1

Basics

- Linux is a free operating system based on UNIX
- First developed as a small but self-contained kernel in 1991 by Linus Torvalds, with the major design goal of UNIX compatibility
- Its history has been one of collaboration by many users from all around the world
- The core Linux operating system kernel is entirely original, but it can run much existing free UNIX software, resulting in an entire UNIX-compatible operating system free from proprietary code
- Linux is a multiuser, multitasking system with a full set of UNIX-compatible tools
- Its file system adheres to traditional UNIX semantics, and it fully implements the standard UNIX networking model

from: Silberschatz, Operating System Concepts

2

The Linux Kernel

- Version 0.01 (May 1991) had no networking, ran only on 80386-compatible Intel processors and on PC hardware, had extremely limited device-drive support, and supported only the Minix file system
- Linux 1.0 (March 1994)
- Version 1.2 (March 1995) was the final PC-only Linux kernel
- Linux 2.0 (June 1996)
- 2.4 and 2.6 increased SMP support, added journaling file system, preemptive kernel, 64-bit memory support
- Linux 4.0 (2015)
- Linux 5.0 (2019)
- Linux 6.0 (2022)

from: Silberschatz, Operating System Concepts

3

Linux Distributions

Standard or Compiled Set of Packages

- Standard, precompiled sets of packages, or distributions, include the basic Linux system, system installation and management utilities, and ready-to-install packages of common UNIX tools
- The first distributions managed these packages by simply providing a means of unpacking all the files into the appropriate places; modern distributions include advanced package management
- Early distributions included SLS and Slackware
  - *Red Hat, Debian and Ubuntu are popular distributions from commercial and noncommercial sources, respectively*
- The RPM Package file format permits compatibility among the various Linux distributions

from: Silberschatz, Operating System Concepts

4

Components of a Linux System

User mode	User applications	bash, LibreOffice, GIMP, Blender, 0 A.D., Mozilla Firefox, ...				
	System components	**init daemon**: OpenRC, runit, systemd...	**System daemons**: polkitd, smbd, sshd, udevd...	**Window manager**: X11, Wayland, SurfaceFlinger (Android)	**Graphics**: Mesa, AMD Catalyst, ...	**Other libraries**: GTK, Qt, EFL, SDL, SFML, FLTK, GNUstep, ...
	C standard library	`fopen` , `execv` , `malloc` , `memcpy` , `localtime` , `pthread_create` ... (up to 2000 subroutines) *glibc* aims to be fast, *musl* aims to be lightweight, *uClibc* targets embedded systems, *bionic* was written for Android, etc. All aim to be POSIX/SUS-compatible.				
Kernel mode	Linux kernel	`stat` , `splice` , `dup` , `read` , `open` , `ioctl` , `write` , `mmap` , `close` , `exit` , etc. (about 380 system calls) The Linux kernel System Call Interface (SCI), aims to be POSIX/SUS-compatible[74]				
		Process scheduling subsystem	IPC subsystem	Memory management subsystem	Virtual files subsystem	Network subsystem
		Other components: ALSA, DRI, evdev, klibc, LVM, device mapper, Linux Network Scheduler, Netfilter Linux Security Modules: *SELinux*, *TOMOYO*, *AppArmor*, *Smack*				
	Hardware (CPU, main memory, data storage devices, etc.)					

from: Wikipedia

5

Standard File Systems

Directory	Description
/	Root, everything appears here, even if not on the same device
/bin	Command binaries, e.g. ls, cat
/dev	Device files
/etc	Configuration files
/home	Home directory
/lib	Libraries for binaries in /bin
/opt	Add-on application software packages
/sys	Information about devices, drivers, and some kernel features
/usr	User utilities and applications
/var	Log files

6

Everything is a file

- Holds for any UNIX like system
- And: *Everything is a stream of bytes*
- Read from and write to by stream of bytes
- Advantage: same tools for everything
- Real and virtual file systems
- Examples under Linux:

/dev/sda	→	*First hard drive*
/dev/sda1	→	*First partition of the first hard disk*
/dev/null	→	*Virtual black hole*
/dev/ttyS0	→	*First serial interface*
/dev/log	→	*System log file*
/proc/1958/stack	→	*Stack of process with ID 1958*
/tmp/myfifo	→	*Named pipe (example)*
/sys/power/state	→	*ACPI power states*

7

1

C Compiler for Linux

- GCC: GNU Compiler Collection
  C, C++, Java, Objective-C, Fortran, Ada, Go
  x86, x86-64, ppc, ppc64, ARM, SPARC, Alpha, H8/300, S/370, S/390, IA-64, 68000, 88000, Coldfire, MIPS, PA-RISC, PDP-11, SuperH, VAX, AVR, AVR32, Blackfin, ...
  Actual Version: 11.1
- LLVM/Clang:
  C, C++, Objective-C, Objective-C++
  x86, x86-64, ppc, ppc64, ARM, SPARC, Alpha, CellSPU, PIC16, MSP430, System z, Xcore
  Actual Version: 12.0
- ICC: Intel C/C++ Compiler

2

GCC: First Steps

```
#include <stdio.h>
int main(void) {
      printf(„Hello, MSE!\n“);
      return 0;
}
```

- Compile and link Hello World example:

```
$ gcc hello.c -o hello
```

- Execute, give full or relative path:

```
$ ./hello
$ /home/graf/projects/task1/hello
```

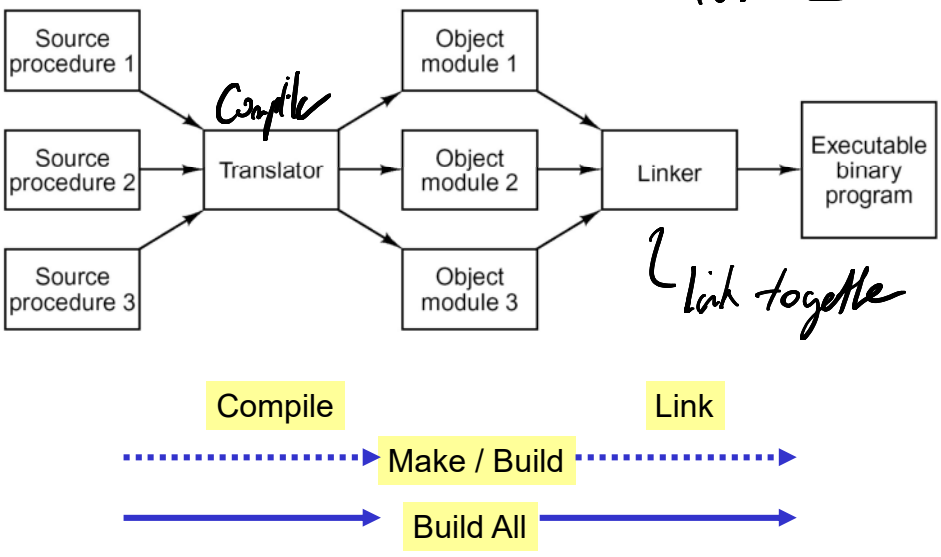
3

GCC: Important Options

- General:
  - *-o file:* *choose output file name, else a.out*
  - *-c:* *compile only → object file*
  - *-S:* *compile into assembly code*
  - *-g:* *include debug information*
  - *--help:* *help*
- Warnings:
  - *-Wall:* *activates all warnings*
  - *-w:* *deactivates all warnings*
- Optimization:
  - *-O, -O1, -O2, -O3: various levels*

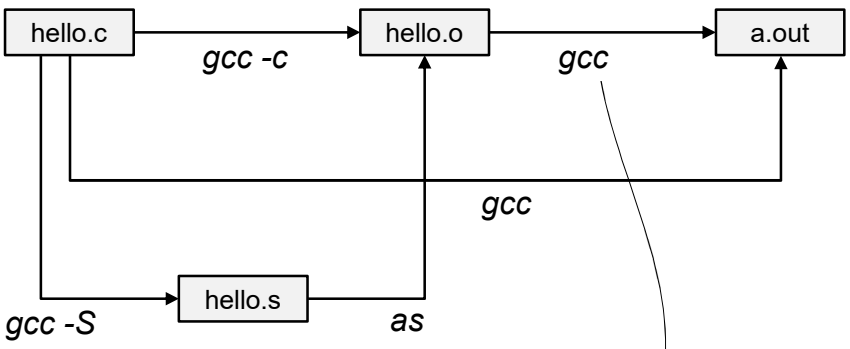
4

Two Stage Translation



5

GCC Compiler / Linker



Linker could be called with *ld hello.o*, however, all necessary libraries must be added, which is extremely tedious -> see https://www.linuxtopia.org/online_books/an_introduction_to_gcc/gccintro_87.html

6

Headerfiles

calculate.c

```
#include "calculate.h"

float calculate(float op) {
	return op * factor;
}
```

calculate.h

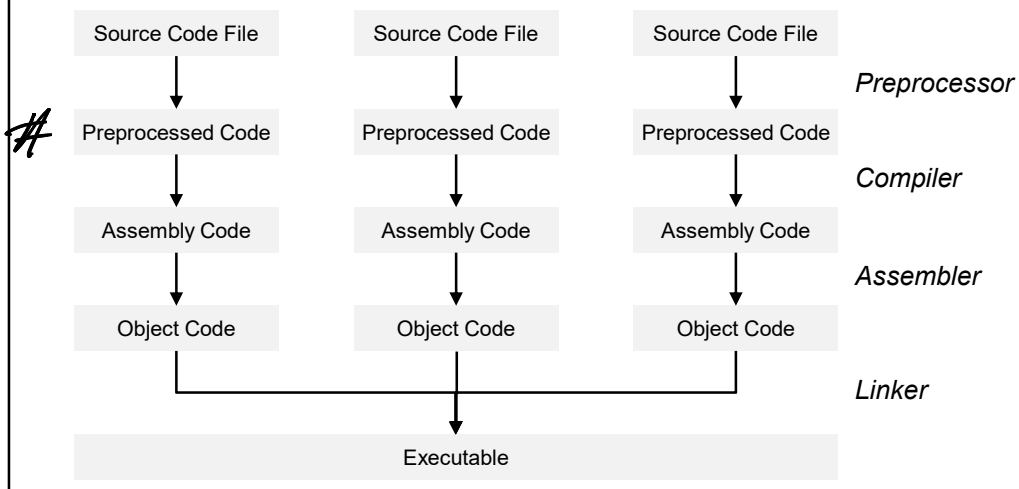
```
#ifndef CALCULATE_H
#define CALCULATE_H

#define factor 10000

float calculate(float);

#endif
```

main.c

```
#include <stdio.h>
#include "calculate.h"

int main(void) {
	printf("result = %f\n", calculate(3.24));
	return 0;
}
```

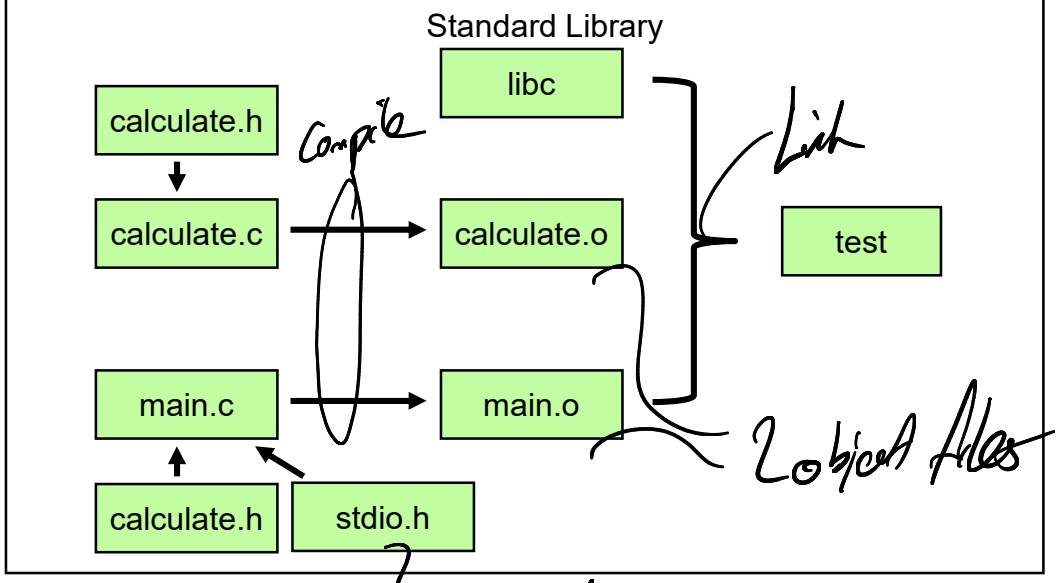
7

Preprocessor



8

Compile / Link



9

Understanding Error Messages

```
main.c: In function ‘main’:
main.c:5:26: warning: implicit declaration of function ‘calculate’ [-Wimplicit-function-declaration]
    5 |  printf("result = %f\n", calculate(3.24));
      |                          ^~~~~~~~~
```

```
/usr/bin/ld: /tmp/ccRU8X1A.o: in function `main':
main.c:(.text+0x11): undefined reference to `calculate'
collect2: error: ld returned 1 exit status
```

```
/usr/bin/ld: /usr/lib/gcc/x86_64-linux-gnu/9/../../../x86_64-linux-gnu/Scrt1.o: in function `_start':
(.text+0x24): undefined reference to `main'
collect2: error: ld returned 1 exit status
```

```
/usr/bin/ld: /tmp/ccWVcfz3.o: in function `main':
main.c:(.text+0x0): multiple definition of `main'; /tmp/ccF9Glu4.o:calculate.c:(.text+0x14): first
defined here
collect2: error: ld returned 1 exit status
```

10

GNU Make

- Build Management Tool
  Runs commands depending on conditions
  Automatically translates projects consisting of many source files
- Interprets a *Makefile*
- *make* searches for *Makefile* in current directory
- A Makefile can have many targets
- A target has dependencies and rules

Target : Dependencies
(tab) Rules

```
prog: prog.c prog.h
	gcc -o prog prog.c
```

11

Make: first example

```
CC = /usr/bin/gcc
CFLAGS = -Wall -g
LDFLAGS = -lm –lpthread
EXEC_NAME = test
OBJ = file1.o file2.o file3.o file4.o file5.o

all: $(OBJ)
	$(CC) $(CFLAGS) -o $(EXEC_NAME) $(OBJ) $(LDFLAGS)

%.o: %.c
	$(CC) $(CFLAGS) -c $<
```

- Variables in the environment become make variables
- Many implicit rules

12

More about Make

Declare and use variables

```
CC = /usr/bin/gcc
    $(CC) –Wall -c $<
```

Rules with patterns

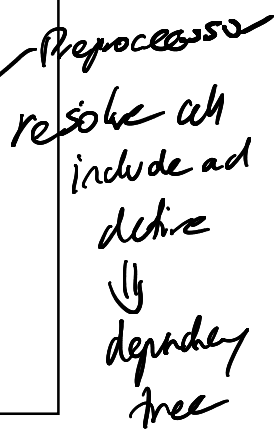
```
%.o: %.c
    gcc -Wall -c $<
```

This rule states that each o-file depends on ist c-file and gives the means to produce it.
$< is the first dependency
You could actually omit this rule alltogether because ist an implicit rule!

13

Make: a more complex example

```
CC = /usr/bin/gcc
CFLAGS = -Wall -g
LDFLAGS = -lm -lpthread
SRC = main.c calculate.c
EXEC_NAME = test
DEPENDFILE = .depend

# Compile and Assemble C Source Files into Object Files
OBJ = $(SRC:.c=.o)

# Create dependencies to make sure the modules are rebuilt after modification.
$(DEPENDFILE): $(SRC)
        @echo Create dependencies...
        $(CC) -MM $(SRC) > $(DEPENDFILE)

# This target creates the executable from all the given object files.
all: $(DEPENDFILE) $(OBJ)
        $(CC) $(CFLAGS) -o $(EXEC_NAME) $(OBJ) $(LDFLAGS)

# This target removes all generated files from the system.
clean:
        @echo Remove generated files...
        rm -f $(EXEC_NAME) $(OBJ) $(DEPENDFILE)

-include $(DEPENDFILE)
```



14

More about Make

Pattern substitution for source files

```
OBJ = $(SRC:.c=.o)
```

Dependencies

```
$(DEPENDFILE): $(SRC)
	$(CC) -MM $(SRC) > $(DEPENDFILE)
```

The option –MM calculates dependencies of compilation units without actually compiling. The output is written it into a dependency file.

Ignore a missing file

```
-include $(DEPENDFILE)
```

Try to include it, suppress error message if file does not exist

15

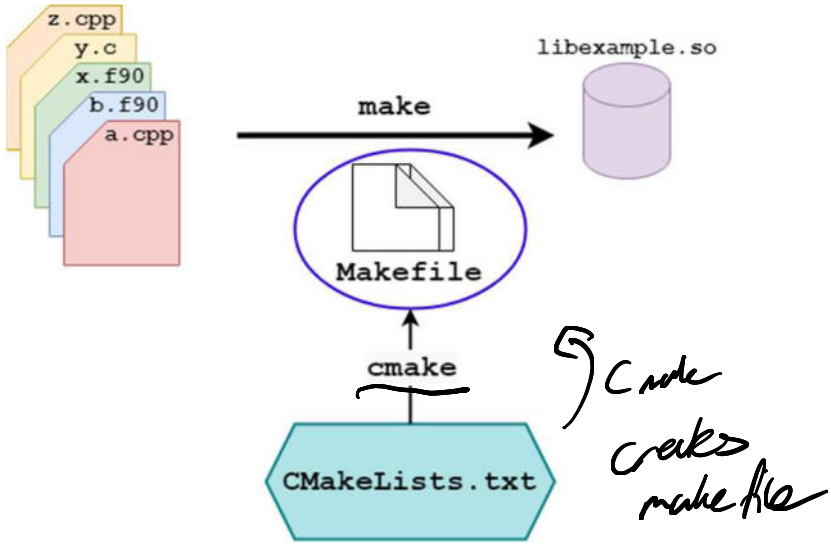
CMake

- Cross-platform Make (nothing to do with C language)
- Creates Makefiles or project files for various IDE's
- Tools for testing and publishing:

  CTest und CPack
- All functionality in text files: *CMakeLists.txt*
- Für C, C++, Java, C#, Fortran …

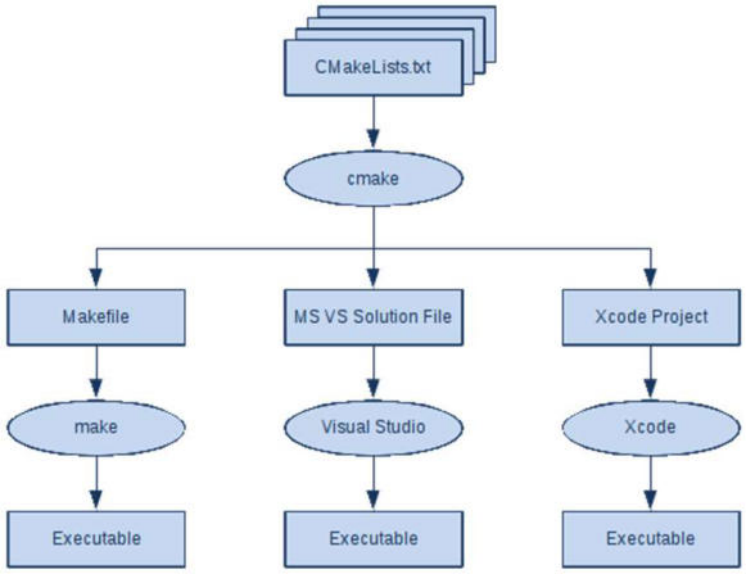
16

Creating a Library



17

Platform Independent



18

CMake: Simple Example

```
cmake_minimum_required(VERSION 3.5)
project(Test)

add_executable(test main.c calculate.c)
```

```
$ mkdir build && cd build
$ cmake ..
$ make
```

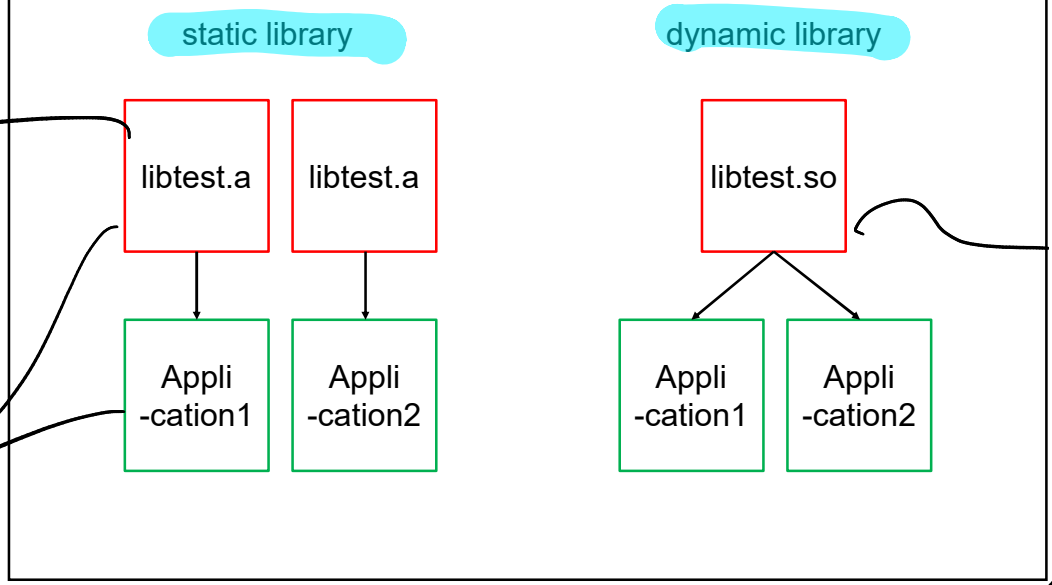
19

Why Libraries

- Are collections of precompiled object files
- Come as static or dynamic library
- Must be built for a certain platform or can be downloaded
- We must know its interface -> for C, we need header files
  - libs go into /usr/lib
  - header files go into /usr/include
- Suffixes
  - On Windows: static = *.lib, dynamic = *.dll
  - On Linux: static = *.a, dynamic = *.so
- GCC links against libgcc (compiler specific subroutines) and libc (or glibc)

1

Static versus Dynamic Library



2

How about the Standard Library libc?

```
ost@ost:~/projects/ss/TestDLopen/build$ ldd ./test
        linux-vdso.so.1 (0x00007fffa41c6000)
        libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007fcceb98a000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fcceb798000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fcceb9d7000)
ost@ost:~/projects/ss/TestDLopen/build$ lsof | grep libc- | head
systemd    1555                          ost  mem       REG              259,3  2029224
pulseaudi 1561                           ost  mem       REG              259,3  2029224
pulseaudi 1561 1612 null-sink            ost  mem       REG              259,3  2029224
pulseaudi 1561 1625 snapd-gli            ost  mem       REG              259,3  2029224
tracker-m 1563                           ost  mem       REG              259,3  2029224
tracker-m 1563 1565 gmain                ost  mem       REG              259,3  2029224
tracker-m 1563 1569 gdbus                ost  mem       REG              259,3  2029224
tracker-m 1563 1574 dconf\x20            ost  mem       REG              259,3  2029224
tracker-m 1563 1643 pool-trac            ost  mem       REG              259,3  2029224
dbus-daem 1567                           ost  mem       REG              259,3  2029224
ost@ost:~/projects/ss/TestDLopen/build$ lsof | grep libc- | wc -l
296
ost@ost:~/projects/ss/TestDLopen/build$ size /usr/lib/x86_64-linux-gnu//libc-2.31.so
   text    data     bss     dec     hex filename
1988852   20456   16184 2025492  1ee814 /usr/lib/x86_64-linux-gnu//libc-2.31.so
ost@ost:~/projects/ss/TestDLopen/build$
```

3

What happens?

- Static linking
  - *All necessary libraries get linked at link time together to form an executable holding every necessary part of a program.*
- Dynamic linking
  - *For every function which is located in a library a stub is inserted into the code. The resulting executable contains lots of those stubs.*
  - *When executing the program it will run into such a stub. The OS will then have to find the library containing the code to which the stub refers.*
  - *The stub is then replaced with a call to the function. Searching and replacing has a time penalty, though, only when this part of the code runs for the first time.*

4

Comparison

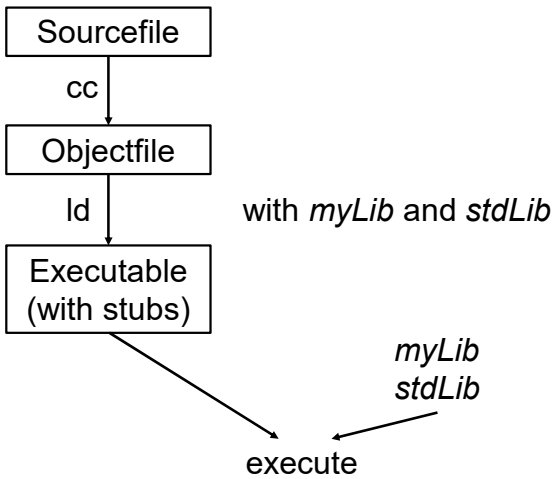
Static	Dynamic
Linker searches used functions and copies them into your executable	Linked dynamically at run-time by the OS, same copy is used by many programs
Executable becomes larger	Executables only has stubs
If library changes, executable has to be linked again	If library changes, executable will automatically use new library
If library changes, previous executable will still run with the old version of the library	If library changes, executable may run with new library or may become incompatible

Use Case 1: Library to read sensor

Use Case 2: Graphic library, ROS, math library

5

Linktime vs Runtime



6

Version Control with Git

Andreas Kunz

Agenda

Today's goals

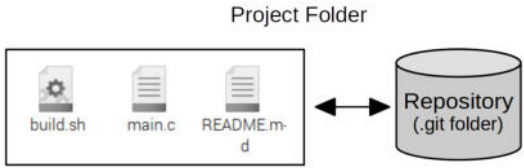
- Short theory section
- Exercises and practice

Why should I use a Version Control System?

- Traceability
  - What was changed and when?
  - Who changed it and why?
- Versioning
  - Control different versions of code
- Teamwork
  - Many developers can work on the same code
  - Code can easily be shared
- **Code Quality**

How does Git work?

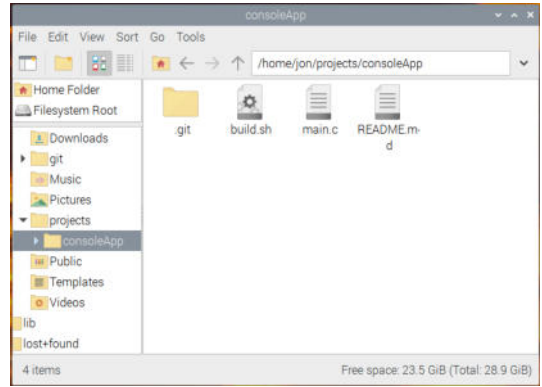
- Git tracks changes in any set of files
- A new state of the files (version), is stored in a **commit**
- Commits are managed in a repository



How does Git work?

Everything can work as git server

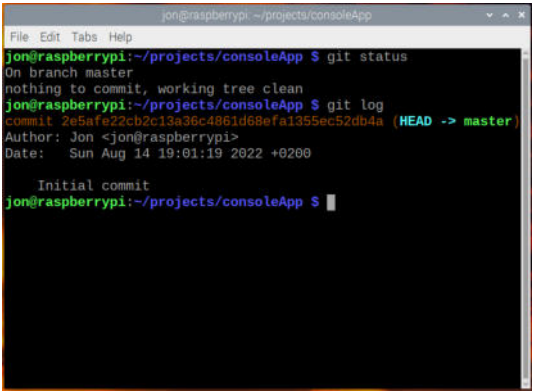
- The Repository is local
- Workspace and Repository are in the same folder
- Git works decentralized / distributed



What is a commit?

commit has unique hash

- The most important thing in Git
- A Commit is a frozen state (version) of all the files in the workspace
- A Commit has an unique commit hash
- A Commit contains also a message, the date and the author information



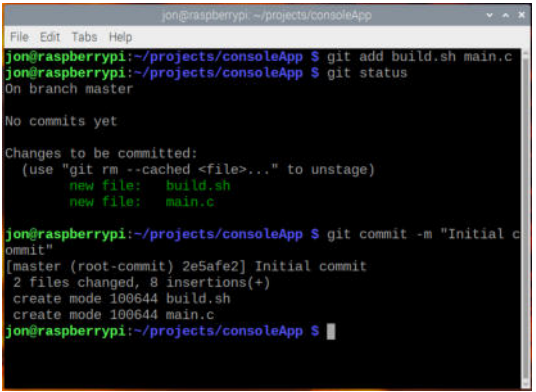
What is a commit?

