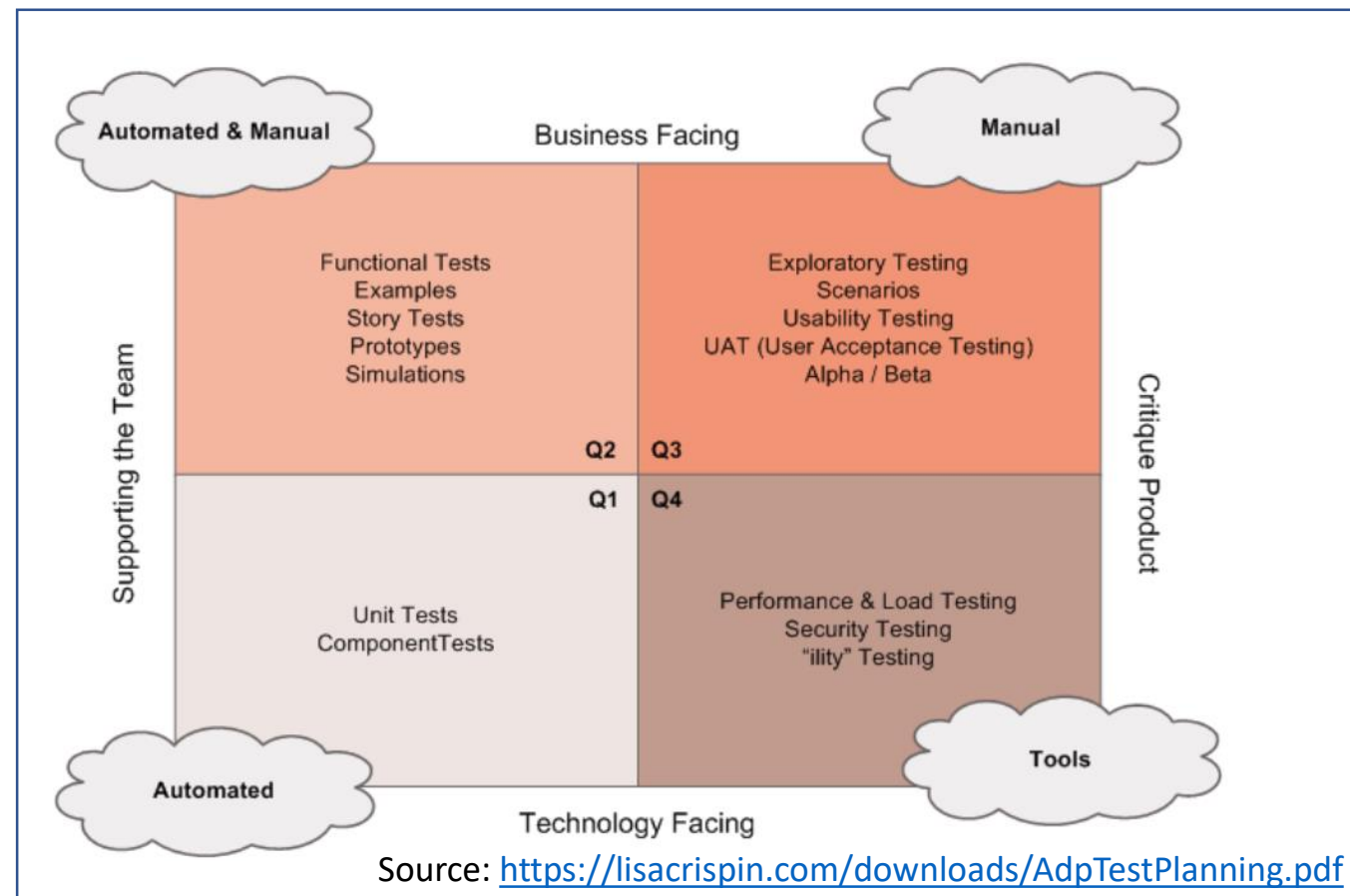
```
x = int(input("x? "))
y = int(input("y? "))

if x > 10:
    if y = x:
        print("Fine")
else
    print("Bad")
```


Ways to test and test types

Ways to test

- Static (code analysis, code review) vs. dynamic tests (execution of code)
- Test environment (manual tests vs. automatic tests)
- Blackbox vs. whitebox Tests



Structure of tests

Test specification

- Written language – like requirements
- Define the preconditions
- Define the environment and potential modifications
- Define the execution steps including input values incl. dynamics
- Define the expected result – measurable, if possible with tolerances!

Test implementation

Test results

- Record the results – measured values
- If test scenario is not deterministic: execute test more than once

Test coverage

Number of test cases

- Are all requirements, interfaces covered?
- Are all possible combinations of values covered?
 - Two 32 bit integers as input: would be $4294967295 \times 4294967295 = 18446744065119617025$ ($>18e^{18}$) combinations
 - Focus on equivalent classes
 - If input range is 0- 4294967295 : e.g. equivalent classes could be 0, 1, 4294967294, 4294967295 for each input

Code coverage

- Function coverage – each function called?
- Statement coverage – each code statement executed?
- Branch coverage – each branch executed (e.g. true and false branch of if/else)
- Modified condition/decision (MCDC) – all relevant combinations of if/else

Key takeaways:

- Clearly understand your story
- Start testing as early as possible!
- Repeat your tests continuously
- Testing is teamwork