Changes and dependencies

- The repository does not only store the commits, it also stores the dependencies between them
- The resulting graph represents the evolution of the project
- Due to the graph, not only the versions are documented also the change sets

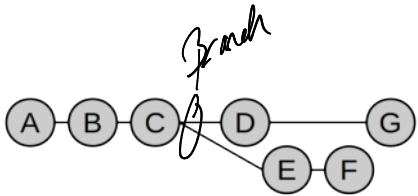
Create a commit

- First, all the changes which should be part of the next commit must be selected
- They are added to the staging area with the **add** command
- Then, the commit is created



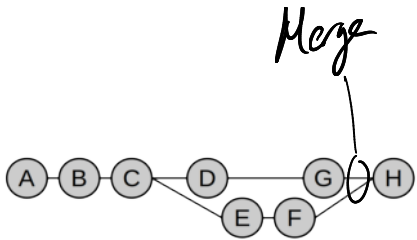
What is a branch?

- A branch is basically a simultaneous work on the same project
- Simultaneous work occurs when:
  - working in a team
  - multiple features are developed in parallel
  - an older software release needs a bugfix
- Since the repositories are local, the co-workers work of course in parallel



How to merge branches?

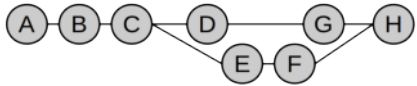
- To combine branches, a merge is necessary
- Git has a complex algorithm, which makes merging branches easy
- When one branch is merged into another, the later then contains the commits of both branches in his history (commit graph)



Merge commit

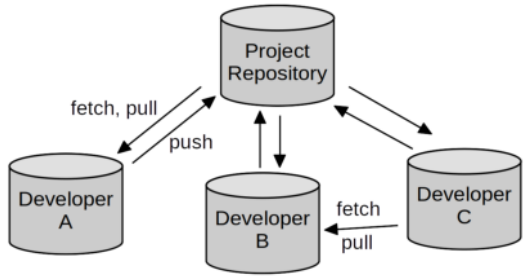
How to merge branches?

- When doing a merge, a merge commit is created (H)
- If on both branches exactly the same changes were made, a merge conflict occures which the user has to resolve



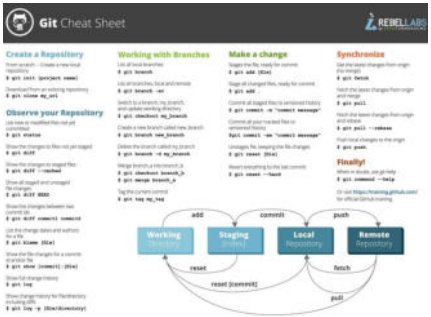
How to work in a team?

- To exchange local commits with other repositories, the **fetch, pull** and **push** commands are used
- Due to Gits decentralized architecture, commits can be exchanged between any repositories
- Often, a repository (remote) on a server is used to store the commits of all developers. One popular example is GitHub.



Whats next?

- Well, to learn Git, practice is necessary!
- There are tons of information in the internet.
- The official documentation is well understandable: https://git-scm.com/doc
- A cheat sheet as the one on the right is very handy, just browse the internet for one.



Embedded Software Development under Linux

August 7, 2024

Preparatory Work (Do in Advance)

Laptop (Windows 10+), possibly with micro-SD Card reader, WiFi and at least 2 USB Ports.

- Download and Install **Raspberry Pi Imager**:
  https://www.raspberrypi.com/software/
- Download Image File: **Raspberry Pi OS with desktop**:
  raspios-bookworm-arm64.img.xz
- Download and Install **PuTTY**:
  https://www.putty.org/
- Optional: Download and Install **FTDI Drivers**:
  https://ftdichip.com/drivers/vcp-drivers/

You can bring your own Raspberry Pi (3, 4 or 5) SBC.

Erich Styger

- ETHZ, HIWARE, Metrowerks, Motorola, Freescale, NXP
- **HSLU**: 2008 Lecturer, Professor & Researcher
- **Focus**: Microcontroller, Communication, Robotics, Software & Tools
- **Curriculum**: Bachelor & MSE (Electrical Engineering & Computer Science)
- **External**: STEM, SatW TechDays, Fraunhofer Roberta Initiative
- **Blog**: McuOnEclipse

Agenda Day 2

0830 **Embedded Linux**
Software & Tools, Raspberry Pi
Pins, LED, Button, I^2C
WLAN, UART, SSH, editing, basics

1000 Break

1030 **Python Programming**

1200 Lunch

1300 **Lab Guided Tour**

1400 **C Programming**

1500 Break

1530 **Threading**

1730 Wrapup

Philosophy

- Material & Snippets on Moodle
- Complete System setup
- Hands-on Labs
- Enabling your own development
- Do things while explained
- Challenges for extended work
- Explore extra sensor breakout modules
- Lots of material, fast pace $\rightarrow$ ask questions!

Preview Day 3:
Networking, Kernel Module, Lab application/challenge

Hardware Setup

- Main Platform: **Raspberry Pi**
- Helper Platform: **RP2040** (Pico) microcontroller board
  - UART-2-USB: Raspberry Pi login console
  - 3 LEDs
  - 5 Push buttons
  - I^2C OLED display
  - I^2C Sensor (Temperature & Humidity)
- Extra breakout sensor boards (data on Moodle)
  - CAP1188 - 8-Key Capacitive Touch Sensor
  - MCP9808 High Accuracy I2C Temperature Sensor
  - MMA8451 Triple-Axis Accelerometer Sensor
  - MPL115A2 - I2C Barometric Pressure/Temperature
  - VCNL4010 Proximity/Light sensor
  - AS7262 6-Channel Visible Light / Color Sensor
  - DS3231 Realtime Clock with AT24C32 EEPROM

Software & Tools

“A foo walks into a bar, takes a look around and says ”Hello World!” – Unknown

August 7, 2024

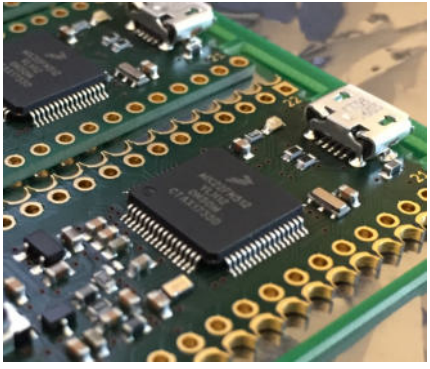
Software & Tools: Goals

- Introduction Embedded Systems
- Hardware
  - Raspberry Pi
  - RPi3, RPi4
  - Extra hardware
- Software und Tools
  - GNU
  - Linux

Overall Goal: System Setup and development of small applications for embedded Linux (GPIO, I^2C, UDP, console, ...) in C and Python.

Embedded System

- Embedded Linux
- Computer, *embedded*
- System, part of the whole thing, part of something
- Not a 'usual' computer
- Without the 'usual' computer equipment: mouse, keyboard, screen, ...
- Optimized for a dedicated purpose (¬ universal computer)
- Real-time capabilities



SPC
Single Board PC

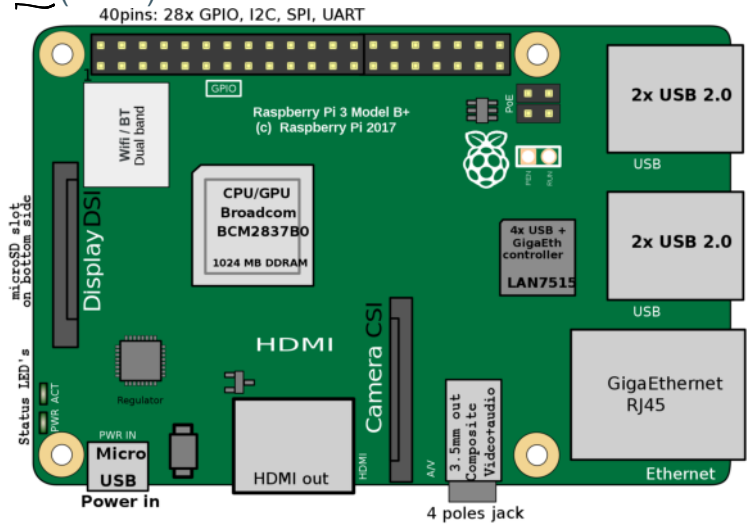
Head less
No Screen, Keyboard

Hardware

- Rasperry Pi 3 Model B+ (minimum)
- Raspberry Pi 4
- $>$8 GB Micro SD-Karte
- 2A 5V Power Supply (USB-C, Micro-USB)
- LAN/WLAN & SSH (to connect it.)
- UART $\leftrightarrow$ USB-CDC Cable
- Breadboard & cables
- Switches, Resistors, LEDs, Sensors, ...

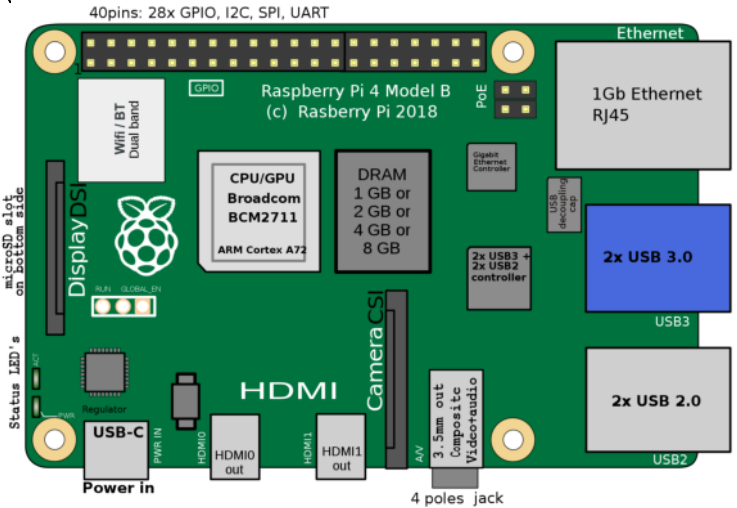
Be careful: ESD, shorts, no hardware modification while powered!

Hardware - RPi3 (2018)



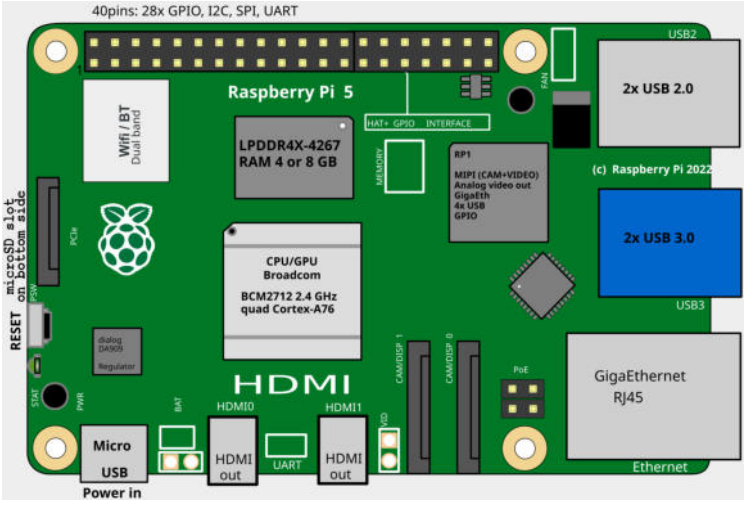
Source: Wikipedia

Hardware - RPi4 (2019)



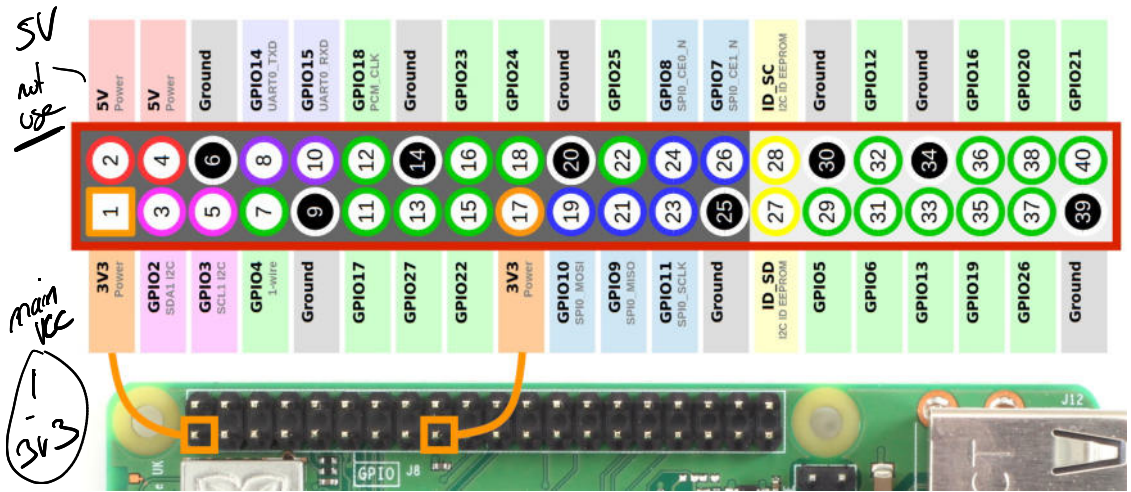
Source: Wikipedia

Hardware - RPi5 (2023)



Source: Wikipedia

Hardware - Pins



Source: https://www.raspberrypi-spy.co.uk

Cross-Development

Develop in Host Machine

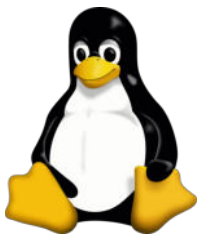
- **Cross Development**: Development on a host for a different target system
  - Host (Windows, Linux) ⇒ Embedded Target (e.g. Microcontroller or embedded Linux system)
  - Host more powerful (RAM, Disk, Display)
- **IDE + Toolchain + Build Environment**:
  - **IDE**: Editor, Build, Projekte
  - **Toolchain**: Compiler, Linker, Tools, Debugger, Standard Libraries
  - **Build**: IDE project, Make, Cmake, Ninja, ...
- Target **Raspberry Pi**
  - Powerful, but less than a usual desktop machine
  - **Cross**: Kernel and System development
  - **Native**: Application development, drivers
  - **GNU** Tools & tool chain
  - **Headless**: No display, no keyboard, no mouse → Network & UART

↳ only used HW (connected + Sensors)

GNU

- GNU: ”GNU’s Not Unix” (recursive acronym)
- Originally: GNU as a Unix-like OS ⇒ ”completely free software operating system”
- Founder: Richard Stallman (MIT) (developed for example the GNU Emacs)
- Free software, GPL (GNU General Public License)
- Own GNU ’Hurd’ Kernel not successful → GNU with Linux Kernel (Linus Torvalds)
- GNU Operating System ⇒ Linux, www.gnu.org
- GNU Tools (cross-platform) ⇒ GNU compiler collection, sed, awk, make, find, grep, bash, ...

Source: Wikimedia



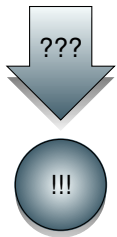
Source: Wikipedia

GNU = Linux + GNU Tools

Software & Tools: Summary

Using

- Hardware
  - Host PC + Raspberry Pi
  - Breadboard, Cables
  - Actuators, Sensors
  - **Headless**
- Software
  - GNU & Linux
  - GNU Toolchain
  - C & Python



Can you do this?

not a full PC
hidden embedded in something

1. Explain key characteristics for an Embedded System.
2. Naming key aspects of Linux. open source OS based on UNIX
3. Knowing meaning of 'headless' development. no Peripherals: keyboard Screen
4. How to properly connect hardware to the Raspberry Pi pins.

↳ VCC: 3V3 → Sensors need 3V3
but there are 5V Pins

↳ Host System
Connects to it.

Dedicated Functionality, Real Time OS

Raspberry Pi

“There is no way for the RPI to boot an operating system without a microSD card present.”
– Timothy Short

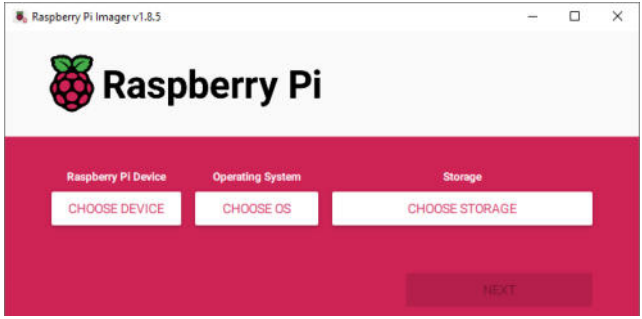
August 7, 2024

Raspberry Pi: Goals

- Setup Raspberry Pi OS
- Creating Image for SD card
- Enabling SSH
- Enabling login console
- Using PuTTY over UART
- LAN & WLAN connection
- Enabling I^2C
- Login
- Shutdown and Reboot

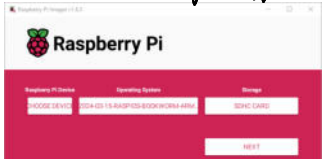
Raspberry Pi OS

- Raspberry Pi Imager
- Download and install: https://www.raspberrypi.com/software/



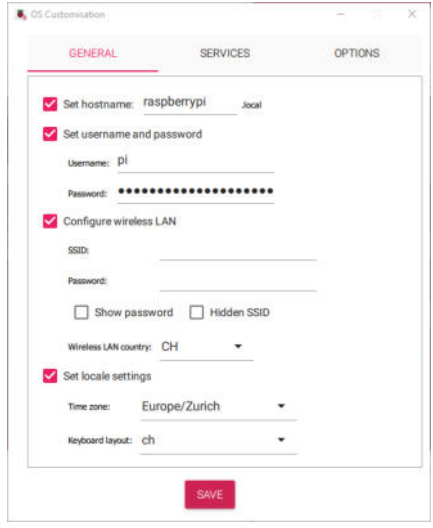
Hint: Use CTRL+SHIFT+L to choose language.

Imager Configuration



- **Choose OS** → Scroll down to select downloaded image → **Use custom**
- Configuration
  - General: Set **username and password**
  - General: Enable **WiFi** with **CH** → SSID, pwd
  - Services: Enable **SSH with password**
- **Write Image**

> ⚠ SD card: be careful!



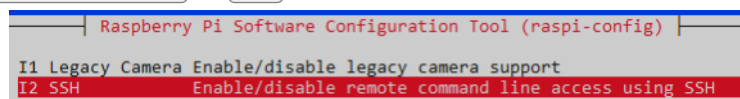
SSH

Secure Shell:

```
$ ssh <user>@<host>
```

```
$ ssh pi@192.168.1.169
```

- Enable using file on the SD card
  - Open SD card content on host
  - Create empty file named `ssh` on file system
  - Note: file will be removed by boot
- Enable using Raspberry Pi Configuration
  - `$ sudo raspi-config`
  - `Interface Options` → `SSH`



Login Console

booths folder

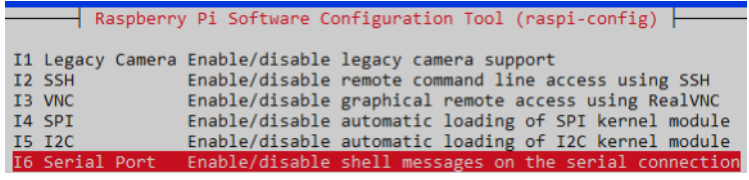
MSE

Login Shell over UART (puTTY)

- Enable using **config.txt** on the SD card
  - Open `config.txt` on SD card with text editor
  - Add

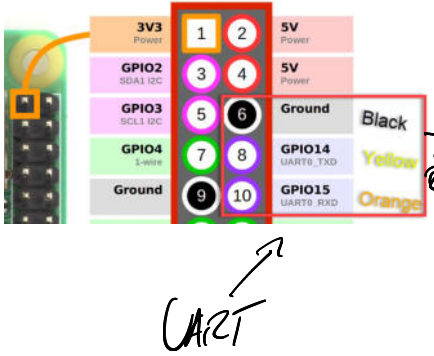
```
[all]
enable_uart=1
```

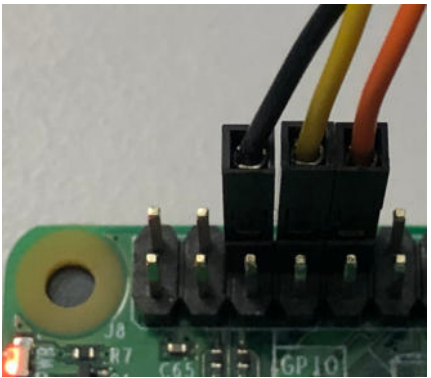
- Enable using Raspberry Pi Configuration
  - `$ sudo raspi-config`
  - `Interface Options` → `Serial Port`



FTDI Cable UART Connection

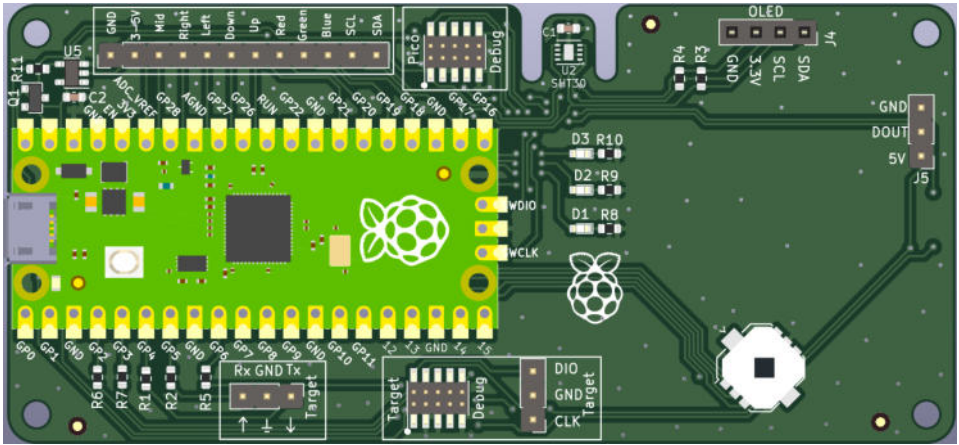
- UART-USB CDC (VCOM) adapter plus Terminal Program on Host (PuTTY)
- FTDI VCP adapter drivers: https://ftdichip.com/drivers/vcp-drivers/
- RPi 5: use UART port (adapter cable needed)



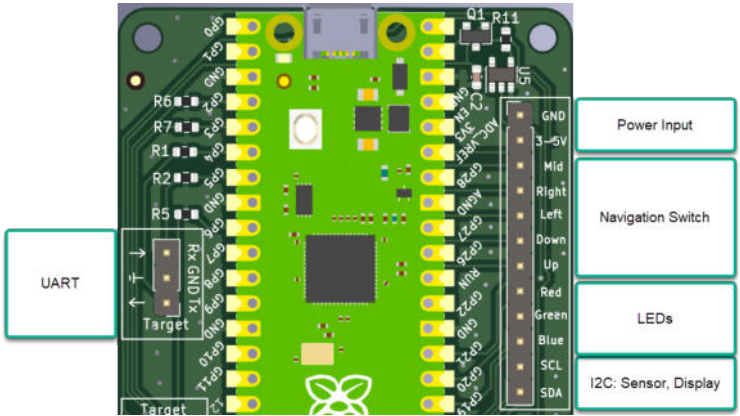


Pico Board

- GPIO, I^{2}C Interface $\rightarrow$ Buttons, LEDs, SHT31, OLED (SSD1306/SHT1106)
- UART Interface $\rightarrow$ VCOM on host



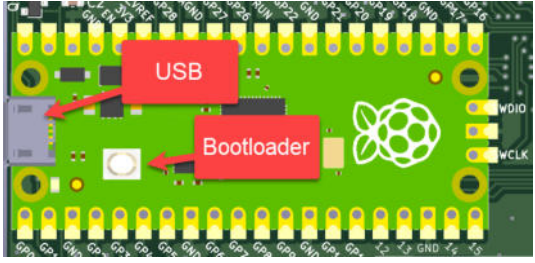
Pico Board Expansion Header



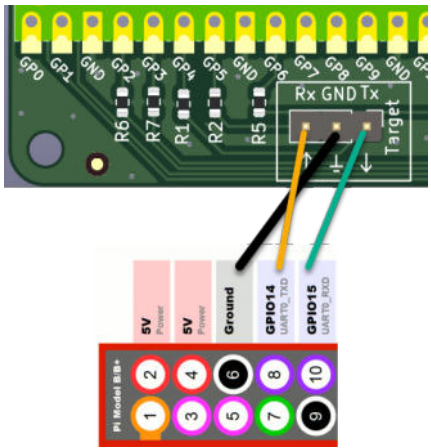
3.3V Logic Levels!

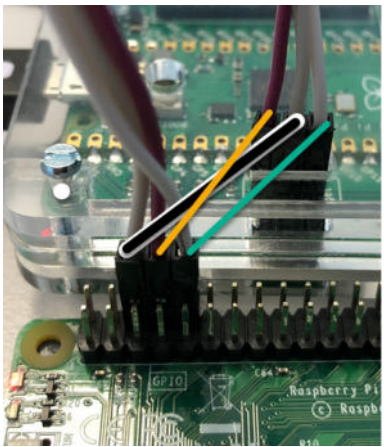
Pico Board Firmware Update

- Power board with Bootloader button pressed
- Enumerates as USB MSD Device
- **RI-RP2** drive with `INDEX.HTM` & `INFO_UF2.TXT`
- Copy **PicoRaspy.uf2** to drive



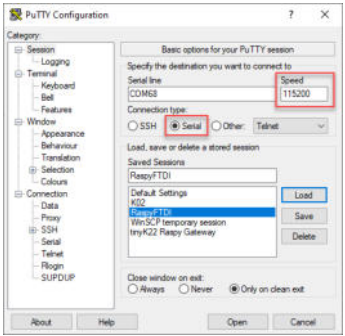
Pico UART Connection

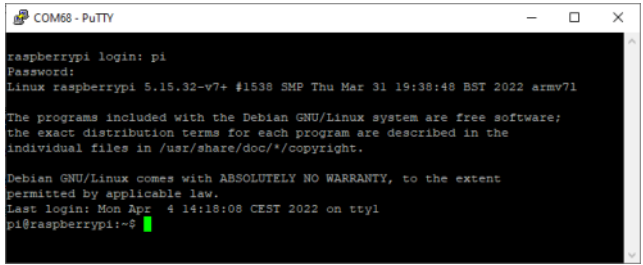




PuTTY

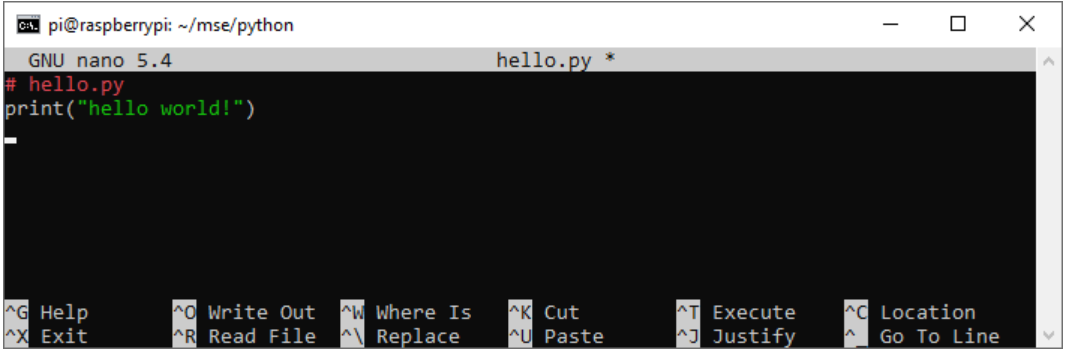
- Terminal program: https://www.putty.org/
- VCOM **Serial** Port with Baud **115200** (or reboot, RPi sometimes changes prescaler)
- Only **one** connection





nano Editor

```
$ nano <file name>
```



exit: CTRL+x

Hostname

```
$ hostname
```

Manually:

```
$ sudo nano /etc/hostname
```

```
$ sudo nano /etc/hosts
```

```
$ sudo reboot
```

Configuration tool:

```
$ sudo raspi-config
```

`System Options` → `Hostname`

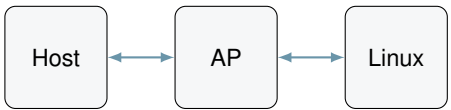
Ethernet Connection

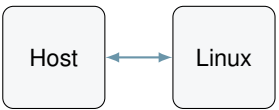
- Recommended: Network (WLAN, LAN) connection with AP/Router
- SSH (Secure Shell), SCP (Secure Copy), WinScp, ftp, ...
- Fallback: Direct Ethernet Cable between Host and Raspberry Pi
  - Name: `<hostname>.local`
  - Issue: can disconnect host from other networks

```
$ ping raspberrypi.local
```

```
$ ssh pi@raspberrypi.local
```

```
$ ifconfig
```



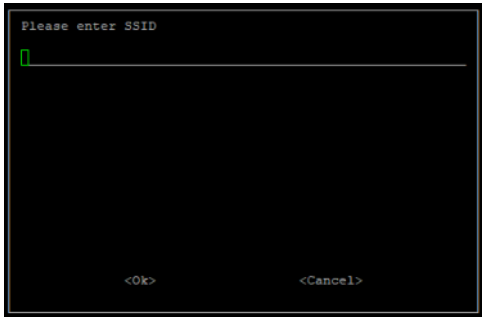


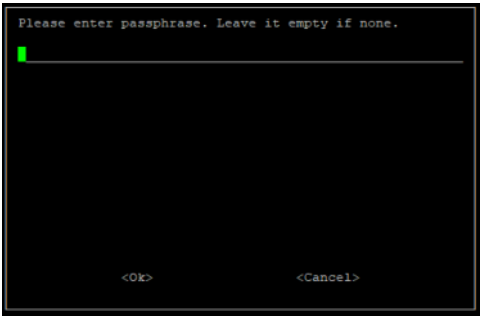
WLAN

Configuration tool:

```
$ sudo raspi-config
```

`System Options` → `Wireless LAN`



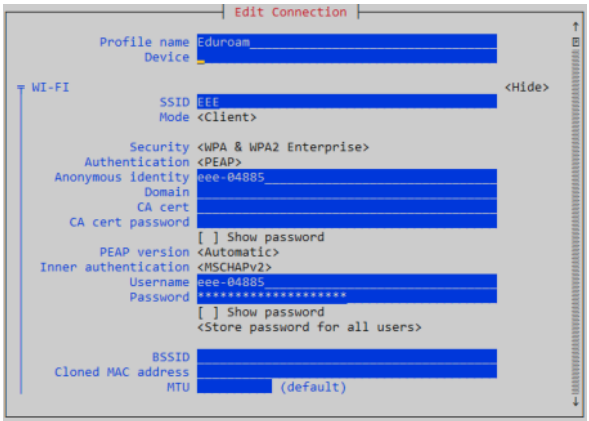


Eduroam

Network Management User Interface:

```
$ sudo nmtui
```

Add → WiFi



Updates

Check network connection:

```
$ ifconfig
```

```
$ ip address
```

```
$ ping www.google.ch
```

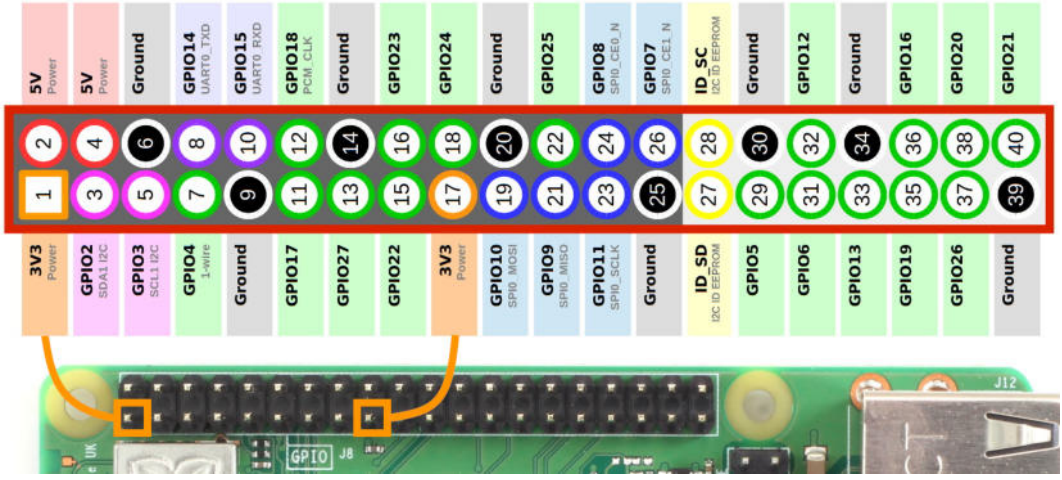
Update package lists and update system:

```
$ sudo apt update
```

```
$ sudo apt upgrade
```

Initial upgrade might take up to 20-30 minutes!

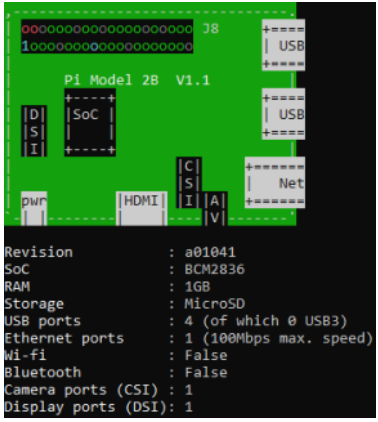
Pins & GPIO



Source: Simple Guide to the Raspberry Pi GPIO Header

Pins & GPIO

```
$ pinout
```




```
J8:
   3V3  (1) (2)  5V
 GPIO2  (3) (4)  5V
 GPIO3  (5) (6)  GND
 GPIO4  (7) (8)  GPIO14
   GND  (9) (10) GPIO15
GPIO17 (11) (12) GPIO18
GPIO27 (13) (14) GND
GPIO22 (15) (16) GPIO23
   3V3 (17) (18) GPIO24
GPIO10 (19) (20) GND
 GPIO9 (21) (22) GPIO25
GPIO11 (23) (24) GPIO8
   GND (25) (26) GPIO7
 GPIO0 (27) (28) GPIO1
 GPIO5 (29) (30) GND
 GPIO6 (31) (32) GPIO12
GPIO13 (33) (34) GND
GPIO19 (35) (36) GPIO16
GPIO26 (37) (38) GPIO20
   GND (39) (40) GPIO21

For further information, please refer to https://pinout.xyz/
```

GPIO Pin Status

```
$ pinctrl help
```

```
$ pinctrl get
```

```
0: ip    pu | hi // ID_SDA/GPIO0 = input
1: ip    pu | hi // ID_SCL/GPIO1 = input
...
```

```
$ pinctrl get 21,26
```

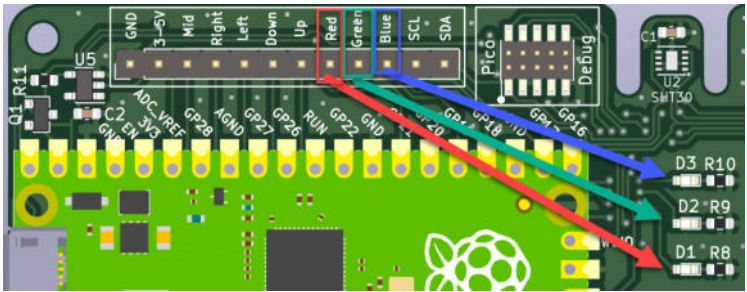
Input `ip`, Output `op`, set high `dh` and low `dl`:

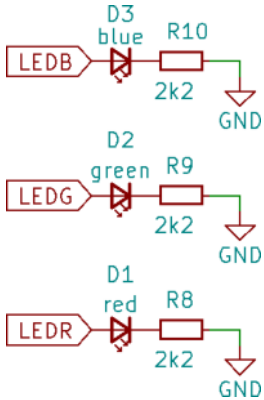
```
$ pinctrl set 21 op
```

```
$ pinctrl set 21 dh
```

LED

- Current limiting resistor
- Pin connection on Anode: HIGH active
- Pin connection on Cathode: LOW active

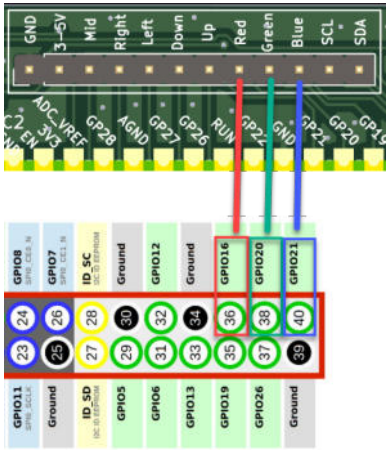




LED

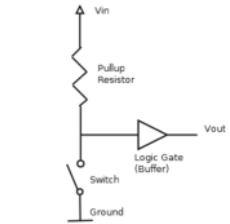
Control the three LED.

- GND connection (UART or I/O header)
- Blue, Green & Red LED pins
- Use Raspberry Pi GPIO: **GPIO16**, **GPIO20** & **GPIO21**
- Use `pinctrl`

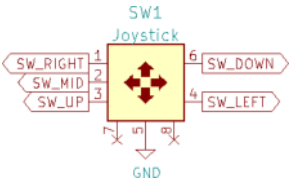


Buttons and GPIO Pull-Ups

- Defined logic level: Pulls (up or down) resistor
- Usually needed for open circuits like switches
- Internal Pulls: $\pm$ 50 kΩ
- `$ pinctrl set 26 pu`
- Different handling based on SoC/Raspberry Pi and Linux version



Source: Wikipedia



Using external hardware resistor is recommended.

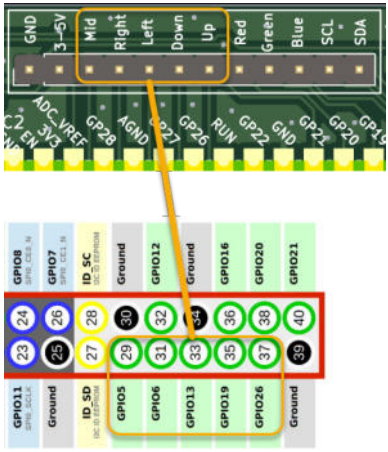
picoRaspy pulls-up nav switch pins!

Buttons and GPIO Pull-Ups

MSE

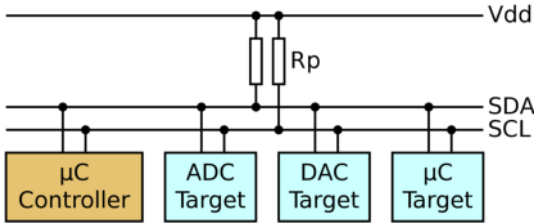
Get the nav switch status.

- GND connection (UART or I/O header)
- Mid, Up, Down, Left & Right pins
- Use Raspberry Pi GPIO:
  GPIO5, **GPIO6**, **GPIO13**, **GPI19** & **GPIO26**
- Use `pinctrl`



I2C Enable

- Inter-Inter-Circuit → IIC → I^2C
- **SCL** (Clock, typical 400 kHz), **SDA** (Data), with pull-ups
- Addressable devices, typical 7-bit address
- Device data: typical memory mapped (device memory address)
- Typical I^2C bus data access
  - **Write**: SelectDeviceAddrOnBus(7b) → WriteMemAddress(8b) → WriteMemData(n*8b)
  - **Read**: SelectDeviceAddrOnBus(7b) → WriteMemAddress(8b) → ReadMemData(n*8b)



Source: Wikipedia

I2C Enable

```
$ sudo raspi-config
```

`Interface Options` → `I2C`

get I²C status with non-interactive mode:

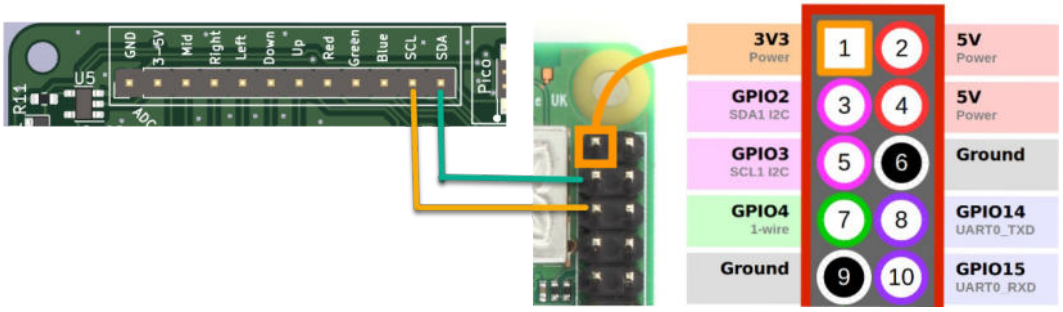
```
$ sudo raspi-config nonint get_i2c
```

1 (!!!): disabled, 0: enabled

Use **zero** (!!!) to **enable** it:

```
$ sudo raspi-config nonint do_i2c 0
```

I2C Connection



I²C devices need to be powered: Pico USB or GND + 3V3. **NOT** 5V!

Do not use 5V! Usually, do **not** change power while system is on.

I2C Detect

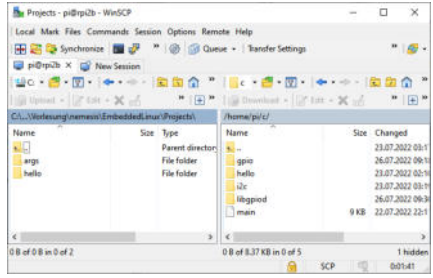
```
$ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- 3c -- -- --
40: -- -- -- -- 44 -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

If `i2cdetect` not found:

```
$ sudo apt-get install i2c-tools
```

Copy Files

- UART
  - `PuTTY`
  - `nano`
  - copy-paste
- WinSCP
- `scp`



Secure Copy and `unzip`:

`scp` `[OPTION]` `[user@]SRC_HOST:]file1` `[user@]DEST_HOST:]file2`

```
> scp files.zip pi@rpi2b:/home/pi
```

```
$ unzip files.zip
```

Shutdown & Reboot

- SD Card subject to failure
- **Wear out**: Log2Ram: Extending SD Card Lifetime for Raspberry Pi LoRaWAN Gateway
- Issue: **Write** during power loss

```
$ sudo reboot
```

```
$ sudo poweroff
```

→ Green LED blinks, then only RED is on.

Always shut down correctly, if you can!

Remote Desktop

only with full image -> otherwise no GUI

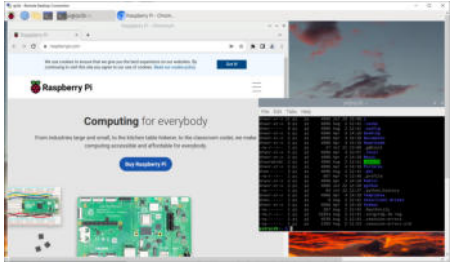
MSE

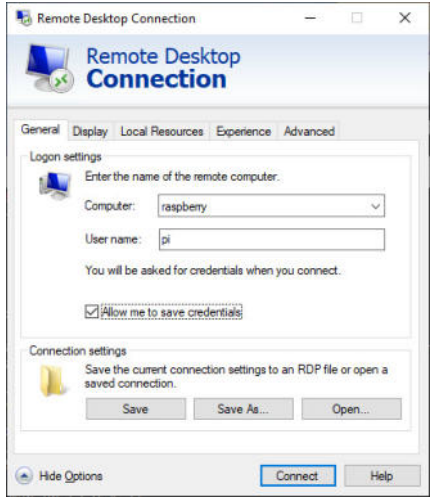
```
$ sudo apt-get install xrdp
```

```
$ sudo reboot
```

```
$ sudo ps -A | grep xrdp
```

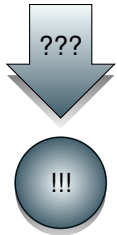
Windows → Remote Desktop Connection





Raspberry Pi: Summary

- Raspberry Pi Installation
- Login: SSH & PuTTY
- Hardware: Pins, UART & I^2C
- LAN & WLAN
- Copy and extract files
- Restart & Poweroff



Can you do this?

1. Use a login console over UART.
2. Use a SSH connection over WiFi.
3. Inspect and change pins.
4. Check what I^2C devices are present on the bus.
5. Copy files and extract zip files.
6. Correctly shutdown your Linux system.
7. Use a remote desktop connection.

Embedded Software Development under Linux

August 7, 2024

Python

“Python is an experiment in how much freedom programmers need. Too much freedom and nobody can read another’s code; too little and expressiveness is endangered.” – Guido van Rossum

August 7, 2024

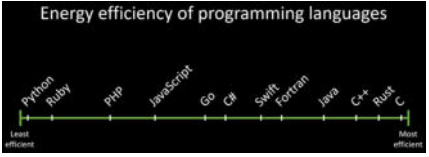
Python: Goals

- Python Overview
- Hello World with Python
- Debugging
- GPIO/LEDs with Python
- PWM (LED)
- Buttons
- I^2C

Python Programming Language

- **Interpreted** language, multi-paradigm (OOP, structured programming, . . .)
- Easy to use, lots of tutorials
- Scripts → Bytecode → Interpreter
- Can run Interpreter in interactive mode
- IDE: IDLE, Geany, Thonny, . . .
- Headless: nano
- Recommended readings
  - A Byte of Python
  - The Python Documentation

Source: Wikipedia



Source: The 10 most energy efficient programming languages

Interactive mode

```
$ python
```

- Command prompt: `>>>`
- `exit()` or `CTRL`+`D` (EOF) to exit Python Interpreter
- `print("text")` writes to stdout

```
pi@eee-04885:~ $ python
Python 3.11.2 (main, May  2 2024, 11:59:08) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hello!")
hello!
>>> exit()
pi@eee-04885:~ $
```

Hello World

```
# hello.py
print("hello world!")
```

- `#`: Comment
- `.py`: default file extension
- `print`: formatted output to `stdout` (console)

Create such a file.

Running Python Script

```
$ nano hello.py
```

```
print("hello world!")
```

Safe and exit editor: CTRL+X, y, ENTER

```
$ python hello.py
```

python3

```
pi@eee-00031:~/mse/python $ nano hello.py
pi@eee-00031:~/mse/python $ cat hello.py
print("hello world!")

pi@eee-00031:~/mse/python $ python hello.py
hello world!
pi@eee-00031:~/mse/python $
```

Virtual Environment

Install library packages:

```
$ pip install some-python-library
```

Problem: **Python Hell** with incompatible packages and libraries
Solution: Creation of virtual environments

Create environment (cd to folder first where to create it):

```
$ python -m venv my-venv
```

Install library package into environment:

```
$ my-venv/bin/pip install some-python-library
```

Run Python within environment:

```
$ my-venv/bin/python myProgram.py
```

Factorial Example

```
# Python program to calculate the factorial or n!: 1*2*3*...(n-1)*n
n = int(input('Type a number, and its factorial will be printed: '))

if n < 0:
  raise ValueError('You must enter a non-negative integer')

factorial = 1
for i in range(2, n + 1):
  factorial *= i

print(factorial)
```

Snippets: factorial.py

Debug

- `print()`-style 'debugging'
- pdb (The Python Debugger) Library

Add the line below, where the program shall be halted:

```
import pdb; pdb.set_trace()
```

- `h`: help
- `s`: step into
- `n`: next, step over
- `r`: return, step out of function
- `c`: continue
- `p` <var>: print variable
- `w`: where, print stack trace
- `l`: list source file
- `a`: args of current function
- `q`: quit

For Loops

```
for x in range(6):
  print(x)   # 0 to 5
```

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

```
for x in range(6):
  if x == 3: break
  print(x)
else: # executed by end of loop, not stopped by break
    print("Finally finished!")
```

While Loops

```
while True:
  print("endless")
```

```
i = 1
while i < 6:
  print(i)
  i += 1
```

```
i = 1
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

Input and Output

in text (h/out)

```
print(42)
number = 42; print(number)
message = input("Type a message to yourself: ")
print("You said:", message)

number = int(input("Type a number:"))
print("You entered:", number)
```

```
pi@eee-00031:~/mse/python $ python name.py
Enter your first name: John
Enter your last name: Doe
Full name: John Doe
```

GPIO: LED

```
import time
import RPi.GPIO as GPIO

led_pin = 21                        # GPIO pin used
GPIO.setmode(GPIO.BCM)              # Use "GPIO" pin numbering
GPIO.setup(led_pin, GPIO.OUT)       # Mux as output pin

for x in range(3):
  GPIO.output(led_pin, GPIO.HIGH) # Turn LED on
  time.sleep(1)                     # Delay for 1 second
  GPIO.output(led_pin, GPIO.LOW)  # Turn LED off
  time.sleep(0.5)                   # Delay for 0.5 second

GPIO.cleanup()
```

ModuleNotFoundError (rpi.gpio legacy installation):

```
$ my-venv/bin/pip install rpi.gpio
```

GPIO with PWM LED

```
import time                     # used for sleep()
import RPi.GPIO as GPIO         # alias to GPIO

led_pin = 21                    # Pin definitions
GPIO.setmode(GPIO.BCM)          # Use "GPIO" pin numbering
GPIO.setup(led_pin, GPIO.OUT)   # Set LED pin as output

pwm = GPIO.PWM(led_pin, 50)     # initialize with 50 Hz
pwm.start(0)                    # initial 0% duty

pwm.ChangeDutyCycle(50)         # change to 50%
time.sleep(2)                   # wait 2 seconds
pwm.ChangeDutyCycle(90)         # change duty to 90%
time.sleep(2)                   # wait 2 seconds

pwm.stop()                      # stop PWM
GPIO.cleanup()                  # cleanup and free pins
```

GPIO: Button

```
import time                     # used for sleep()
import RPi.GPIO as GPIO         # alias to GPIO

btn_pin = 26                    # push button pin
led_pin = 21                    # LED pin
GPIO.setmode(GPIO.BCM)          # using BCM numbers
GPIO.setup(btn_pin, GPIO.IN)    # button: input
# GPIO.setup(btn_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)   # or PUD_DOWN
GPIO.setup(led_pin, GPIO.OUT) # LED: output
try:
  while True:
    if GPIO.input(btn_pin):
      GPIO.output(led_pin, GPIO.LOW)
    else:
      GPIO.output(led_pin, GPIO.HIGH)

finally: # for ctrl+c, this will be called
  GPIO.cleanup()
```

I2C: SHT31

SMBus legacy installation:

```
$ my-venv/bin/pip install smbus
```

```
import time
import smbus

i2c_bus = smbus.SMBus(1) # Get I2C bus
i2c_address = 0x44       # SHT31 address

# start measurement command: 0x2c06
i2c_bus.write_i2c_block_data(i2c_address, 0x2C, [0x06])
time.sleep(0.5)  # wait 500 ms

# Read data back from 0x00(00), 6 bytes
# Temp MSB, Temp LSB, Temp CRC, Humididty MSB, Humidity LSB, Humidity CRC
data = i2c_bus.read_i2c_block_data(i2c_address, 0x00, 6)
```

interpretation of data

I2C: SHT31

```
# Convert the data
temp = data[0] * 256 + data[1]
cTemp = -45 + (175 * temp / 65535.0)
fTemp = -49 + (315 * temp / 65535.0)
humidity = 100 * (data[3] * 256 + data[4]) / 65535.0

# Output data to console
print("Temperature in Celsius: %.2f C" %cTemp)
print("Temperature in Fahrenheit: %.2f F" %fTemp)
print("Relative Humidity: %.2f %%RH" %humidity)
```

I2C: SSD1306

MSE

Using an I2C OLED Display Module with the Raspberry Pi:

```
$ sudo apt install -y git
```

```
$ sudo apt install -y python3-pil libjpeg-dev zlib1g-dev
libfreetype6-dev liblcms2-dev libopenjp2-7 -y
```

```
$ source my-venv/bin/activate
```

```
$ python -m pip install -upgrade luma.oled
```

```
$ mkdir oled && cd oled
```

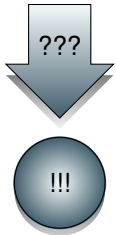
```
$ git clone https://github.com/rm-hull/luma.examples.git
```

```
$ cd luma.examples/examples
```

```
$ python 3d_box.py
```

Python: Summary

- Running Python
- Interactive
- Scripts
- Debugging
- Examples
  - GPIO (input, output)
  - PWM
  - I^2C: SHT, SSD1306, ...



Can you do this?

1. Implement a program which ask for your first and then your last name, and printing the full name.
2. Debug a Python program.
3. Toggle all the LED's on the Pico board.
4. Changing the brightness of a LED.
5. Check if a button is pressed and if yes, turn on a LED and write a message to the console.
6. Print temperature and humidity values from the I^2C sensor.
7. Display something on the OLED display.

Embedded Software Development under Linux

August 7, 2024

C

“There is a fight to be fought in every step we take and in every level we reach. This fight was started the moment we said ’Hello World’ with our first our first baby cry and it will end when we say ’ Goodbye World’ with our last breath of life.” – Euginia Herlihy

August 7, 2024

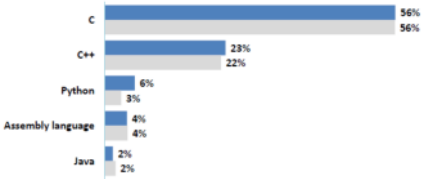
C: Goals

- First small program in C
- Building
- Running the program
- Printing a ’hello’ to the console
- Make and CMake refresher
- C quick refresher

C Programming Language

- Structured programming language
- Invented for UNIX → Linux
- Efficient, widely used for Linux and Embedded Applications
- Compiler → Object Files → Linker → Application
- ¬ OOP → C++

Source: Wiimedia



Source: EETimes

Main

```c
/* helloworld.c */
#include <stdio.h>
int main(int argc, char *argv[]) {
  printf("Hello world!\n");
  return 0;
}
```

- `/* */`: Comment
- `#include`: interface for `printf`
- `.c`: default source file extension, `.h` for interface files
- `main`: application entry point
- `argc`, `argv`: arguments for the program
- `printf`: formatted output to `stdout` (console)
- `0`: no error, otherwise $\neq 0$: error code

Compiling

```
$ g++ -ggdb hello.c -o hello
$ arm-linux-gnueabihf-g++ -ggdb hello.c -o hello
$ aarch64-linux-gnu-g++ -ggdb hello.c -o hello
```

- `g++`: C/C++ compiler, qualified for cross compiling
- `-ggdb`: generate debug information for GNU Debugger (gdb)
- `hello.c`: file to compile
- `-o hello`: output file to be created

> **!** Compile the source file.

Executing

```
$ file hello
```

```
hello: ELF 64-bit LSB pie executable, ARM aarch64,
version 1 (SYSV), dynamically linked,
interpreter /lib/ld-linux-aarch64.so.1,
BuildID[sha1]=c5b86da44795de907f821fcb392bed783981e98a,
for GNU/Linux 3.7.0, with debug_info, not stripped
```

- Executable **not** in the search path needs to use the current directory
- Security concern

```
$ ./hello
```

Run the program.

makefile - Simple

```
all: hello

CC=aarch64-linux-gnu-g++

clean:
  -rm -f hello hello.o

hello.o: hello.c
  $(CC) -g3 -c hello.c -o hello.o

hello: hello.o
  $(CC) hello.o -ggdb -o hello
```

```
$ make -f FILE [target]
```

```
$ make
```

```
$ make clean
```

```
$ make all
```

Use `TAB` in make files! Default file name: `makefile`!

Make - Multiple Files and Directories

```
...
LINK_TARGET = $(OBJ_DIR)program
...
SRC_DIR = ./
OBJ_DIR = ./
OBJS = \
  $(OBJ_DIR)extra.o \
  $(OBJ_DIR)main.o
############################
# List of dependency files
C_DEPS = \
  $(OBJ_DIR)extra.d \
  $(OBJ_DIR)main.d
...
all: $(LINK_TARGET)
  @echo All done!
...
```

include source files

include object files

CMake

Installation:

```
$ sudo apt install -y cmake
```

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.5)
project(hello C)

add_executable(hello main.c calc.c)
```

```
$ mkdir build
$ cd build
$ cmake ..
$ make
```

Functions

```
int i; /* global */
void inc(void) {
  i++
}
```

```
int multiply(int a, int b) {
  return a*b;
}
```

```
void myFunc(int a) {
  int t; /* local */

  t = multiply(3, a);
  i = 20;
  inc();
}
```

Loops

```c
int i = 0;
while(i < 10) {
  printf("%d\n", i);
  i++;
}
```

```c
for(int i=0; i<10; i++) {
  printf("%d\n", i);
}
```

```c
int i = 0;
do {
  printf("%d\n", i);
  i++;
} while(i<10);
```

if-else if-else

```
if (i==5) {
  i++;
}
```

```
if (i==0) {
  i++;
} else {
  i--;
}
```

```
if (i>10) {
  i++;
} else if (i<=4) {
  i--;
} else {
  i *= 2;
}
```

switch

```
int foo(int i) {
  int x = 0;
  switch(i) {
    case 0:
    case 1:
      x++:
      break;

    case 42:
      return x+1;

    default:
      break;
  }
  return x;
}
```

Variables

```
/* globals */
extern int var0; /* declaration, interface */

int var0; /* definition, implementation, initialized with 0 */

int foo(int i, signed char c, unsigned long l) { /* parameters */
  static int staticLocal = 5; /* static local variable (as global, but
      local scope) */
  int temp = 0x10; /* local variable */

  return temp;
}
```

Pointer

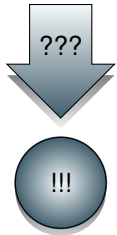
```
const int32_t *p; /* pointer to a constant int 32bit value */

int16_t **pp; /* pointer to a pointer to signed 16bit integer value */

int foo(int param0, char param1); /* function declaration */

int (*fp)(int, char) = foo; /* function pointer, initialized with foo */

fp(); /* call of the function (pointer) */
```

Dynamic Memory

```c
#include <stdlib.h> /* for malloc */
#include <string.h> /* for strcpy */

void foo(const char *str) {
  /* allocate new memory buffer with size of string */
  char *p = (char*)malloc(strlen(str)+1); /* plus one: zero byte */

  strcpy(p, str); /* copy string */
  /* do something with it */
  free(p);
}
```

C: Summary

- Created a first C program
- Compiling (and linking)
- Executing
- Output text on Console
- Basic C programming



Can you do this?

1. Write a C program to write your name 5 times on the console.
2. Use Make to build your project.
3. Create and build a ’hello world’ program with CMake.

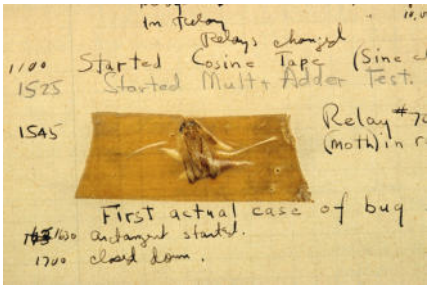
GNU Debugger

“The purpose of a debugger such as gdb is to allow you to see what is going on ’inside’ another program while it executes—or what another program was doing at the moment it crashed..” – Richard Stallman

August 7, 2024

GNU Debugger: Goals

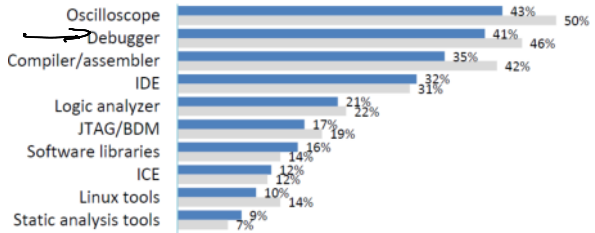
- **gdb**: GNU Debugger
- Architecture
- Debugging with command line and gdb
- Debug with VS Code from Host



Source: AmericanHistory

Debugger

- Debugger == De-Bug, remove bugs
- Download & program application binary to target
- Inspect system state (variables, registers, memory, ...)
- Attach to running target
- Stop, step, continue, terminate
- 30-50% of developing time spent with validating and debugging software (Source: ACM)
- Essential for larger projects



Source: EETimes

gdb © GNU

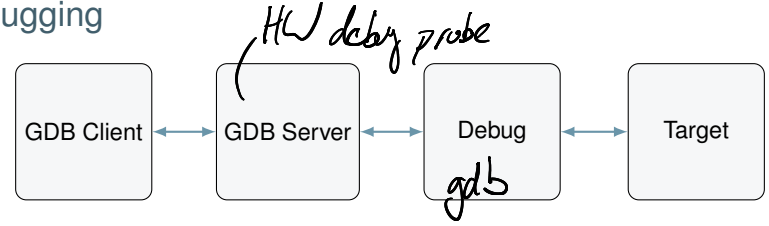
gdb

- gdb == GNU Debugger
- Client - Server architecture
- Command line interface (User)
- GDB MI (Machine Interface)

```
$ sudo apt install gdbserver
```

Source: Wikipedia

Remote Debugging



- Client-Server connection (TCP, Port)
- Example running server listening on port 1234 debugging the program 'hello':

```
$ gdbserver :1234 hello
```

> `ps -a`, then `kill -9 <pid>` kill process

- Example starting client in interactive mode:

```
$ gdb
```

GDB Remote Debug Session

GDB interactive session with remote target:

```
target remote 192.168.0.2:1234
file hello
break main
continue
quit
```

- `target remote`: Connection to remote target (hostname, IP address) and port
- `file`: load symbols from executable
- `break`: set breakpoint at function, program will stop
- `run`: start and run executable and stop on breakpoint
- `continue`: continue execution
- `quit`: exit debug session

GDB Local Debug Session

Running local gdb with program:

```
$ gdb hello
```

```
break main
run
```

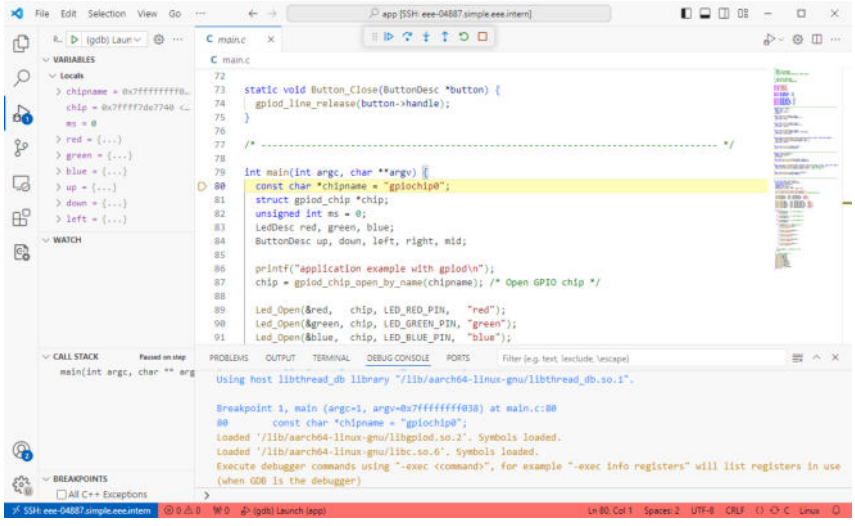
- Runs directly with the binary
- No external connection or configuration needed
- `break`: set breakpoint
- `run`: run executable

GDB Cheat Sheet

GDB Quick Reference on Moodle.

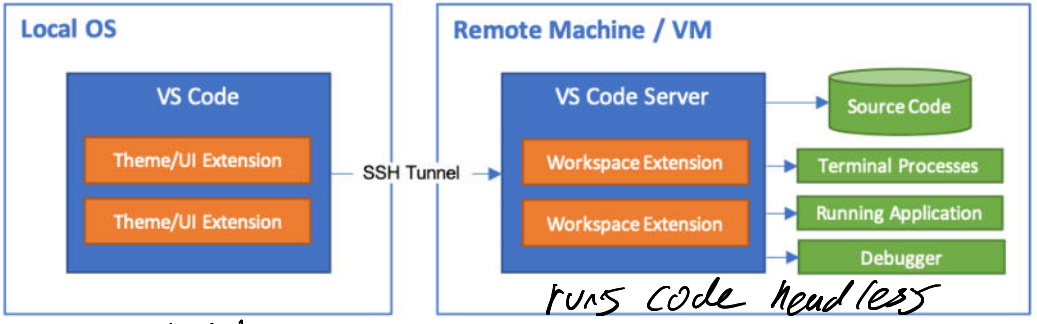
- `gdb <executable>`: Debug local executable
- `r`: Run
- `n`: Execute next line
- `c`: Continue running
- `info locals`: Show local (stack) variables
- `info breakpoints`: Show list of breakpoints
- `help`: Show help list
- `q`: Quit the program

GDB with VS Code



Remote Development with VS Code & SSH

- Microsoft VS Code: https://code.visualstudio.com



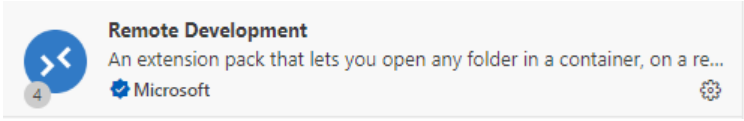
Source: Microsoft

Local Host
Windows

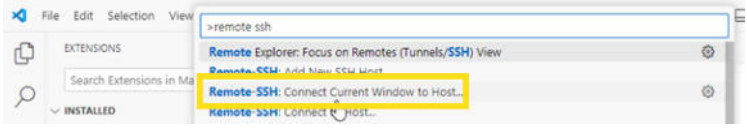
View ← Build & debug
in Localhost on Target in Linux

VS Code: SSH Connection

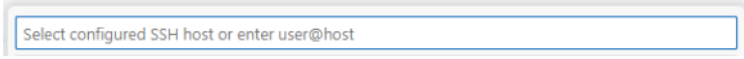
- Microsoft **Remote Development** Extension Pack from Marketplace:



- CTRL+SHIFT+P: **Remote SSH**: Connect Current Window to Host...



- Select host (pi@<host>)

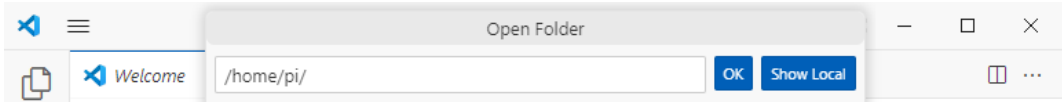


- Choose platform (Linux) and password
- Installs VS Code Server on remote (takes a while for the first time)

VS Code: Build

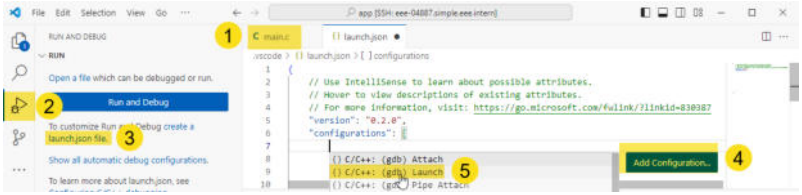
VS code has no Project, it has Folders

- Open 'hello' folder on remote host: Menu File → Open Folder ...



- Build with Terminal

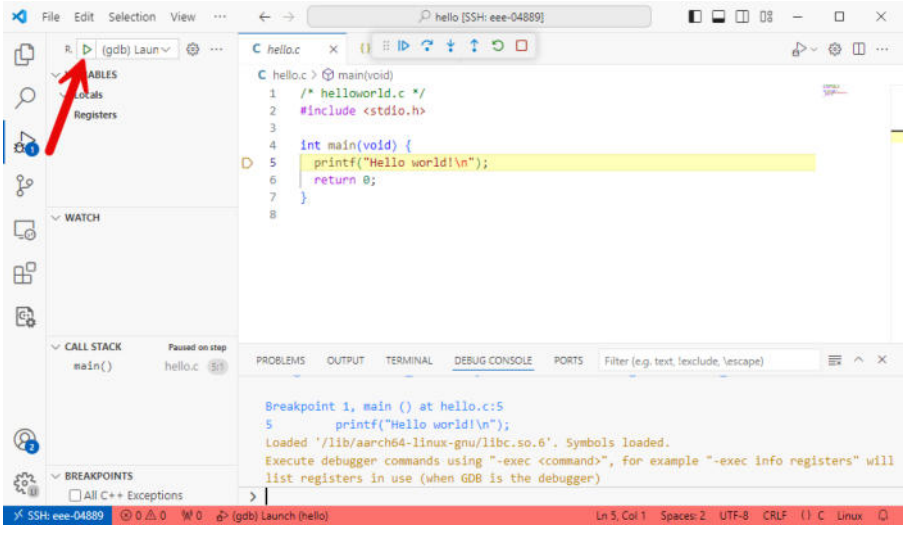
VS Code: Debugging



1. Open source file (e.g. `hello.c`) in editor
2. Open **Debug** Pane
3. **Create launch.json**
4. **Add Configuration**
5. **C/C++: (gdb) Launch**

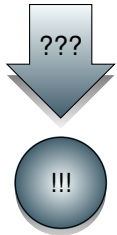
- Update program, e.g: `"program": "$workspaceFolder/hello",`
- `"stopAtEntry": true,`

VS Code: Debugging



GNU Debugger: Summary

- Reasons for a debugger
- GNU debugger (gdb)
- GDB local and remote debugging
- `.gdbinit`
- Simple debug sessions
- Killing processes
- Build & Debug with VS Code



Can you do this?

1. Use gdb to debug a C/C++ program.
2. Use breakpoints.
3. Step through your code.
4. Inspect variables.
5. Kill your program from another console.
6. Build and debug with VS Code.

Arguments

“My life is my argument.” – Albert Schweitzer

August 7, 2024

Arguments: Goals

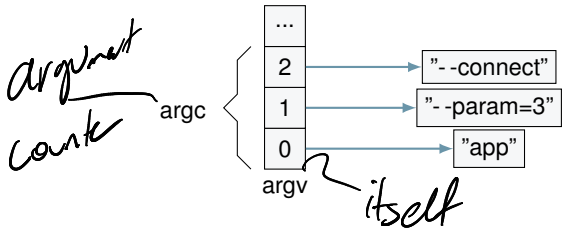
- Passing arguments to program
- Options, configuration, user input, ...
- Commandline
- Argument vector

Argument Vector

```
int main(int argc, char *argv[]);
```

- `argc`: **Argument counter**, number of arguments in vector
- `argv[]`: **Argument vector**, array of pointers to argument strings
- `argv[0]`: name of program

```
./app --param=3 --connect
```



Argument Vector

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
  int count;

  printf("argc: %d\n", argc);
  for (count = 0; count < argc; count++) {
    printf("argv[%d] = %s\n", count, argv[count]);
  }
  return  0;
}
```

Arguments

- Spaces
  - `mv "my file" "other file"`
  - `mv my\ file other\ file`
- Options
  - 'old', UNIX: `-c`
  - 'new', GNU: `--create`
  - Utility Conventions: `utility_name` `[-abcDxyz]` `[-p arg]` `[operand]`
- Simple: DIY with `strcmp()`

Simple Arg Parsing

```c
#include <stdio.h>  /* printf */
#include <string.h> /* strcmp */
#include <stdlib.h> /* atoi */

int main(int argc, char *argv[]) {
  int counter = 0;
  printf("usage: [-h] | [-c [num]]\n");
  if (argc==1) { /* no arguments */
  } else if (argc==2 && strcmp(argv[1], "-h")==0) { /* -h */
    return 0; /* ok */
  } else if (argc==2 && strcmp(argv[1], "-c")==0) { /* -c */
    counter = 1;
  } else if (argc==3 && strcmp(argv[1], "-c")==0) { /* -c num */
    counter = atoi(argv[2]);
    if (counter==0) { /* atoi() failed */
      return -1; /* failed */
    }
  } else {
    return -1; /* failed */
  }
  return  0;
}
```

getopt()

`getopt()` for e.g.: `app [-ir] [-x arg] [-f [arg]]`

```
int getopt(int argc, char *const argv[], const char *optstring);
   /*!< returns option character, or '?' */
extern char *optarg; /* returns option argument */
extern int opterr; /* set to 1 to supress internal error message */
extern int optind; /* option counter */
extern int optopt; /* option if missing or not recognized */
```

- `getopt()`: returns option character, `:` or `?`
  - `argc`, `argv`: received in `main()`
  - `optstring`: Option string
    - Single option character, starting with `-` → `"ir"` ⇒ `-i`, `-i -r`, `-ri`, `-r`
    - `:` at the start: return `:` instead of `?` if no argument is given
    - `:`: Option with argument → `"x:"` ⇒ `-x3`
    - `::`: Option with *optional* argument → `"f::"` ⇒ `-f`, `-f3`
- `getopt_long()`: Example of Parsing Long Options

getopt()

```
printf("usage: [-i] [-x arg] [-f [arg]]\n");
while((opt = getopt(argc, argv, ":ix:f::")) != -1) {
  switch(opt) {
    case 'i':
      printf("option -%c\n", opt);
      break;
    case 'x':
      printf("option -%c with argument: %s\n", opt, optarg);
      break;
    case 'f':
      if (optarg!=NULL) {
        printf("option -%c with argument: %s\n", opt, optarg);
      } else {
        printf("option -%c with default argument\n", opt);
      }
      break;
    ...
```

getopt()

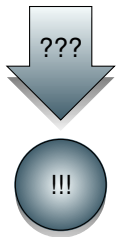
Build up a complex argument parse

```
  ...
  case ':':
    switch(optopt) {
      case 'x':
      fprintf(stderr, "option -%c is missing a required argument\n", optopt);
      break;
    } /* switch */
    break;
  case '?':
    printf("unknown option -%c\n", optopt);
    break;
    } /* switch */
  } /* while */
  for(; optind < argc; optind++) { /* optind is for the extra arguments which are
      not parsed */
    printf("extra arguments: %s\n", argv[optind]);
  }
  return 0;
}
```

Arguments: Summary

need to know how many entries it had

- Argument count: `argc`
- Argument vector: `argv`
- Arguments with spaces
- Simple argument parsing
- `getopt()`



Can you do this?

1. Understand the benefits and usage of `argc` and `argv`.
2. Implement a 'blinky'. Which LED to be used is passed as argument to the program.

GPIO

“All this sensory input, which begins in the brain, has its effect throughout the body.” – Norman Cousins

August 7, 2024

GPIO: Goals

- GPIO with `sysfs`
- GPIO with `char device`
- GPIO with `libgpiod`
- Electrical characteristics

GPIO with sysfs

- GPIO configuration and access through file system: 'everything is a file'
- Export a GPIO via `/sys/class/gpio/export`
- Configure the GPIO direction (input/output) via: `/sys/class/gpio/gpioX/direction`
- Read/write GPIO value via `/sys/class/gpio/gpioX/value`

Issues:

- Many GPIO read/write operation
- Limited configuration of hardware (pull resistor, open drain, active LOW/HIGH, ...)
- Possible race conditions
- Polling not reliable
- → `sysfs` considered as depreciated! ☹

GPIO with Character Device

From Kernel version 4.8:

- GPIO chip: controls multiple GPIO *lines*
- Character device `/dev/gpiochip0`
- `open()`, `read()`, `write()`, `close()`
- `ioctl()` to configure hardware specific settings

```
#define DEV_NAME "/dev/gpiochip0"

int fd, ret;
fd = open(DEV_NAME, O_RDONLY);
if (fd < 0) {
  printf("Unabled to open %s: %s", dev_name, strerror(errno));
  return;
}
/* control GPIO here, such as: configure, read, write, polling */
(void)close(fd);
```

GPIO with libgpiod

- Introduction to libgpiod, C **lib**rary and tools for interacting with the linux **GPIO** character **d**evice
- Utilities: `gpiodetect` `gpioinfo` `gpioget` `gpioset` `gpiomon` `gpiofind`

```
$ sudo apt install gpiod
```

- Get line (pin) 26 (button center) status of chip 0 and print it:

```
$ gpioget 0 26
```

- Set line 24 (led blue) of chip 0 to HIGH for 1 second and then release line:

```
$ gpioset --mode=time --sec=1 0 21=1
```

- Set line (pin) 26 (blue led) of chip 0 to low:

```
$ gpioset 0 26=1
```

- Development libraries:

```
$ sudo apt install libgpiod-dev
```

GPIO with libgpiod

```c
#include <gpiod.h>
```

```c
struct gpiod_chip *chip;
chip = gpiod_chip_open_by_name("gpiochip0"); /* Open GPIO chip */
...
gpiod_chip_close(chip); /* release chip */
```

```c
struct gpiod_line *line;
line = gpiod_chip_get_line(chip, 21); /* get GPIO line 21 */
...
gpiod_line_release(line); /* release line */
```

```c
gpiod_line_request_output(line, "myOutput", 0); /* open as output */
gpiod_line_set_value(line, 1); /* set line HIGH */
```

```c
gpiod_line_request_input(line, "myInput");
if (gpiod_line_get_value(line)==0) {
  /* line is LOW */
}
```

GPIO with libgpiod: Electrical characteristics

```
$ gpioget -bias=pull-up 0 26
```

- *GPIOD_LINE_REQUEST_FLAG_BIAS_DISABLE*: disabled
- *GPIOD_LINE_REQUEST_FLAG_OPEN_DRAIN*: output, to GND or open
- *GPIOD_LINE_REQUEST_FLAG_OPEN_SOURCE*: output: to Vdd or open
- *GPIOD_LINE_REQUEST_FLAG_BIAS_PULL* or *_DOWN*: Pull to GND or Vdd
- *GPIOD_LINE_REQUEST_FLAG_ACTIVE_LOW*: high is the default

```
struct gpiod_line_request_config config;

config.consumer = "button";
config.request_type = GPIOD_LINE_REQUEST_DIRECTION_INPUT;
config.flags = GPIOD_LINE_REQUEST_FLAG_BIAS_PULL_UP;
if (gpiod_line_request(button, &config, 0)!=0) {
  printf("failed requesting line with pull-up");
  return -1;
}
```

GPIO with libgpiod

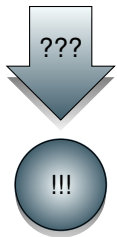
```c
#include <gpiod.h>
#include <stdio.h>

int main(int argc, char **argv) {
  struct gpiod_chip *chip;
  struct gpiod_line *led;    /* LED */
  struct gpiod_line *button; /* Pushbutton */
  int i, val;

  chip = gpiod_chip_open_by_name("gpiochip0"); /* Open GPIO chip */
  led = gpiod_chip_get_line(chip, 24); /* get LED line */
  button = gpiod_chip_get_line(chip, 6); /* get button line */
  gpiod_line_request_output(led, "led", 0); /* open as output */
  gpiod_line_request_input(button, "button"); /* open as input */
  ...
  gpiod_line_release(led); /* release led */
  gpiod_line_release(button); /* release button */
  gpiod_chip_close(chip); /* release chip */
  return 0;
}
```

Linking with `libgpiod.a`:

```
$(CC) $(OBJS) -ggdb -lgpiod -o $@
```

GPIO: Summary

- GPIO with *sysFS*
  → simple file I/O, possible race conditions, depreciated
- GPIO with *char device driver*
  → reliable, more complex
- GPIO with *libgpiod*
  → easy-to-use wrapper for char device driver
- Electrical characteristics: pull-up, pull-down, ...
  → depends on Hardware, not stable (yet)



Can you do this?

1. Understand different ways to use GPIO.
2. Implement a program using all LEDs and buttons using `libgpiod`: print button pressed on console and change LED.

I2C

“It is not enough to wire the world if you short-circuit the soul. Technology without heart is not enough.” – Tom Brokaw

August 7, 2024

I2C: Goals

- Access device with file I/O (Character device)
- Open Device
- Read/Write
- Close device
- Example: Sensirion sensor
- Example: OLED Display

I2C: SHT31

Open file handle:

```
char *filename = (char*)"/dev/i2c-1";
if ((file = open(filename, O_RDWR)) < 0) {
  printf("Failed to open the i2c bus");
  return -1;
}
```

Set I²C device address:

```
int addr = 0x44;  /* device address of SHT31 */
if (ioctl(file, I2C_SLAVE, addr) < 0) {
  printf("Failed to acquire bus access and/or talk to slave.\n");
  return -1;
}
```

Write data:

```
  char config[2] = {0};
  config[0] = 0x2C;
  config[1] = 0x06;
  write(file, config, sizeof(config)); /* send measurement command (0x2C06) */
  sleep(1); /* sensor needs some time */
```

I2C: SHT31

Read data:

```
char data[6] = {0}; /* 6 bytes of data: temp, humidity and CRC */
if(read(file, data, sizeof(data)) != sizeof(data)) {
  printf("Error: reading failed\n");
  exit(1);
}
```

Process data:

```
double cTemp = (((data[0] * 256) + data[1]) * 175.0) / 65535.0  - 45.0;
printf("Temperature in Celsius: %.2f C\n", cTemp);
```

Close file:

```
close(file);
```

I2C: OLED

- Uses part of McuLib: → Makefile project
- Generic I^2C and SSD1306 library
- `McuSSD1306`: Device driver, simple text output

Init:

```
McuI2cLib_Init();
McuSSD1306_Init();
```

Simple text:

```
McuSSD1306_Clear();
McuSSD1306_PrintString(0, 0, (uint8_t*)"Hello\nWorld!");
McuSSD1306_PrintString(3, 15, (uint8_t*)"The quick brown fox");
McuSSD1306_PrintString(4, 15, (uint8_t*)"jumps over");
McuSSD1306_PrintString(5, 15, (uint8_t*)"the lazy dog");
```

I2C: OLED

- `McuGDisplaySSD1306`: **g**raphic **display** function
- Double-buffer RAM for graphics and fonts: orientation, clear, line, circle, box, bitmaps, ...
- 'Update' function transfers data to display

```
int x = 0;
int y = 0;
int color = McuGDisplaySSD1306_COLOR_BLUE;

McuGDisplaySSD1306_Clear();
McuGDisplaySSD1306_DrawBox(x, y, McuGDisplaySSD1306_GetWidth()-1,
    McuGDisplaySSD1306_GetHeight()-1, 1, color);
McuGDisplaySSD1306_DrawCircle(x, y, 5, color);
McuGDisplaySSD1306_DrawLine(x, y, x+20, y+15, color);
McuGDisplaySSD1306_UpdateFull();
```

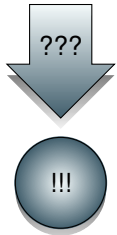
I2C: OLED

- `McuFontDisplay`: draw different bitmap fonts
- fonts: **Helv**etica, **Cour**ier
- size: 8, 10, 12, 14, 18, 24
- face: normal, bold
- `fonts` directory: McuFontHelv12Normal, McuFontCour18Bold, ...

```
PGFONT_Callbacks font = McuFontHelv12Normal_GetFont();
McuFontDisplay_PixelDim x, y;
int color = McuGDisplaySSD1306_COLOR_BLUE;

McuGDisplaySSD1306_Clear();
x = y = 0;
McuFontDisplay_WriteString((unsigned char*)"hello\nworld!", color, &x, &y
    , font);
McuGDisplaySSD1306_UpdateFull();
```

I2C: Summary

- File I/O for I^{2}C devices
- `open()` to open device
- `ioctl()` to select device address
- `write(buf)` to write data from address
- `write(addr)` → `read(data)`: to read from an address
- `close()` to close device
- OLED → McuLib



Can you do this?

1. Read sensor values and print on console.
2. Write sensor values on the OLED display.
3. Draw on the OLED display.
4. Use fonts to display text on the display.
5. Show temperature on the display.

Embedded Software Development under Linux

August 7, 2024

Agenda Day 3

0830 **Threading**

1000 Break

1030 **Networking**

1200 Lunch

1300 **Linux Kernel Module**

1500 Break

1530 **Lab Application**

1730 Wrapup

Threading

“Love weaves itself from hundreds of threads.” – David Levithan

August 7, 2024

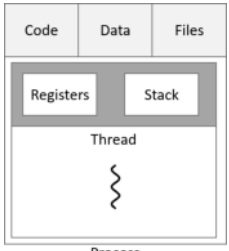
Threading: Goals

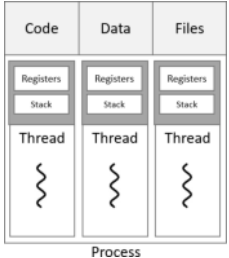
- Running multiple 'tasks' or 'threads'
- Relationship: Process vs. Thread
- Creating and running threads: POSIX `pthread`
- Concurrent access to data
- Thread synchronization

Processes and Threads

heavy process
to start exe.

- **Process**
  - Program under execution
  - Isolated execution → different address space (MMU)
  - `$ ps -ef`
- **Thread**
  - Lightweight process
  - Shared address space with parent
  - Spawns quickly
  - **L**ight-**W**eight **P**rocess (LWP)
  - `$ ps -eLf`
  - → `LWP`: Unique thread identifier inside a process
  - → `NLWP`: Number of threads for a given process





POSIX pthreads

```
#include <pthread.h>  /* POSIX (Portable Operating System Interface) threading */
```

```
int pthread_create(         /* create a new thread */
  pthread_t *thread,        /* thread handle */
  pthread_attr_t *attr,       /* thread attributes */
  void *(*start_routine)(void*),  /* thread function pointer */
  void *arg              /* arguments to pass to the thread */
  );
```

2 optional

```
int pthread_join(     /* block until thread terminates */
  pthread_t thread,   /* thread handle */
  void **value_ptr);  /* pointer to exit data from thread */
```

```
void pthread_exit(    /* exit thread */
  void *value_ptr); /* pointer to exit data to return */
```

Link with `libpthread.a`:

```
$(CC) $(OBJS) -ggdb -lpthread -o $@
```

Example pthread

```c
static int exitValue; /* used for pthread_exit() */

static void *myThread(void *arg) {
  printf("received: %d\n", *((int*)arg));
  exitValue++;
  pthread_exit(&exitValue); /* exit thread */
}

int main(void) {
  pthread_t thread; /* thread handle */
  int threadArg = 10; /* argument passed to thread */
  int *exitValuePtr; /* used to receive with pthread_exit() */
  if (pthread_create(&thread, NULL, myThread, &threadArg)!=0) {
    printf("failed creating thread\n");
    return -1;
  }
  if (pthread_join(thread, (void**)&exitValuePtr)!=0) { /* block */
    printf("failed joining thread");
    return -1;
  }
  printf("thread exit value: %d\n", *exitValuePtr);
  return 0;
}
```

exit thread with value

default exit value = 0
↳ exitValuePtr = exit value++

pthread race: single worker

```c
#define NOF_WORKERS (1)
static long value = 0;

void *worker(void *arg) {
  value = 42;
  pthread_exit(NULL);
}

int main(void) {
  pthread_t t;
  pthread_create(&t, NULL, worker, NULL);
  value = 43;
  pthread_join(t, NULL);
  printf("%d\n", value);
  return 0;
}
```

Expected result?

```
$ for i in {1..1000}; do ./pthreadRace; done | sort -n |
uniq -c
```

pthread race: double worker

Set `NOF_WORKERS` in pthreadRace `main.c` to have two workers:

```c
#define NOF_WORKERS (2)
void *worker0(void *x) {
  int v = value;
  value = v+1;
  pthread_exit(NULL);
}
```

```c
void *worker1(void *x) {
  int v = value;
  value = v-1;
  pthread_exit(NULL);
}
```

```c
int main(void) {
  pthread_t t[2];
  value = 42;
  pthread_create(&t[0], NULL, worker0, NULL);
  pthread_create(&t[1], NULL, worker1, NULL);
  pthread_join(t[0], NULL);
  pthread_join(t[1], NULL);
  printf("%d\n", value);
}
```

Expected result? What added `ADD_YIELD`?

pthread Mutex

mutex handle

Initialization of mutex:

```
int pthread_mutex_init( /* create a mutex */
  pthread_mutex_t *mutex,              /* mutex handle */
  const pthread_mutexattr_t *mutexattr /* optional attributes or NULL */
  );

int pthread_mutex_destroy(pthread_mutex_t *mutex); /* destry mutex */
```

Initialization:

```
static pthread_mutex_t my_mutex = PTHREAD_MUTEX_INITIALIZER;
```

Lock and unlock mutex: exclusive access

```
int pthread_mutex_lock(pthread_mutex_t *mutex); /* get lock */
int pthread_mutex_unlock(pthread_mutex_t *mutex); /* release lock */
int pthread_mutex_trylock(pthread_mutex_t *mutex); /* returns immediately, check
    return value */
```

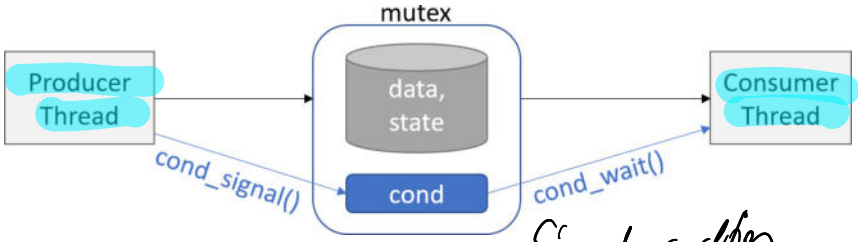
`pthread_mutex_trylock()`: Always check return code: `0` for OK!

check return!

pthread Mutex Example

```c
static pthread_mutex_t lock_shared; /* mutex */

static void *myThread(void *arg) {
  int id = *((int*)arg);
  printf("Task ID: %d\n", id);
  pthread_mutex_lock(&lock_shared); /* lock mutex */
  /* Critical Section, access to shared memory */
  pthread_mutex_unlock(&lock_shared); /* unlock mutex */
  pthread_exit(NULL); /* exit thread */
}

int main(void) {
  pthread_t thread0, thread1; /* thread handle */
  int id0 = 0, id1 = 1;       /* IDs for threads */

  pthread_mutex_init(&lock_shared, NULL); /* init mutex */

  pthread_create(&thread0, NULL, myThread, &id0);
  pthread_create(&thread1, NULL, myThread, &id1);

  pthread_join(thread0, NULL);
  pthread_join(thread1, NULL);
  return 0;
}
```

Consumer - Producer Threads



Signal condition

Consumer waits on Signal

- Single or multiple producer/consumer threads
- Data transfer with signaling of data
- **data**: Variable, Queue, List, Stack, . . .
- **state**: State of data, full, empty, noflItems, . . .
- **Critical Section** for concurrent access → `pthread` `Mutex`
- **Signaling of data arrived** → `pthread` `Condition Variable`

pthread Condition Variables

MSE

Initialization:

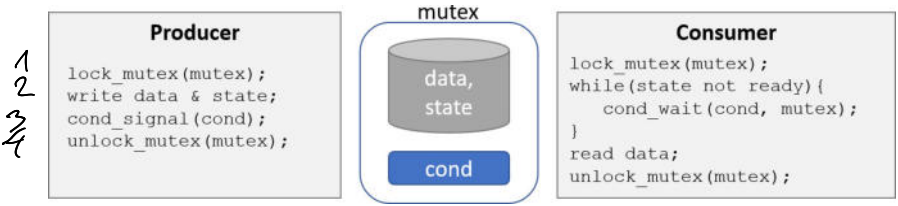
```
int pthread_cond_init(pthread_cond_t *cond, pthread_condattr_t *cond_attr);
static pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

Wait, signal or broadcast:

```
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
```

- `pthread_cond_wait`: Wait (sleep) for a signal inside a mutex, mutex will be temporarily released
- `pthread_cond_signal`: Send a signal to one thread waiting for the signal.
- `pthread_cond_broadcast`: Send a signal to all threads waiting for the signal.

pthread Condition Variable Usage



- Condition and Mutex need to be used as a **pair**
- `pthread_cond_wait()`: unlocks the mutex before putting the thread to sleep, locks it again during wakeup
- **Spurious wakeups**: Could be other reasons for wakeup, state could change before thread returns → need to recheck `cond`!
- `while` loop with testing `state` and checking `cond`

pthread Condition Variable Usage

```c
static pthread_mutex_t buffer_mutex; /* mutex for buffer */
static pthread_cond_t buffer_cond; /* condition for buffer */
static struct {
  bool isReady;
  int data;
} buffer;

#define NOF_DATA   (10)
```

```c
static void *producer(void *arg) {
  for (int i=0; i<NOF_DATA; i++) {
    pthread_mutex_lock(&buffer_mutex); /* lock mutex */
    buffer.data = i;
    buffer.isReady = true;
    pthread_cond_signal(&buffer_cond);
    pthread_mutex_unlock(&buffer_mutex); /* unlock mutex */
    sleep(1); wants to see.
  }
  pthread_exit(NULL); /* exit thread */
}
```

big delay

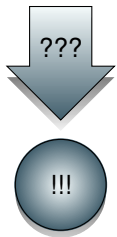
pthread Condition Variable Usage

```
static void *consumer(void *arg) {
  printf("starting consumer\n");
  for (int cntr=0;cntr<NOF_DATA;cntr++) {
    pthread_mutex_lock(&buffer_mutex); /* lock mutex */
    while(!buffer.isReady) {
      pthread_cond_wait(&buffer_cond, &buffer_mutex);
    }
    buffer.isReady = false;
    pthread_mutex_unlock(&buffer_mutex); /* unlock mutex */
  }
  pthread_exit(NULL); /* exit thread */
}
```

Verify functionality. Challenge: get rid of `sleep()`?

Threading: Summary

- Process vs. Thread
- POSIX
- `pthread`: create, join, exit
- **Mutex**: create, lock, unlock
- **Condition Variable**: create, signal, wait
- Other topics or references
  - `pthread_barrier_wait()`: synchronize a number of threads
  - `pthread_kill()`: signal a thread to terminate
  - `pthread_self()`: get a handle of the parent thread



Can you do this?

1. `pthread`: from `main()`, create two threads running the `myThread()` code.
2. `pthreadMutex`: Implement a clock (hh::ss), one thread (clock) incrementing clock every second, another thread (clockprint) printing the time. What could be a reentrancy issue?
3. `pthreadCond`: Extend the clock with an extra thread (tick) which sends the seconds 'tick' to the 'clock' thread.

Embedded Software Development under Linux

August 7, 2024

Networking

“The ARPAnet was the first transcontinental, high-speed computer network.” – Eric S. Raymond

August 7, 2024

Networking: Goals

Client Server Architecture

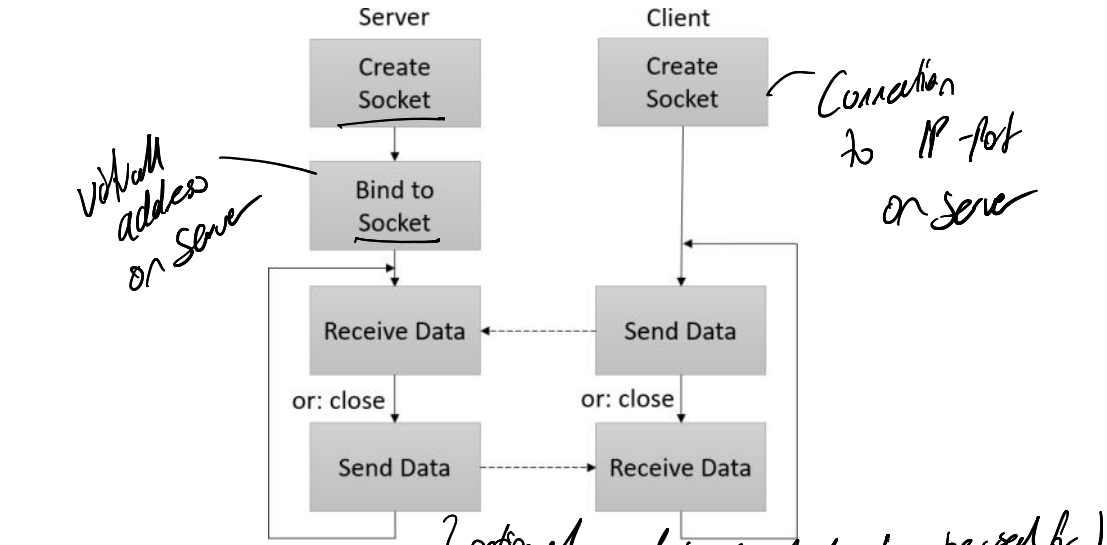
UDP/TCP

- User Datagram Protocol
- Client - Server
- Sending Data
- Receiving Data
- Timeout
- Host name $\rightarrow$ IP address

UDP

- **UDP**: **U**ser **D**atagram **P**rotocol
- Messages oriented transport layer protocol over **IP** (**I**nternet **P**rotocol)
- `UDP-Datagramm`:
  $\texttt{SrcPort}_{16b} + \texttt{DstPort}_{16b} + \texttt{Length}_{16b} + \texttt{CheckSum}_{16b} + \texttt{Data}$
- IP-Datagramm $= \texttt{IP-Header}_{96b\ (IPv4)} + \texttt{UDP-Datagramm}$
- IPv4 Header: $\texttt{SrcIP}_{32b} + \texttt{DstIP}_{32b} + 0_{8b} + \texttt{P-ID}_{8b} + \texttt{Length}_{16b}$
- 'Unreliable connection-less protocol'
  - No acknowledge (fire & forget)
  - No guarantee of packet order, no flow control
  - Transaction oriented: Request → Response
  - Stateless, ideal for multiple clients
  - Faster and more efficient than TCP (no checks, 8 Byte Header vs. 20 Bytes TCP)
  - Ideal for realtime applications
- Used for: DNS, DHCP, VoIP, ...

UDP Client-Server



UDP Server

Includes:

```
#include <netdb.h>        /* network database library */
#include <sys/socket.h>   /* sockets */
#include <arpa/inet.h>    /* address conversions */
```

Creating socket:

```
int fd;          /* socket file descriptor */
fd = socket(AF_INET/*domain*/, SOCK_DGRAM/*type*/, 0/*protocol*/);
```

Do the binding:

```
struct sockaddr_in myaddr;  /* our address */
const int port = 1234; /* port used for socket */
myaddr.sin_family = AF_INET; /* Internet protocol */
myaddr.sin_addr.s_addr = htonl(INADDR_ANY); /* accept all incoming */
myaddr.sin_port = htons(port);  /* port number for socket */
bind(fd, (struct sockaddr *)&myaddr, sizeof(myaddr);
```

all ports > 1000, ok

2 size: could be different

Closing socket:

```
close(fd);
```

UDP Server: Send and Receive

MSE

Receiving (blocking):

```
int recvlen;       /* # bytes received */
struct sockaddr_in remaddr; /* remote address */

socklen_t addrlen = sizeof(remaddr);     /* length of addresses */
recvlen = recvfrom(fd, buf, sizeof(buf), 0 /*flags*/, (struct sockaddr*)&remaddr,
    &addrlen);
```

Sending back response (optional):

```
sendto(fd, "ok!", strlen((char*)"ok!"), 0 /*flags*/, (struct sockaddr *)&remaddr,
    addrlen);
```

`udpServer` example.

UDP Client

Build of address for the remote:

```
struct sockaddr_in remaddr; /* remote address */
socklen_t slen=sizeof(remaddr);
const char *host = "192.168.1.150"; /* IP of host */
remaddr.sin_family = AF_INET;
remaddr.sin_port = htons(port);
inet_aton(host, &remaddr.sin_addr);
```

change ip for a UDP Server

Sending data:

```
sendto(fd, "hello!", strlen("hello!"), 0, (struct sockaddr *)&remaddr, slen);
```

Receiving response (optional):

```
int recvlen;    /* # bytes in acknowledgment message */
recvlen = recvfrom(fd, buf, sizeof(buf), 0, (struct sockaddr *)&remaddr, &slen);
```

! `udpClient` example. Change host IP address!

Server with Timeout

- `recvfrom()` is blocking
- Can set timeout (number of secs plus μsec)
- Configure after binding

```
struct timeval tv;

tv.tv_sec = 5;
tv.tv_usec = 0;
if (setsockopt(fd, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv)) < 0) {
  perror("Error");
}
```

`timeout` example.

Hostname to IP Address

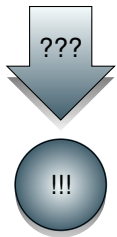
```c
int hostname_to_ip(const char *hostname, char *ipBuf, size_t ipBufLen) {
  struct hostent *he;
  struct in_addr **addr_list;
  int i;

  /* get the host info */
  if ((he = gethostbyname(hostname)) == NULL) {
    return -1; /* failed */
  }
  addr_list = (struct in_addr **) he->h_addr_list;
  for(i = 0; addr_list[i]!=NULL; i++) {
    /* Return the first one */
    strncpy(ipBuf,  inet_ntoa(*addr_list[i]), ipBufLen);
    ipBuf[ipBufLen-1] = '\0'; /* terminate string */
    return 0; /* ok */
  }
  return -1; /* failed */
}
```

`hostname` example.

can be used with localhost

Networking: Summary

- UDP as simple communication protocol
- Client & Server application
- Send → Response
- Receive → Response



Focus on UDP, TCP examples in the snippets☺.

Can you do this?

1. Ask for a UDP packet from a server.
2. Use a command line interface to connect a client to a server.
3. Use a name and DNS to connect to a server.
4. Use a client which asks the server for some data, using a timeout.

Embedded Software Development under Linux

August 8, 2024

Kernel Modules

“Your problem is another’s solution; Your solution will be his problem.” – Unknown

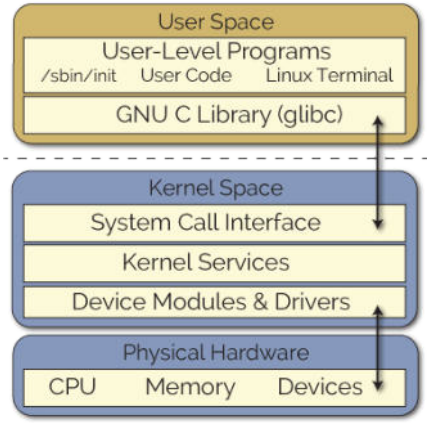
August 8, 2024

Kernel Modules: Goals

- Kernel Modules
  - Simple Kernel Module
  - GPIO
  - Interrupts
  - Threads
  - Parameter

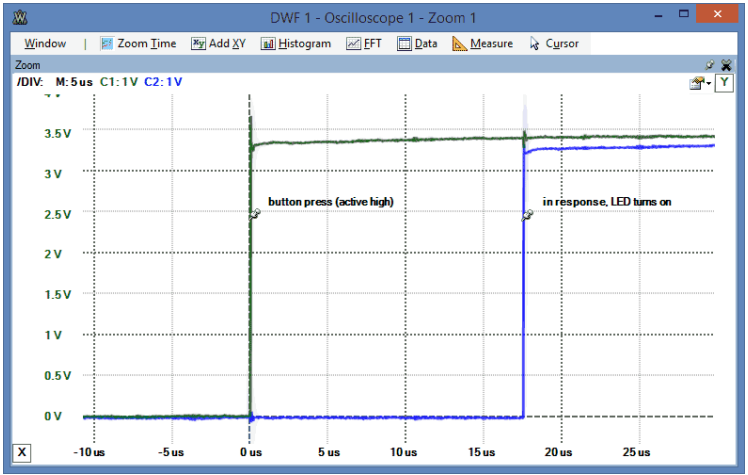
Kernel Modules

- Linux Kernel: User Space & Kernel Space
- Calling Kernel functions using System Calls
- Recommended: Run applications in User Space!
- Kernel Module
  - Extends the Kernel
  - Useful for hardware related drivers
  - non-sequential: Event/Hook driven
  - `Init` ⇒ `Exit`
  - No `printf()` or user space library usage allowed!
  - Higher execution priorities
  - Interrupt support
  - No direct FPU support



Source: Writing a Linux Kernel Module — Part 1: Introduction)

Kernel Modules



Source: Writing a Linux Kernel Module — Part 1: Introduction

Kernel Modules

- Linux: Monolithic Kernel
- **Kernel Module**: Load and unload at runtime
- Typical file ending: `.ko`
- Running in Kernel Space ⇒ Ideal for hardware drivers
- Default Folder: `/lib/modules/<KernelVersion>`
- Kernel Version: (print system information: Kernel Release)

```
$ uname -r
```

- Loading of custom kernel modules at boot time (Be careful!)
  - Add module into `/etc/modules` folder (without `.ko` extension)
  - Or: place module inside own folder:
    `/lib/modules/`uname -r`/kernel/drivers`
  - `depmod`: Generate modules.dep and map files

Kernel Modules

- List of loaded modules:

```
$ cat /proc/modules
```

```
$ lsmod
```

- Show information about module:

```
$ modinfo module.ko
```

- Load module:

```
$ sudo insmod module.ko
```

```
$ sudo insmod module.ko params
```

- Unload a loaded module:

```
$ sudo rmmod module
```

Kernel Headers

Install kernel header files:

```
$ sudo apt-get install raspberrypi-kernel-headers
$ sudo reboot
```

Check for `build` folder in:

```
$ ls /lib/modules/`uname -r`
```

Simple Module

Simple kernel module: `simple_km.c`

```c
#include <linux/module.h>    /* Needed by all modules */
#include <linux/kernel.h>    /* Needed for KERN_INFO */

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Erich Styger <erich.styger@hslu.ch>");
MODULE_DESCRIPTION("Simple kernel module");

static int __init simple_init_module(void) {
  printk(KERN_INFO "Init my kernel module.\n");
  return 0; /* non-zero means failed, module cannot be loaded */
}

static void __exit simple_cleanup_module(void) {
  printk(KERN_INFO "cleanup my kernel module.\n");
}

module_init(simple_init_module);
module_exit(simple_cleanup_module);
```

Simple Module

File `Makefile`:

```
PATH_TO_KERNEL_SOURCES = /lib/modules/$(shell uname -r)/build
PWD = $(shell pwd)

obj-m += simple_km.o

all:
  make -C $(PATH_TO_KERNEL_SOURCES) M=$(PWD) modules

clean:
  make -C $(PATH_TO_KERNEL_SOURCES) M=$(PWD) clean
```

- Build:
  ```
  $ make
  ```
- Generates module with `.ko` extension
- Clean:
  ```
  $ make clean
  ```
- Show file information
  ```
  $ modinfo simple_km.ko
  ```

Lab: Simple Kernel Module

- Create a simple kernel module
- Update/change module information
- Build it
- Inspect it: `modinfo`
- Load it: `insmod`, `lsmod`
- Check Kernel log: `dmesg | tail`
- Unload module: `rmmod`
- Check Kernel log again: `dmesg`

> **!** Use example `simple_km`.

GPIO

Interface:

```
#include <linux/gpio.h>     /* GPIO interface */
```

Kernel Module Init:

```
#define LED_BLUE_PIN_BCM  21 /* GPIO21, blue LED */
gpio_request(LED_BLUE_PIN_BCM, "LED_BLUE");
gpio_direction_output(LED_BLUE_PIN_BCM, 0);
```

Usage:

```
gpio_set_value(LED_BLUE_PIN_BCM, 1); /* on */
```

Kernel Module Exit:

```
gpio_free(LED_BLUE_PIN_BCM); /* release pin during exit */
```

Kernel GPIO with Kernel 6.1

6.6 @ raspi 4

MSE

With Kernel 6.1 or later (Bookworm), GPIO numbers in the Kernel must use the device tree.

Workaround:

- Get GPIO information

how Kernel interacts with GPIO

```
$ cat /sys/kernel/debug/gpio | grep GPIO21
```

- gives: `gpio-533 (GPIO21 )`
- Use number:

for users

```
#define LED_PIN_BCM    533 /* blue LED, gpio-533 for GPIO21 */
```

Dependency on pin numbers.

Device Tree Source File

describes devices

Device Tree with **overlay** for GPIO21 (0x15): `blue-led-gpio.dts`

```
/dts-v1/;
/plugin/;
/ {
  compatible="brcm,brcm2835";
  fragment@0 {
    target = <&gpio>;
    __overlay__ {
      blue_led_gpio:
          blue_led_gpio_conf {
        brcm,pins = <0x15>;
        brcm,function = <0x1>;
        brcm,pull = <0x1>;
      };
    };
  };
  ...
```

```
  ...
  fragment@1 {
    target-path = "/";
    __overlay__ {
      blue_led: blue_led_gpio {
        gpios = <&gpio 0x15 0x0>;
        compatible = "blue_led";
        status = "ok";
        pinctrl-0 = <&blue_led_gpio>;
        pinctrl-names = "default";
      };
    };
  };
};
```

Adding Overlay with Device Tree

- Compile device tree file to device tree blob:

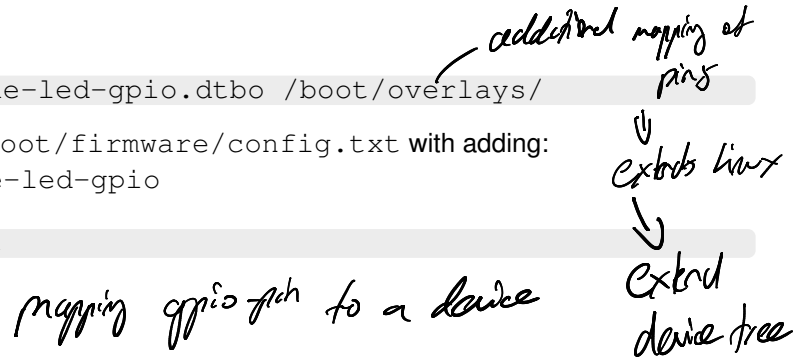
```
$ dtc -@ -I dts -O dtb -o blue-led-gpio.dtbo
blue-led-gpio.dts
```

- Copy to overlays:

```
$ sudo cp blue-led-gpio.dtbo /boot/overlays/
```

- Enable overlay in `/boot/firmware/config.txt` with adding:
  `dtoverlay=blue-led-gpio`

-
```
$ sudo reboot
```



Entries in Device Tree

list device tree

```
$ dtc -I fs /proc/device-tree | less
```

```
...
gpio@7e200000 {
  ...
  blue_led_gpio_conf {
    brcm,pull = <0x01>;
    brcm,function = <0x01>;
    phandle = <0xf6>;
    brcm,pins = <0x15>;
  };
  ...
```

```
/{
...
blue_led_gpio {
  pinctrl-names = "default";
  pinctrl-0 = <0xf6>;
  compatible = "blue_led";
  status = "ok";
  phandle = <0xf7>;
  gpios = <0x07 0x15 0x00>;
};
...
```

Driver with Device Tree

```
static struct of_device_id match_table[] = {
  {.compatible = "blue_led"},
  {/* end node */}
};

static struct platform_driver blueLed_driver = {
  .probe = gpio_init_probe,
  .remove = gpio_exit_remove,
  .driver = {
    .name = "blue_led_driver",
    .owner = THIS_MODULE,
    .of_match_table = match_table,
  }
};

module_platform_driver(blueLed_driver);
```

match blue LED

Init and Exit with gpiod

```c
#include <linux/of_device.h>
#include <linux/gpio/consumer.h>

struct gpio_desc *blue;

static int gpio_init_probe(struct platform_device *pdev) {
  blue = devm_gpiod_get(&pdev->dev, NULL, GPIOD_OUT_LOW); /* default
      output low */
  gpiod_set_value(blue, 1); /* high */
  return 0;
}

static int gpio_exit_remove(struct platform_device *pdev) {
  gpiod_put(blue); /* dispose and free descriptor */
  return 0;
}
```

Lab: Kernel Module with GPIO

- Extend existing module
- Init: Initialize LED and turn it on
- Exit: Turn off LED and free GPIO pin again
- Check functionality and kernel log

> **!** Use example `gpio_km`.

Interrupts

```
#include <linux/interrupt.h> /* IRQ interface */

static unsigned int gpioButton = 26; /* GPIO26: center button  */
static int irqNumber;          /* used to share the IRQ number */
static int numberPresses = 0;  /* store number of button presses */
```

Init:

```
gpio_direction_input(gpioButton);     // configure as input
irqNumber = gpio_to_irq(gpioButton); // get IRQ number
result = request_irq(irqNumber,  // interrupt number
          my_gpio_irq_handler,   // handler
          IRQF_TRIGGER_FALLING,  // falling edge
          "button_handler",      // used in /proc/interrupts
          NULL);                 // the *dev_id for shared lines
```

Exit:

```
free_irq(irqNumber, NULL); // free the IRQ number
gpio_free(gpioButton);     // unexport the Button GPIO
```

Interrupts

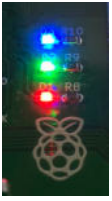
```
static irqreturn_t my_gpio_irq_handler(int irq, void *dev_id)
{
   printk(KERN_INFO "Button Interrupt: Button state: %d\n",
       gpio_get_value(gpioButton));
   ledOn = !ledOn;                    // Toggle LED state on button press
   gpio_set_value(gpioLED, ledOn); // Set the physical LED accordingly
   numberPresses++;                   // count number of presses
   return IRQ_HANDLED;                // IRQ was handled correctly
}
```

List of interrupts:

```
$ cat /proc/interrupts
```

Lab: Kernel Module with Interrupts

- Extend kernel module
- Generate interrupt with push button/switch
- Toggle the LED in the interrupt
- Count the number of interupts and log it to the kernel log
- Inspect `/proc/interrupts`
- Inspect kernel log



> **!** Use example `interrupt_km`.

Threads

Interface:

```
#include <linux/kthread.h>
#include <linux/delay.h>
```

Task Handle:

```
static struct task_struct *taskHndl;
```

Kernel Module Init:

```
int myThread(void *pv); /* prototype */
taskHndl = kthread_run(myThread, NULL, "myThread");
```

Kernel Module Exit:

```
kthread_stop(taskHndl);
```

Threads

```
static int myThread(void *data){
  for(;;) {
    if (kthread_should_stop()) {
      break;
    }
    doWork();
    msleep(1000);
  }
  return 0;
}
```

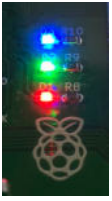
endless loop -> in Kernel Mode Critical

- `ndelay()`, `udelay()`, `mdelay()`: ⇒ xsleep() verwenden!
- `usleep_range(min, max)`
- `msleep(ms)` → milli second !
- `msleep_interruptible(ms)`:
- `ssleep(sec)`

don't use seconds

Lab: Kernel Module with Threads

- Extend kernel module
- Create thread
- Implement thread
- Blink LED in thread
- Use of `sleep()`



> **!** Use example `thread_km`.

Parameter

Parameter definition with name, type and access rights:

```
static char *name = "unknown";
static int count = 1;
module_param(name, charp, S_IRUGO);
MODULE_PARM_DESC(name, " Module name (default=unknown)");
module_param(count, int, S_IRUGO);
MODULE_PARM_DESC(count, " Count (default=1)");
```

Usage:

```
static int __init simple_module_init(void) {
  int i;
  printk(KERN_INFO "module loaded. name='%s' count=%d\n", name, count);
  for (i = 0; i < count; ++i) {
    printk(KERN_INFO " i=%d / %d\n", i, count);
  }
  return 0;
}
```

Installation:

```
$ insmod module.ko name=John count=10
```

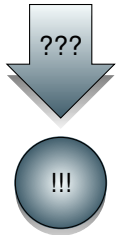
Lab: Kernel Module with Parameters

- Extend kernel module
- GPIO as LED Pin
- Task to blink the LED
- Parameters for GPIO number and blink frequency
- Example:
  - `ledPinNr=533`: Pin number
  - `LedMs=1000`: Period, milliseconds, (2-2000)

Use example `param_km`.

Summary

- User Space vs. Kernel Space
- Kernel Module
  - Creating and building kernel modules
  - load, inspect, unload
  - GPIO
  - Interrupts
  - Threads
  - Parameter



Can you do this?

1. Detect push button and toggle different LEDs.
2. Use a thread to blink a LED.
3. Use a parameter to select the LED to be used.

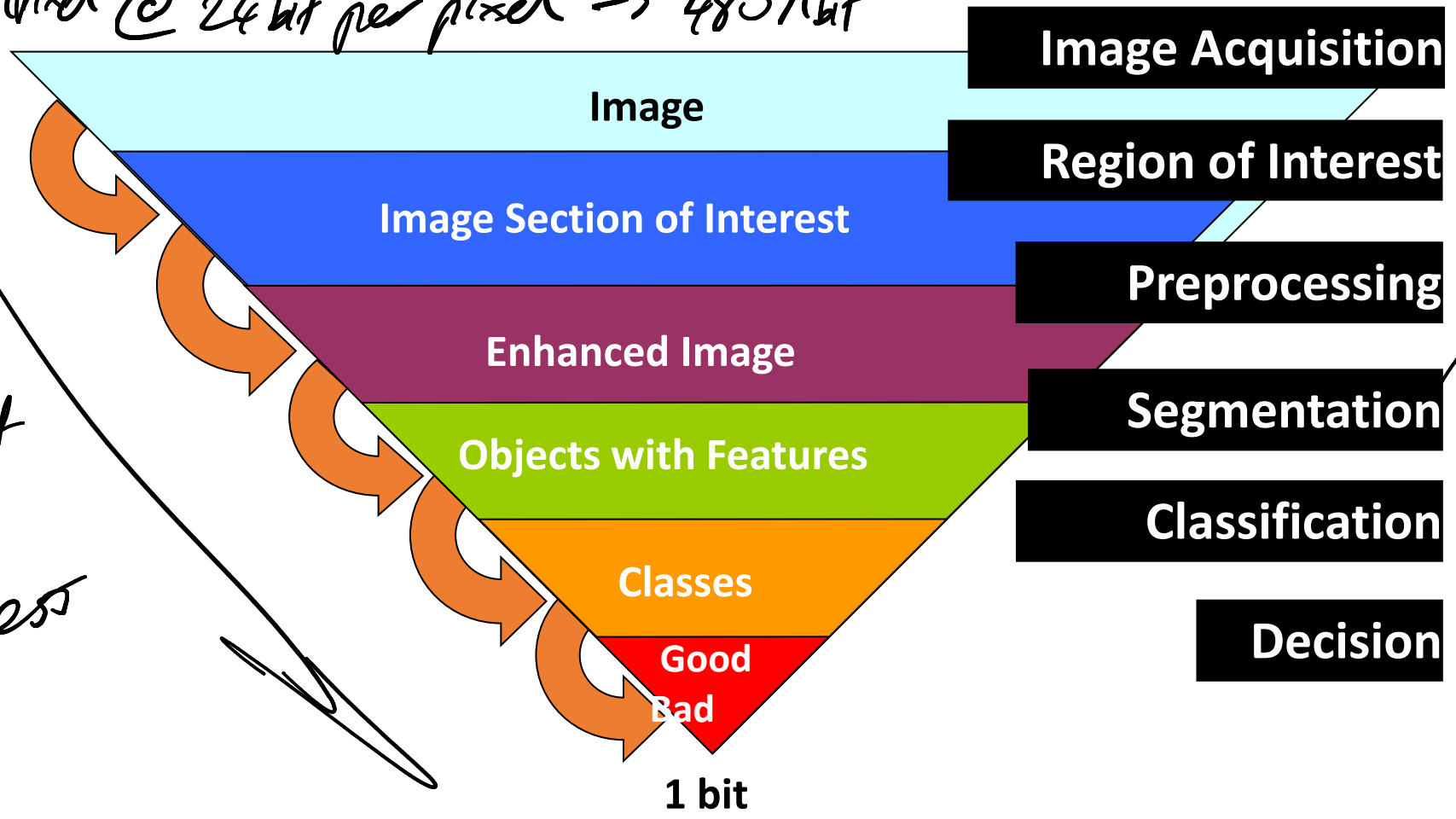
Image Acquisition and Processing with OpenCV on a Raspi

Carlo.Bach@ost.ch / Rjano.Ryser@ost.ch

Outline

- Short Introduction to Machine Vision and Image Processing
- Image Acquisition on the Raspi
- Image Processing with OpenCV

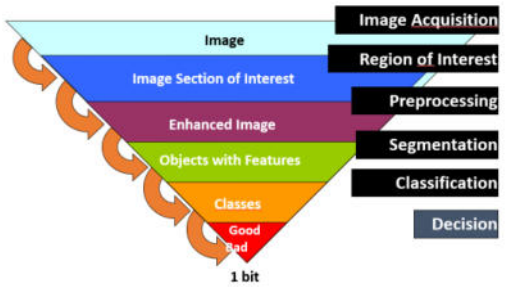
Typical Process Model in Image Processing



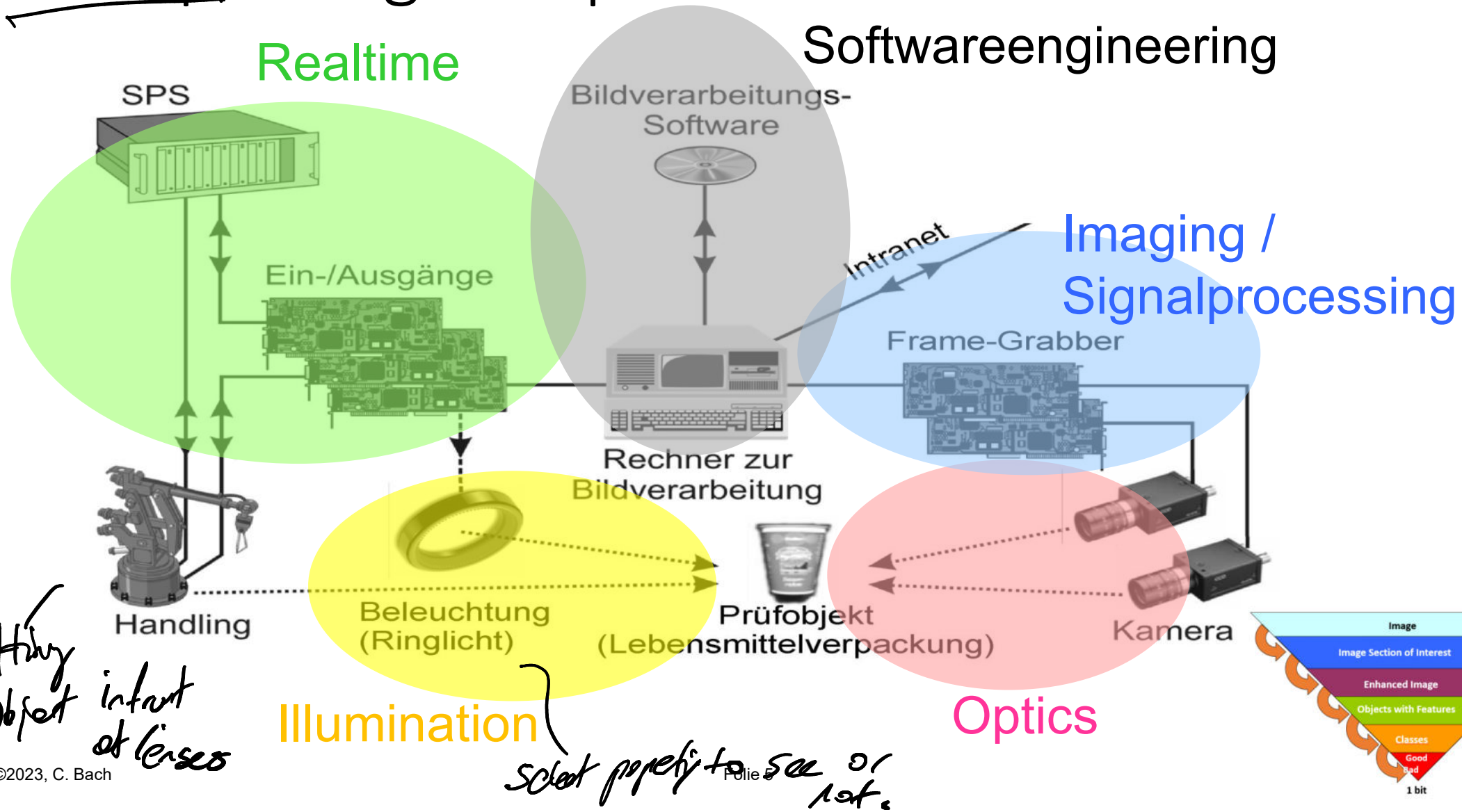
Example: Are there 6.15 Fr on the table here?

Task to verify amount of Money?

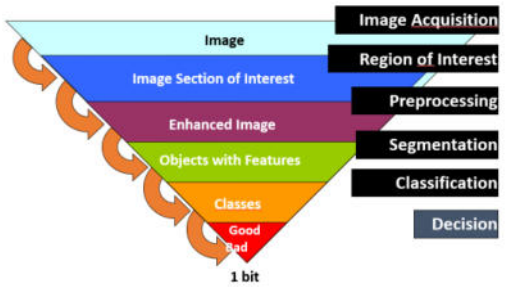
Payment accepted



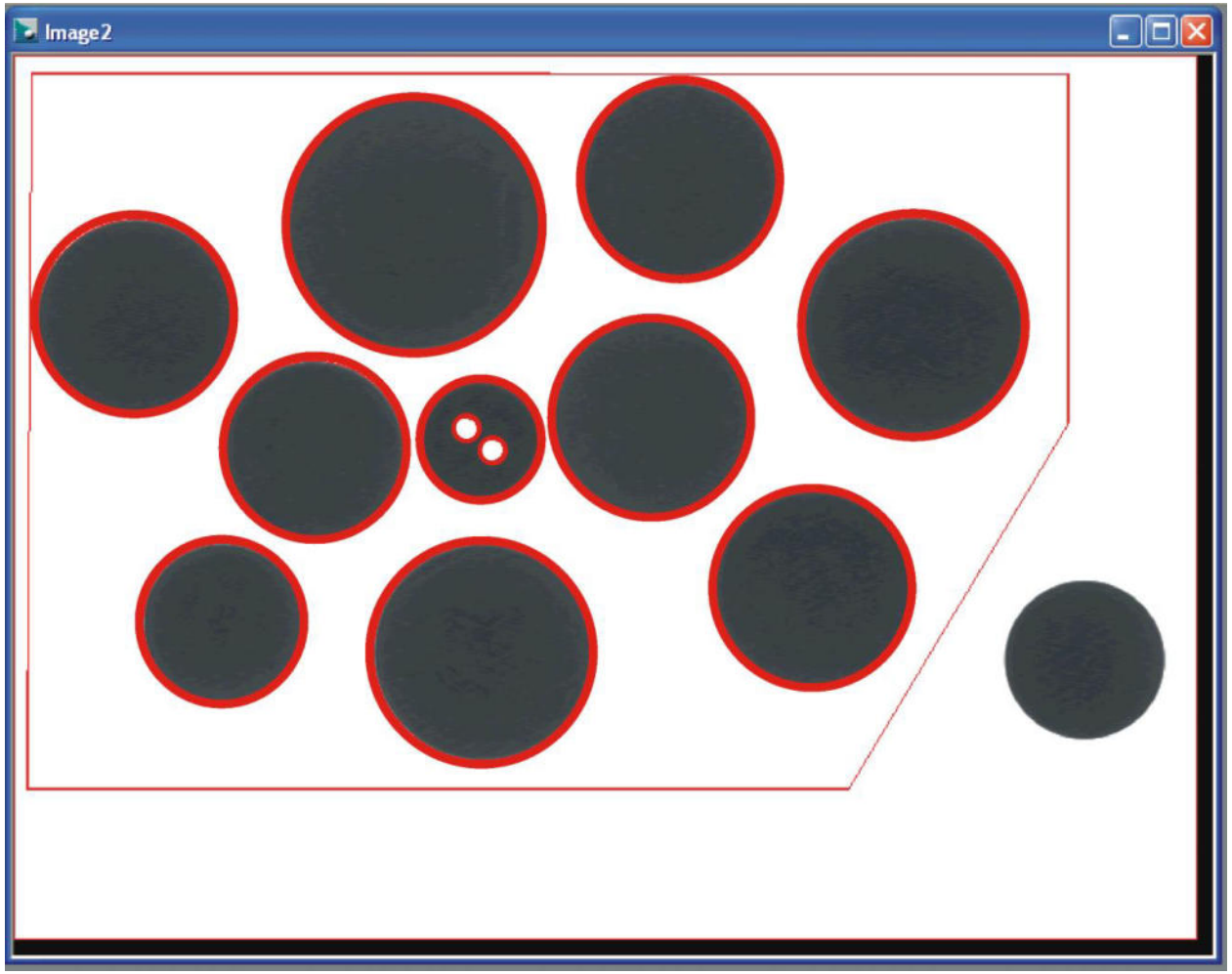
1. Step: Image Acquisition

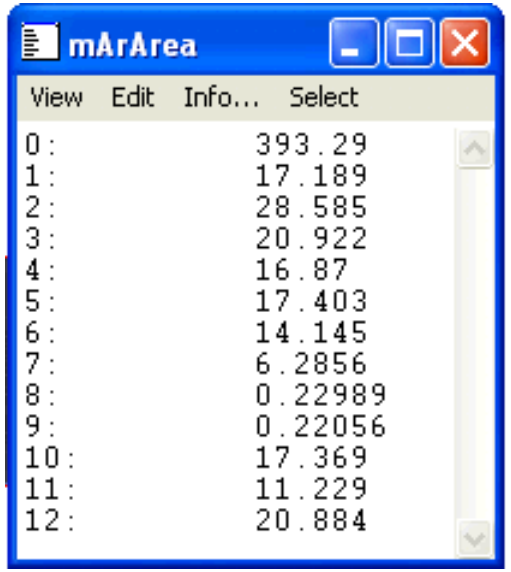


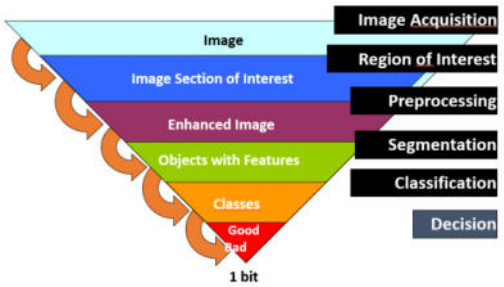
2. Step: Region of Interest



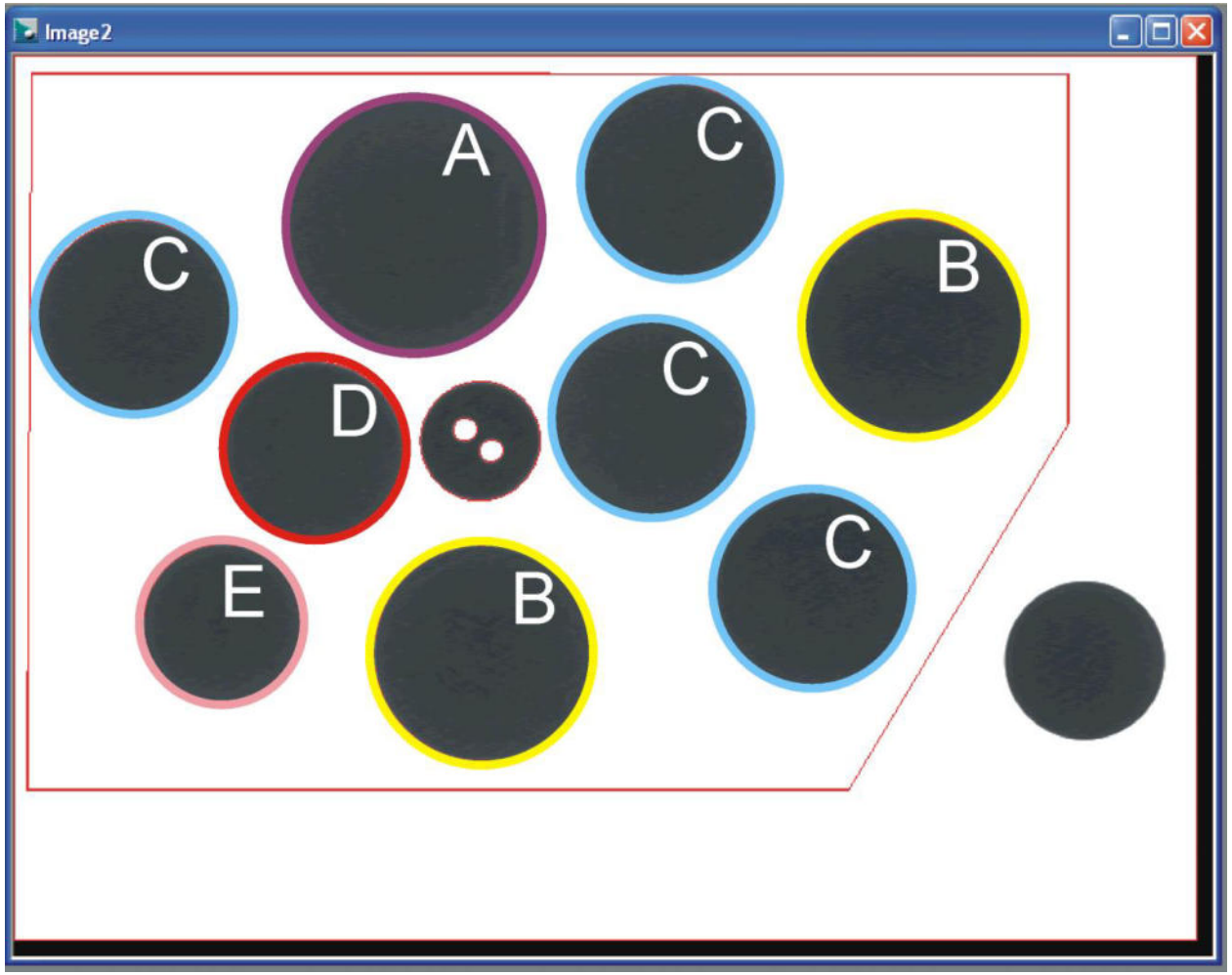
4. Step: Segmentation (e.g. BinaryLargeOBjects)

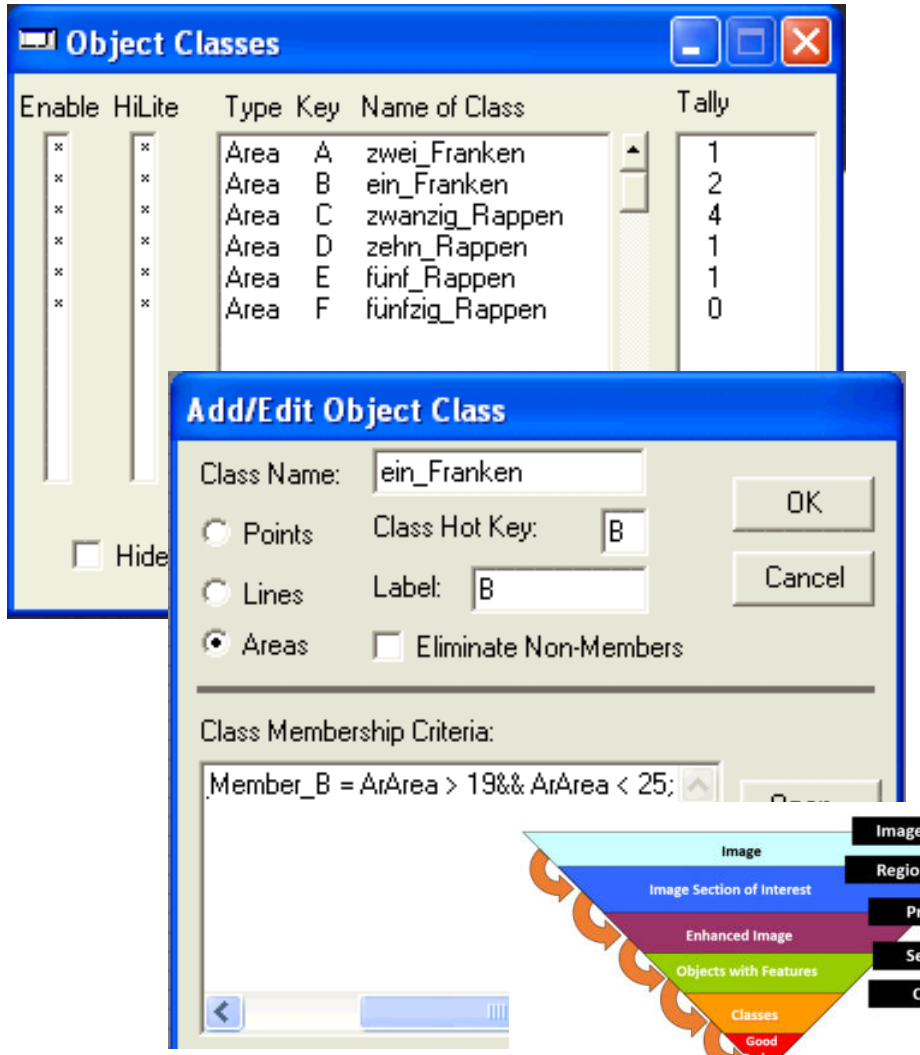






5. Step: Classification



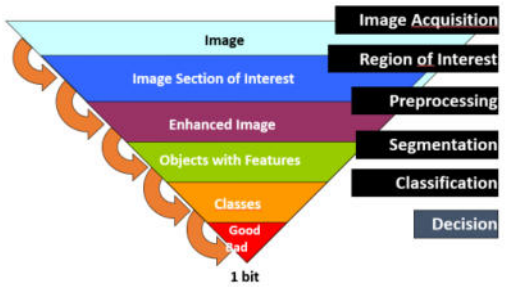


6. Step: Decision

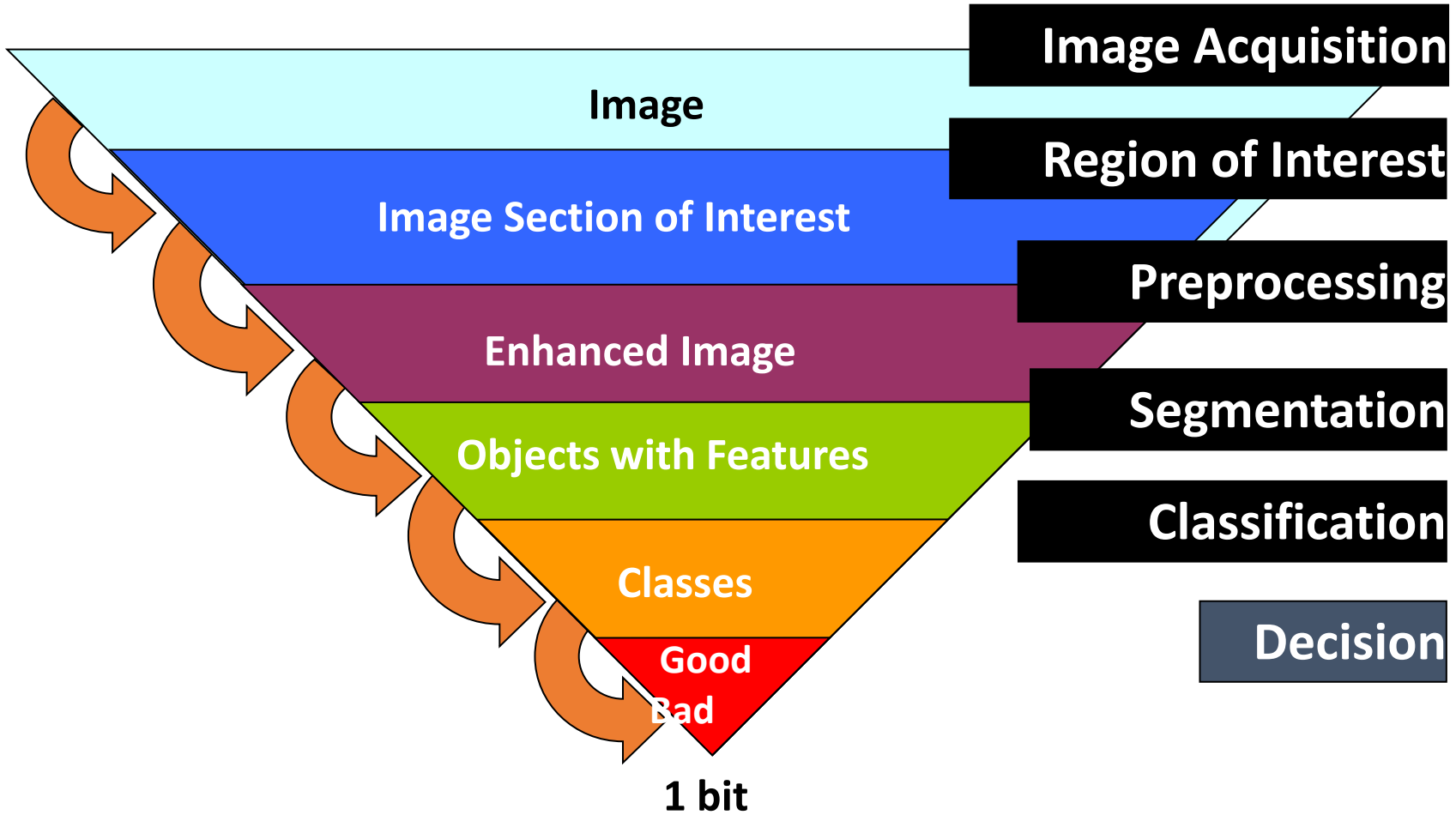
Bezahlung beurteilen

	Wert	Anzahl	Bezahlung	Preis
zwei Franken	SFr. 2.00	1	SFr. 2.00	
ein Franken	SFr. 1.00	2	SFr. 2.00	
fünfzig Rappen	SFr. 0.50	4	SFr. 2.00	
zwanzig Rappen	SFr. 0.20	0	SFr. -	
zehn Rappen	SFr. 0.10	1	SFr. 0.10	
fünf Rappen	SFr. 0.05	1	SFr. 0.05	
Summe			**SFr. 6.15**	**SFr. 6.15**

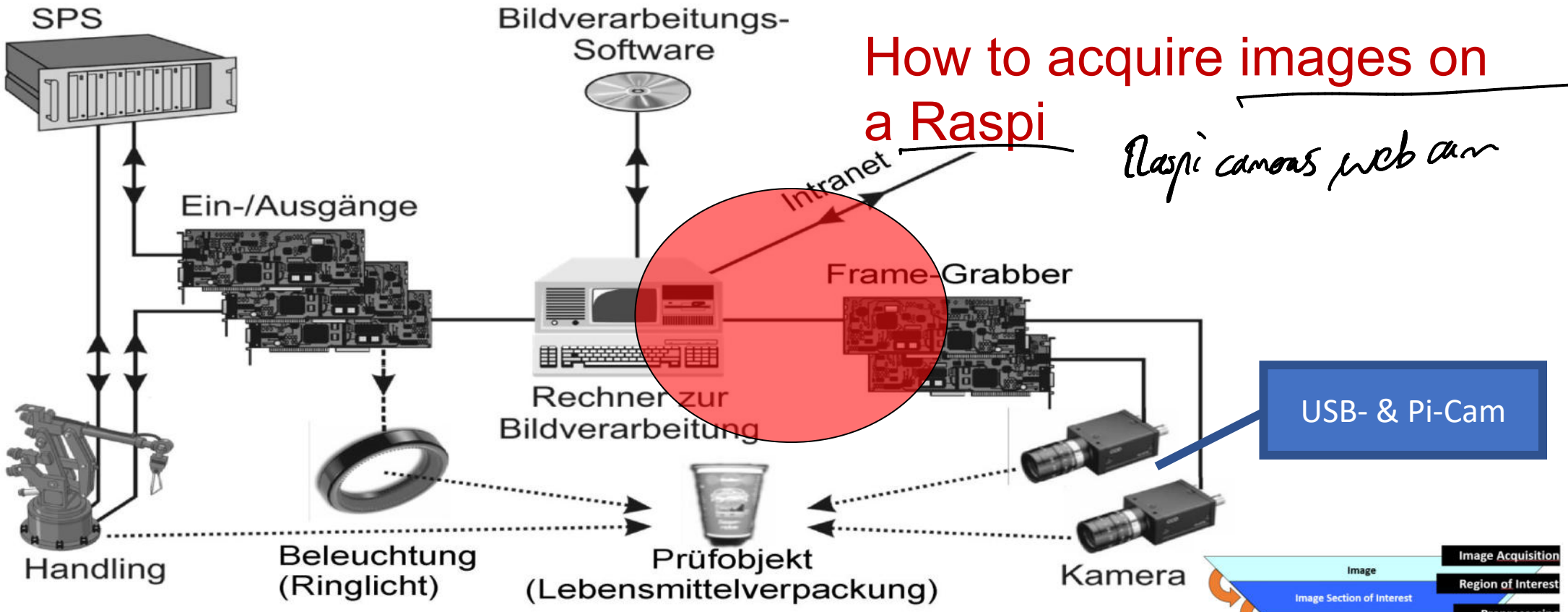
Payment accepted



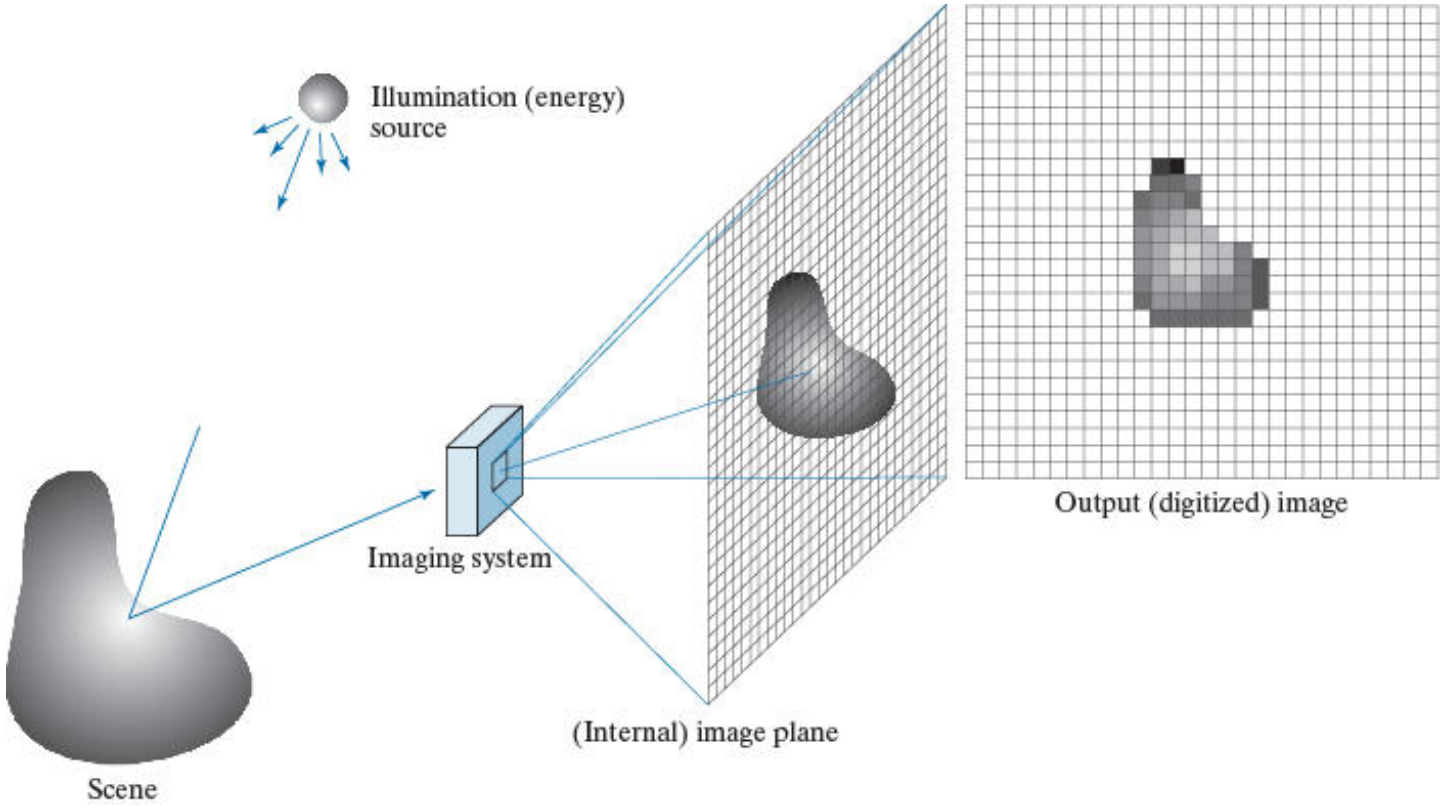
Typical Process Model in Image Processing



Our Focus: Image Acquisition and Formation



Images: Formation

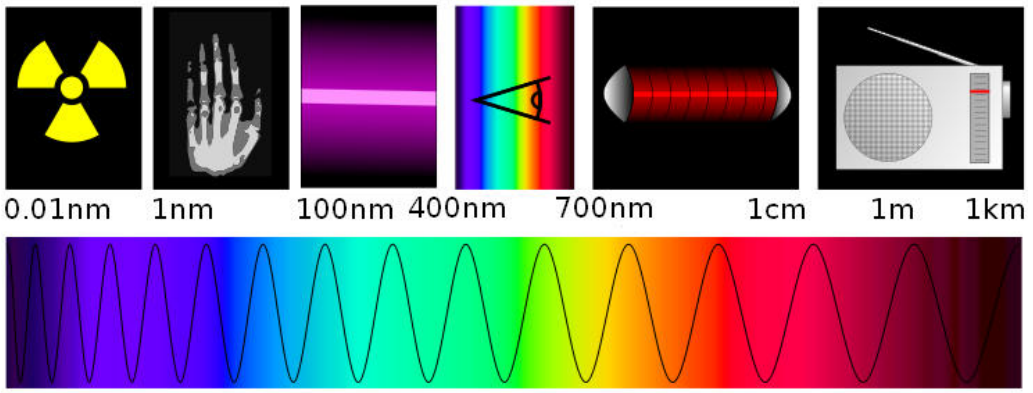


a
b c d e

FIGURE 2.15 An example of digital image acquisition. (a) Illumination (energy) source. (b) A scene. (c) Imaging system. (d) Projection of the scene onto the image plane. (e) Digitized image.

Image Formation: Light

- Light is an electro-magnetic wave
  - $c = \lambda \cdot f$
  - with c = 299‘792 km/s
  - spectrum (λ)



- Relevant ranges for λ in image processing:
  - **U**ltra**V**iolet (100 – 380nm; **UV-A (315 – 380nm)**)
  - **V**isible (380nm (violet), 520nm (green), 780nm (red))
  - **I**nfra**R**ed (780nm – 1mm; IR-A, IR-B)

Image Formation: Illumination

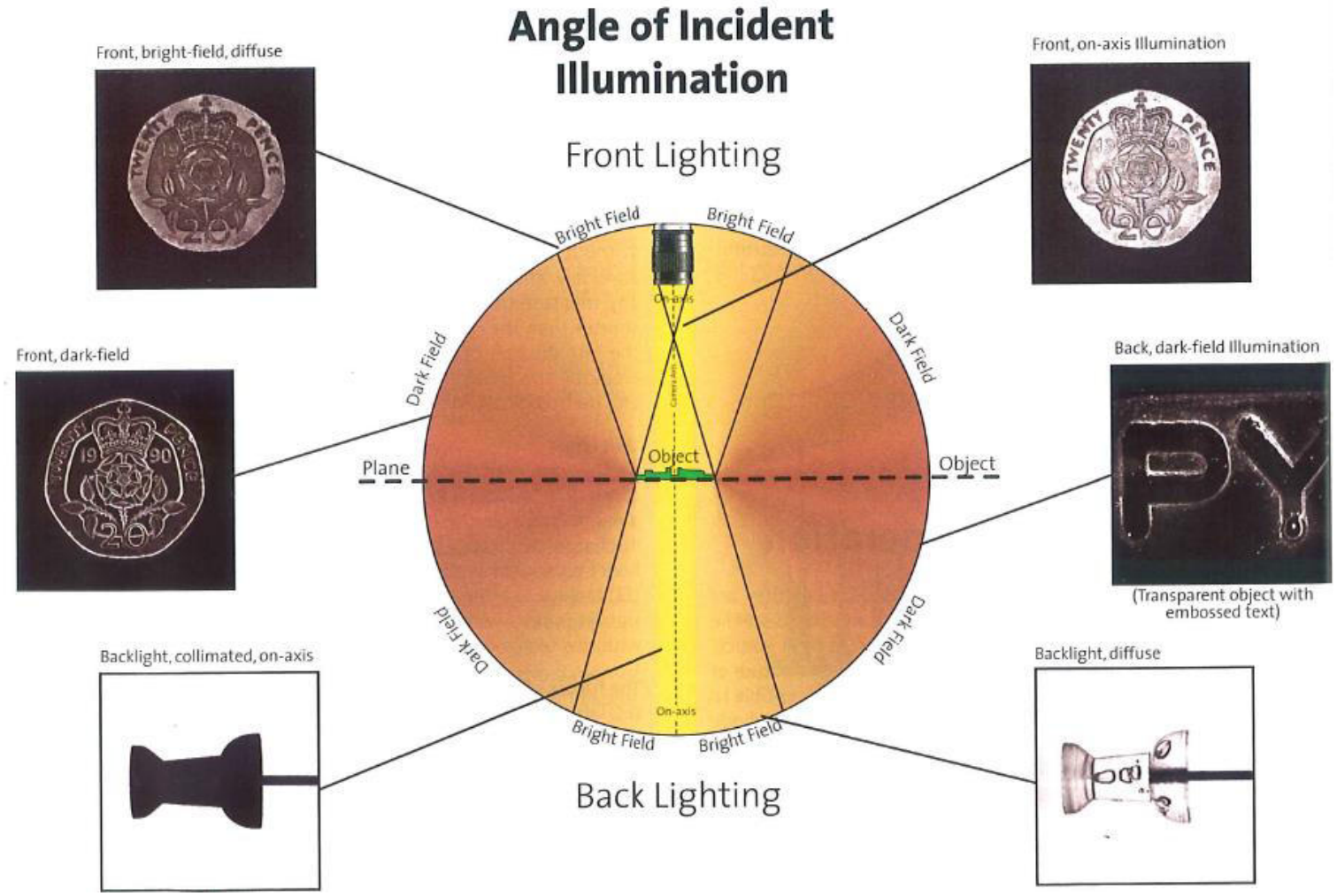


Image Formation: Optics and Cameras

Field of View (FOV): The viewable area of the object under inspection.

Working Distance (WD): The distance from the front of the lens to the object under inspection.

Resolution: The minimum feature size of the object that can be distinguished by the imaging system.

Depth of Field (DOF): The maximum object depth that can be maintained entirely in acceptable focus.

Sensor Size: The size of a camera sensor's active area

PMAG: The **P**rimary **Mag**nification of the lens is the ratio between the sensor size and the FOV.

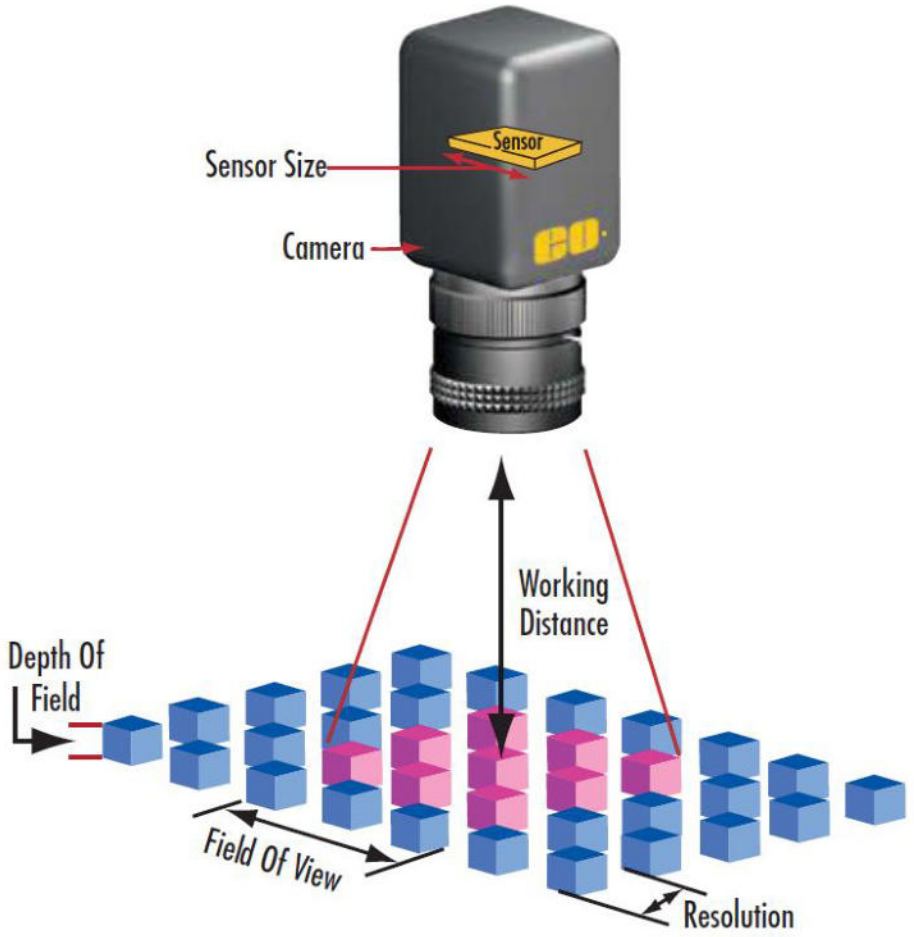


Image Formation: Sensors

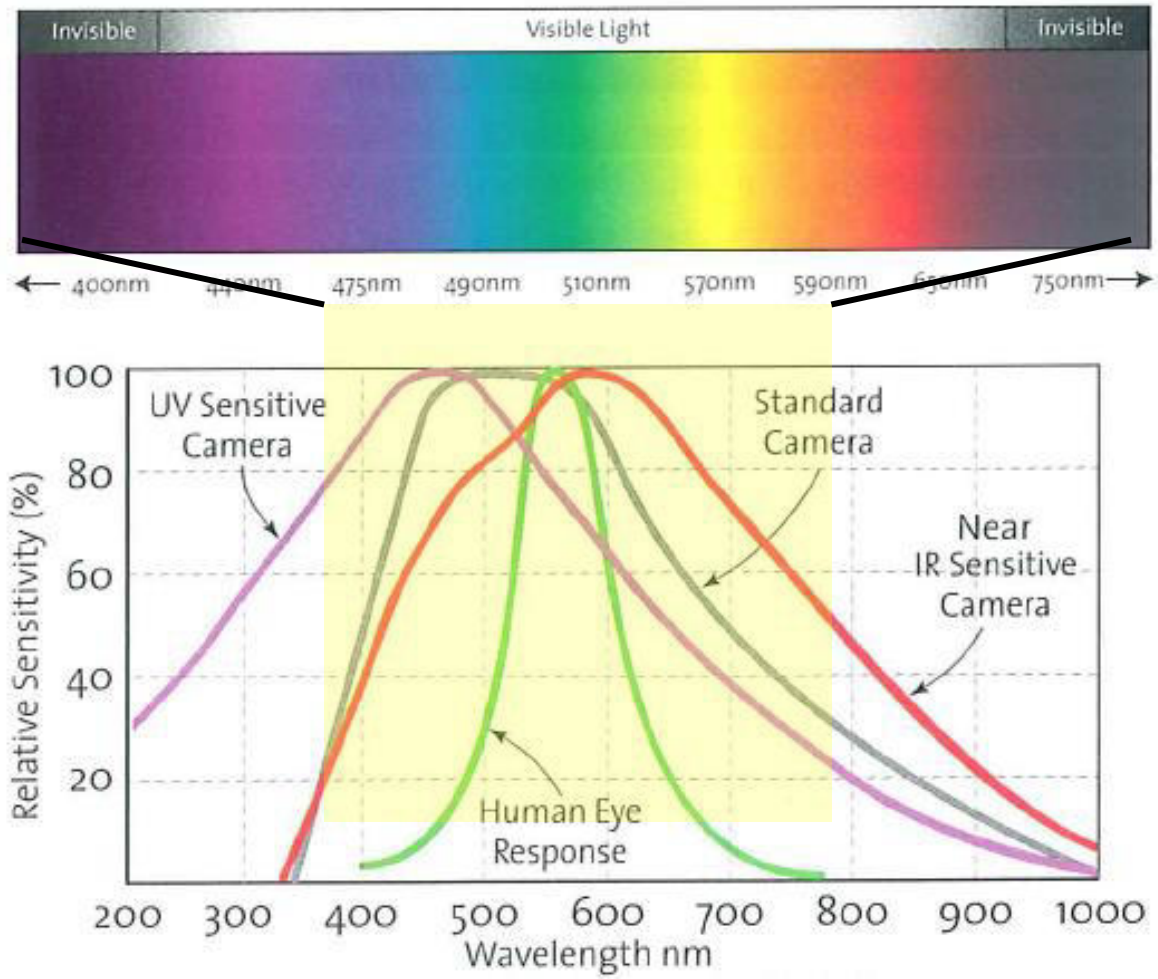
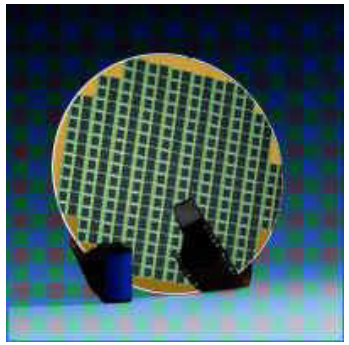
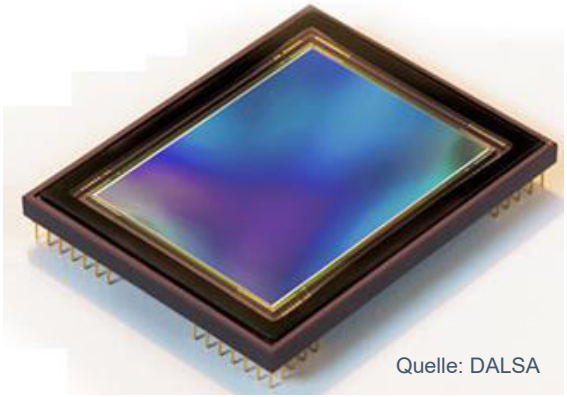
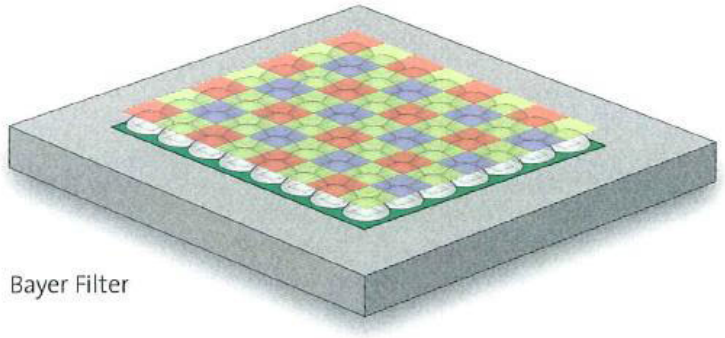


Image Formation: Sensors: Bayer Filter, 3 CCD (Prisma)







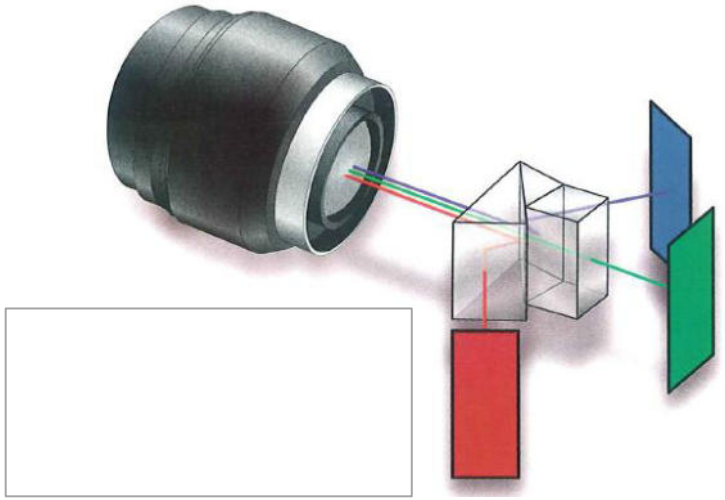
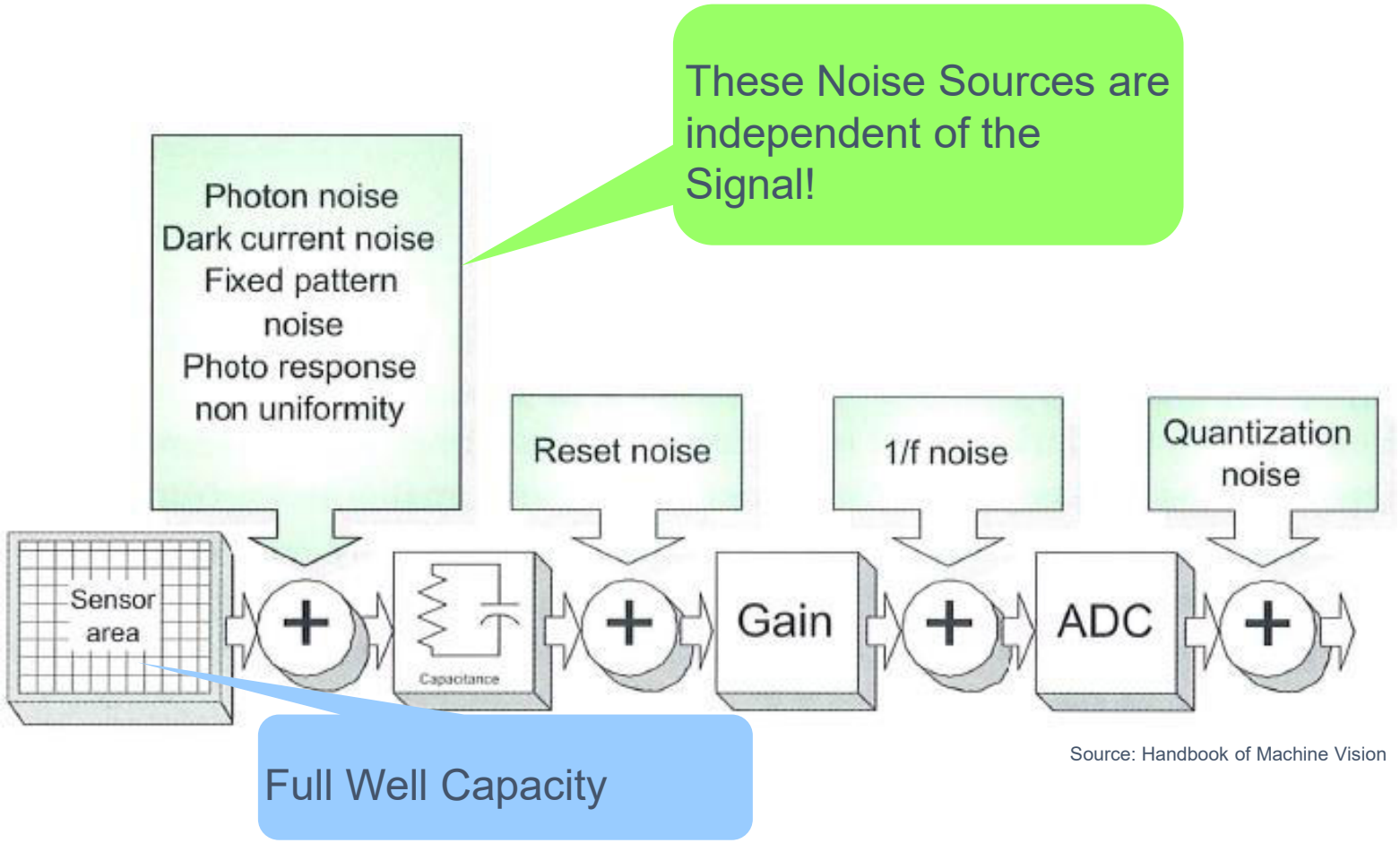


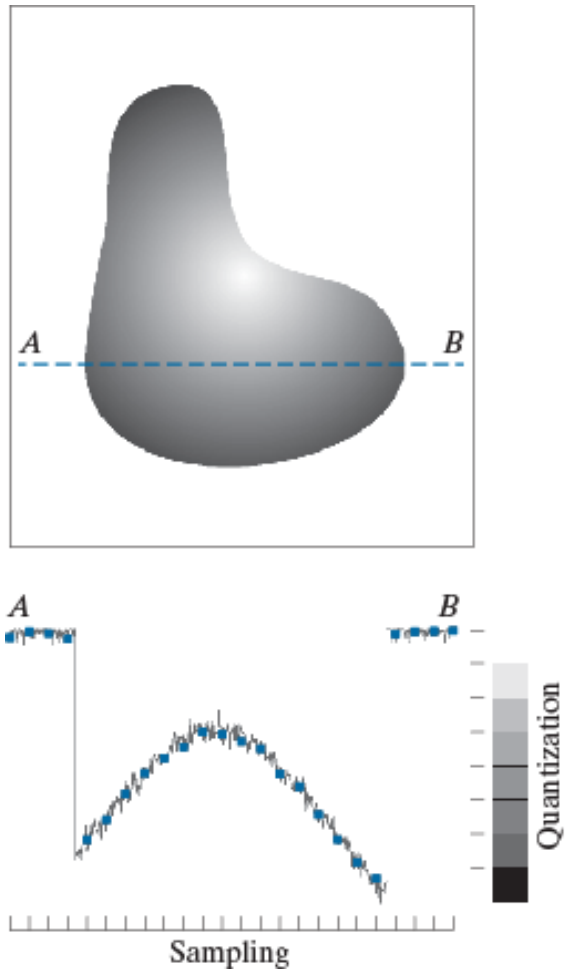
Image Formation: Signal Processing / Noise



Source: Handbook of Machine Vision

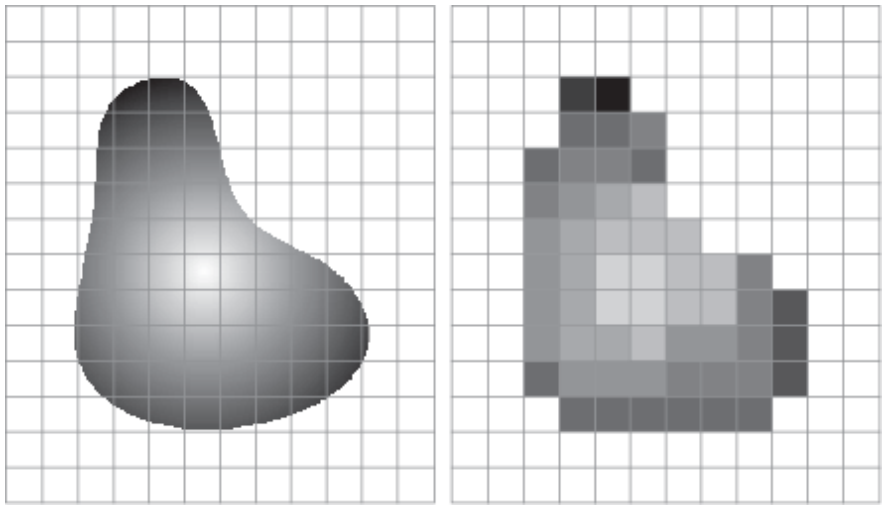
For more information see: https://www.emva.org/standards-technology/emva-1288/

Images: Sampling & Quantization



a b
c d

FIGURE 2.16
(a) Continuous image. (b) A scan line showing intensity variations along line AB in the continuous image. (c) Sampling and quantization. (d) Digital scan line. (The black border in (a) is included for clarity. It is not part of the image).



a b

FIGURE 2.17
(a) Continuous image projected onto a sensor array. (b) Result of image sampling and quantization.

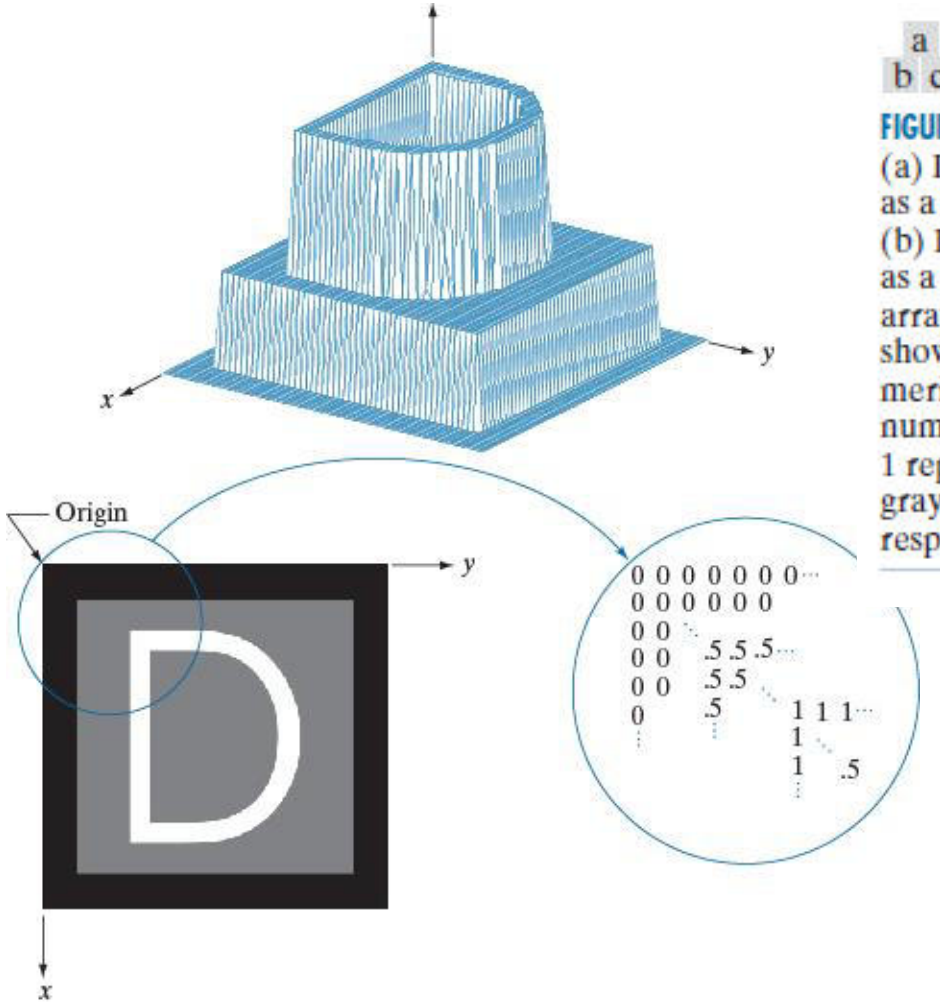
Images: 2D Image as a discrete function

Image Size / Sampling:

- M (rows) * N (columns) pixels (picture elements)
- Spatial resolution:
  - size in the real world (e.g. dpi, lpi, pixel/m, …)
- Coordinate system:
  - **origin in upper left corner,**
  - **x-axis downwards, y-axis to the right!**

Quantization:

- Number of intensity values
  - e.g. 8 bit → 2^8 = 256 different values
  - Int: {0 (black) .. 255 (white)} or Double: { 0.0 .. 1.0}



a
b c

FIGURE 2.18
(a) Image plotted as a surface. (b) Image displayed as a visual intensity array. (c) Image shown as a 2-D numerical array. (The numbers 0, .5, and 1 represent black, gray, and white, respectively.)

Images: Variety of Digital Images

- Quantization
  - bits/pixel
  - **I(u, v)**
- Channels
  - gray scale 1
  - color 3
  - multi spectral >3
  - **I(u, v, n)**
- Sequences
  - time
  - **I(u, v, n, t)**

Grayscale (Intensity Images):

Chan.	*Bits/Pix.*	*Range*	*Use*
1	1	0...1	Binary image: document, illustration, fax
1	8	0...255	Universal: photo, scan, print
1	12	0...4095	High quality: photo, scan, print
1	14	0...16383	Professional: photo, scan, print
1	16	0...65535	Highest quality: medicine, astronomy

Color Images:

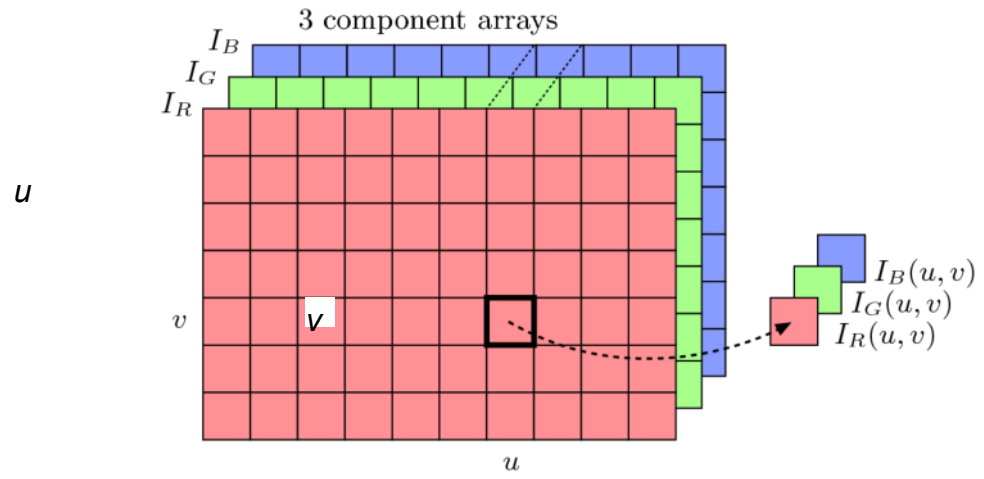
Chan.	*Bits/Pix.*	*Range*	*Use*
3	24	$[0...255]^3$	RGB, universal: photo, scan, print
3	36	$[0...4095]^3$	RGB, high quality: photo, scan, print
3	42	$[0...16383]^3$	RGB, professional: photo, scan, print
4	32	$[0...255]^4$	CMYK, digital prepress

Special Images:

Chan.	*Bits/Pix.*	*Range*	*Use*
1	16	$-32768...32767$	Whole numbers pos./neg., increased range
1	32	$\pm 3.4 \cdot 10^{38}$	Floating point: medicine, astronomy
1	64	$\pm 1.8 \cdot 10^{308}$	Floating point: internal processing

Images: Color Image Representation

- Full Color Images



- Indexed Images
- (Gray Level Images)

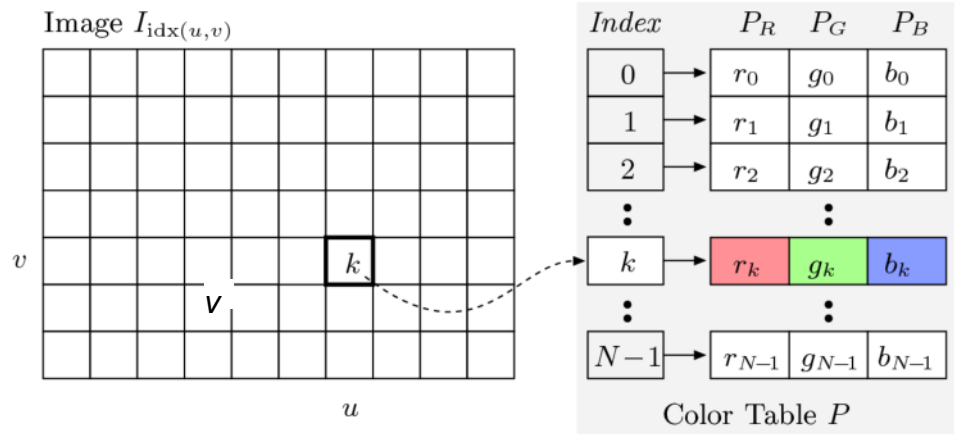
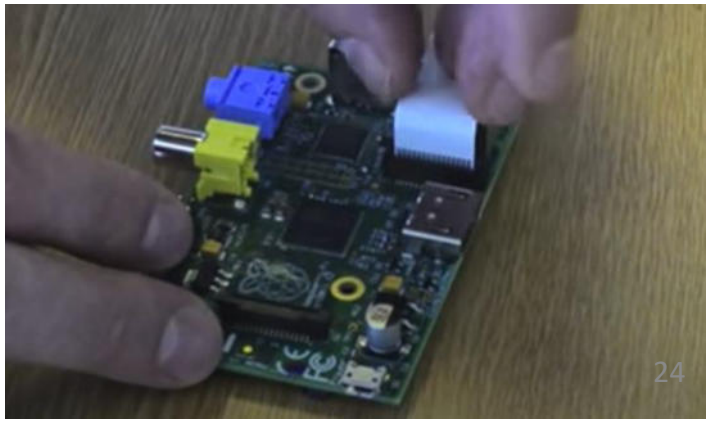


Image Acquisition on a Raspi - Hardware

- Cameras with a USB-Port (like a webcam)
  - Difficult to control, but good support

- Raspi Cameras
  - Full control (depending on model), software support evolving (libcamera)
  - See Raspberry Pi Documentation – Camera (https://www.raspberrypi.com/documentation/accessories/camera.html)
  -



Raspi Cameras – Software Integration / Drivers

- Open Source Stack:
  - libcamera (the only preinstalled version on OS Bullseye)
  - https://www.raspberrypi.com/documentation/accessories/camera.html#libcamera-and-libcamera-apps
  - Several apps
    - libcamera-hello: starts a camera preview stream and displays it on the screen.
    - libcamera-jpeg: runs a preview window and then capture high resolution still images.
    - libcamera-still: like raspistill.
    - libcamera-vid: a video capture application.
- Raspicam2 – Python Module
  - Builds on libcamera, allows
  - https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf
  - Has to be enabled explicitly (only in the Lite-Image)

Web Cameras – Software Integration / Drivers

- Independent of stack

- Tools
  - Video4Linux: (https://www.mankier.com/1/v4l2-ctl)
    - v4l2-ctl --list-devices
    - v4l2-ctl -d /dev/video0 --list-ctrls
    - v4l2-ctl --device /dev/video4 --set-ctrl=auto_exposure=1
    - Or interactively with qv4l2
  - fswebcam application

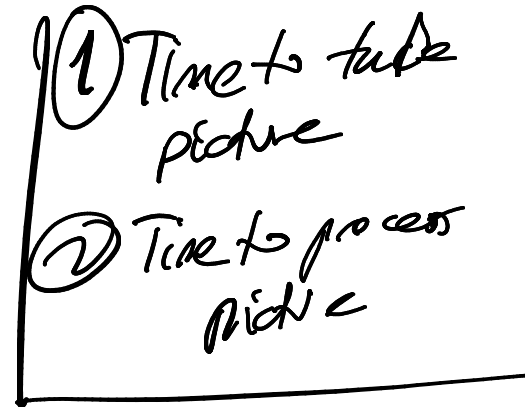
Image Acquisition with OpenCV and Python

- OpenCV (https://opencv.org/)
  - Open source (free for commercial use as well)
  - Highly optimized
  - Cross Plattform: Linux (including Raspi), MacOS, Windows, iOS, Android
  - Written in C++ with bindings for Java and Python
  - A lot of resources (e.g. tutorials)

- opencv-python (core) and opencv-contrib-python (with contrib/extra modules – see https://docs.opencv.org/4.x/)
  - (To install use "apt-get install python3-opencv")

- Python – see https://www.python.org/ https://docs.python.org/3.9/

Demo: My first image (v4l2 / q4l2)



1. USB-Cams
2. Raspicams

- Set: resolution
  - focus time
  - fps
  - zoom

v4l2 - Video4Linux

1. https://www.mankier.com/1/v4l2-ctl
2. All Devices: v4l2-ctl --list-devices
3. Available formats: v4l2-ctl --list-formats-ext
4. Properties: v4l2-ctl -d /dev/video0 --list-ctrls
5. Properties: v4l2-ctl -d /dev/video0 --set-ctrl=auto_exposure=1

— alternative: checks configuration
  - image format
  - adjust to better.

more better when recording videos

Demo: My first image (libcamera)

1. Raspi camera module

libcamera

1. https://libcamera.org/docs.html#documentation
2. Check if raspi-cam is working: libcamera-hello

Demo (with opencv and python)

- Getting started: is your system ready for opencv-python? (t0.py)

- Basics: Reading, Writing, Manipulating, Displaying Images (t1.py)
  - imread, imwrite, imshow
  - Image Data Structure (arrays from numpy)

Demo (with python)

- Image Acquisition
  - OpenCV Style: with VideoCapture class (see t2_grab.py)
  - Raspicam2 style: with Picamera2 class (see t2_grab_picamera2.py)

Demo (with python)

- Image Acquisition for a Thermal Image camera with its own library
  - https://www.waveshare.com/wiki/Thermal_Camera_HAT

Demo (cont.)

- Image Processing: a simple application (t3_grab_times)
  - Calculate Grab-Times

Demo (cont.)

- Image Processing: a simple application (t4_camera_settings)
  - Check out how to set different settings for cameras (if camera allows it)
    - Raspicam2
    - OpenCV

Demo (cont.)

- Image Processing: a simple application (t5_blob_detector)
  - Color transformations: color 2 grayscale (Attention: Raspicams use BGR)
  - SimpleBlobDetector

Demo (cont.)

- Image Processing: a simple application (t6_qr_code_detector)
  - Powerful Integrated OpenCV-Functionalities
  - OpenCV > v4.6 needed

Demo (cont.)

- Image Processing: a simple application (t7_object_detector)
  - Running a neural network on raspi

Demo (cont.)

- Application of AI on Jebot with Nvidia Jetson Nano
  - GPU used for processing neural network
- Jetson Nano vs Raspi Model 4B
  - Raspi Model 4B
    - 4GB RAM, 1.4GHz Quad-Core CPU, no separate GPU
  - Jetson Nano
    - 4GB RAM, 1.43GHz Quad-Core CPU, 128-core NVIDIA Maxwell GPU

Choose :

CamType
Cam Port – vs. USB

Exercise (choose yourself)

- Redo all the Demos

- Changing acquisition parameters (e.g. exposure time) of your camera
  - With properties in the VideoCapture class
  - With the Picamera2 class

- Grabbing from 2 cameras
  - If you have two cameras (e.g. a Raspi-Cam and a USB-Cam), grab images from these two cameras and display them in two windows (in a loop)

- Producer-Consumer Pattern with threading
  - Implement an application with 2 threads (based on VideoGet and VideoShow). How can you synchronize these threads?

Takeaway

- Image Acquisition becomes easier with Bookworm on a Raspi. Debugging is still difficult.

- Image Processing needs a lot of hardware resources.
  - A Raspi is not very powerful
  - Alternatives are available: e.g. NIVIDIA Jetson
  - Or stream the images to the cloud

What is the BeagleBone Blue?

From https://beagleboard.org/blue

BeagleBone® Blue is an all-in-one Linux-based computer for robotics, integrating onto a single small (3.5" x 2.15") board the Octavo OSD3358 microprocessor together with wifi/bluetooth, IMU/barometer, power regulation and state-of-charge LEDs for a 2-cell LiPo, H-Bridges, and discrete connectors for 4 DC motors+encoders, 8 servos, and all of the commonly-needed buses for additional peripherals in embedded applications. Fully open source and actively supported by a strong community, the real-time performance, flexible networking, and rich set of robotics-oriented capabilities make building mobile robots with the Blue fast, streamlined, affordable, and fun.

1

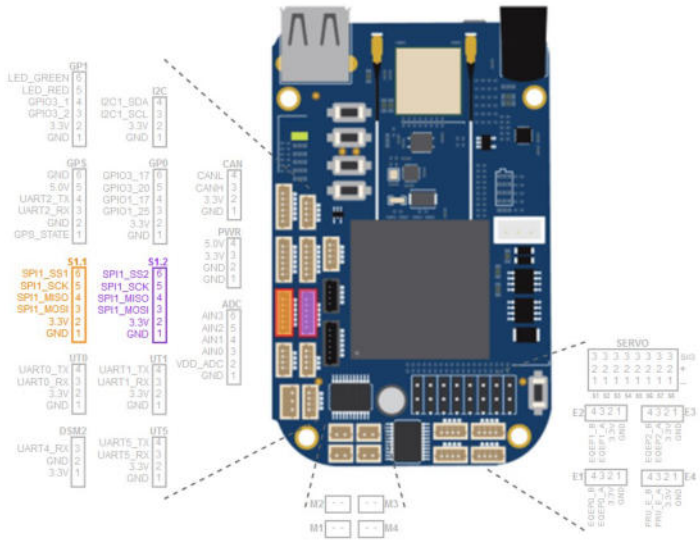
Robot Control Library

- C library and example programs at https://beagleboard.org/librobotcontrol/index.html
- For all beagle boards
- Aimed at developing robotics software on embedded computers.
- RCPY python interface available at https://github.com/mcdeoliveira/rcpy.
- e.g. interface for reading the barometer

```
int  rc_bmp_init (rc_bmp_oversample_t oversample, rc_bmp_filter_t filter)
     powers on the barometer and initializes it with the given oversample and filter settings. More...

int  rc_bmp_set_sea_level_pressure_pa (double pa)
     If you know the current sea level pressure for your region and weather, you can use this to correct the altitude reading.

int  rc_bmp_power_off (void)
     Puts the barometer into a low power state, should be called at the end of your program before close. More...

int  rc_bmp_read (rc_bmp_data_t *data)
     Reads the newest temperature and pressure measurments from the barometer over the I2C bus. More...
```

2

Pinout



directly connect motors and sensors

3

Power & Booting from the SD Card

The USB connector delivers power and to your Beaglebone board. However, your host might not supply enough power or the connecting cable might have shaky contacts. For this, your board has a barrel jack. The voltage should be 12V.
You must use this 12V supply in any case if you want to drive motors or any other actuators.

1. Apply power through USB or through barrel jack
2. Board will boot from internal eMMC flash
3. Insert sd card with our custom image
4. Press button SD (the one in the corner) and hold it pressed
5. Press and release button Reset
6. Release button SD, board will boot from sd card
7. After rebooting, will start from sd card again, setting is preserved

4

Crossdevelopment

- Develop on a high power system such as your laptop / desktop
- Use compiler and libraries for your target, here ARM architecture

- All necessary tools for cross compilation must be present on your host system (so called toolchain). This can be cumbersome to setup correctly
- Target system can be minimal, e.g. no editors, compilers, may be realtime capable
- We use Yocto build system, creates
  - Target image
  - SDK for use on the host

5

Crosscompile

```
$ gcc hello.c –o hello
$ file hello
hello: ELF 64-bit LSB shared object, x86-64, version 1
   (SYSV), dynamically linked, interpreter /lib64/ld-linux-
   x86-64.so.2,
   BuildID[sha1]=689c1c4683375c433a37bd7938ec6c659a8079b4,
   for GNU/Linux 3.2.0, not stripped
```

$ /opt/bbb/bin/arm-buildroot-linux-gnueabi-gcc hello.c -o hello

```
$ file hello
```

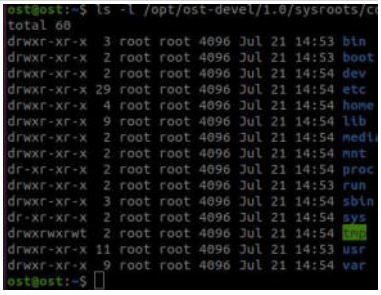
hello: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.3, for GNU/Linux 2.6.16, not stripped

$ ./hello

bash: ./hello: cannot execute binary file: Exec format error

6

Using a SDK



- Built together with the target image, can be downloaded separately
- Target system is present on the host in special location
- Tools for the target are present in special location
- SDK includes a script that needs to be sourced in order to use the SDK. This, among other things, sets up the standard environment variables (such as CC, LD and CFLAGS).
- Run the script with (must be a space in between!)

```
$ . /home/graf/sdk/ost-devel/1.0/bbblue/
        environment-setup-cortexa8hf-
        neon-poky-linux-gnueabi
```

7

Read-Only File System

Images come in two flavors, *devel* and *prod*.
prod is write protected which makes it safe for powering it off without proper shutdown procedure. However, when deploying to such an image, you will get

```
$ scp: /home/ost/bin/hello: Read-only file system
```

Before downloading you have to make the file system writable. Connect to your target with

```
$ ssh ost@192.168.7.2
```

then remount the file system with

```
ost@bblue:$ sudo mount / -o remount,rw
```

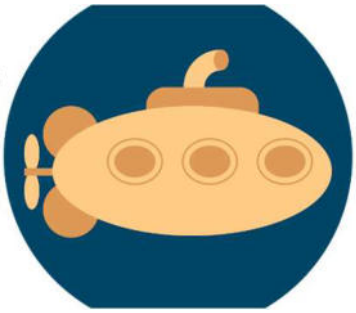
A *devel* system should be stopped gracefully at the end with

```
ost@bblue:$ sudo /sbin/halt
```

8

U-Boot

Everything has a beginning. You cannot have a running device with Embedded Linux without a bootloader that initializes the hardware and load and start the OS. U-Boot (subtitled “the Universal Boot Loader” and often shortened to **U-Boot)**, started by Wolfgang Denx more than 20 years ago, has become a de-facto standard for Embedded Linux Device and not only. It is available for a large number of different computer architectures, including 68k, ARM, Blackfin, MicroBlaze, MIPS, Nios, SuperH, PPC, RISC-V and x86.



from: https://www.denx.de/project/u-boot/

Boot process for Embedded Linux

real mode: no memory Process

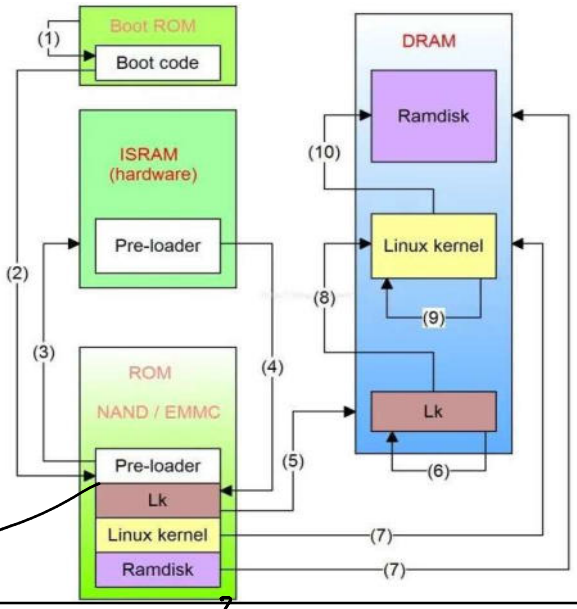
1

Steps

- Processor runs in “real mode”: The MMU is off. There is no memory translation nor protection and no cache. U-Boot plays a few dirty tricks based on this.
- Pre-relocation initialization (possibly directly from flash or other kind of ROM)
- Relocation: Copy the code to RAM.
- Post-relocation initialization (from proper RAM).
- Execution of commands: Through autoboot or console shell
- Passing control to the Linux kernel (or other target application)

2

Memory involved

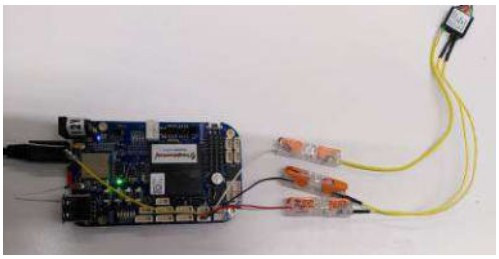


Lk = loading kernel

https://www.linkedin.com/pulse/how-u-boot-loads-linux-kernel-praveen-singh

3

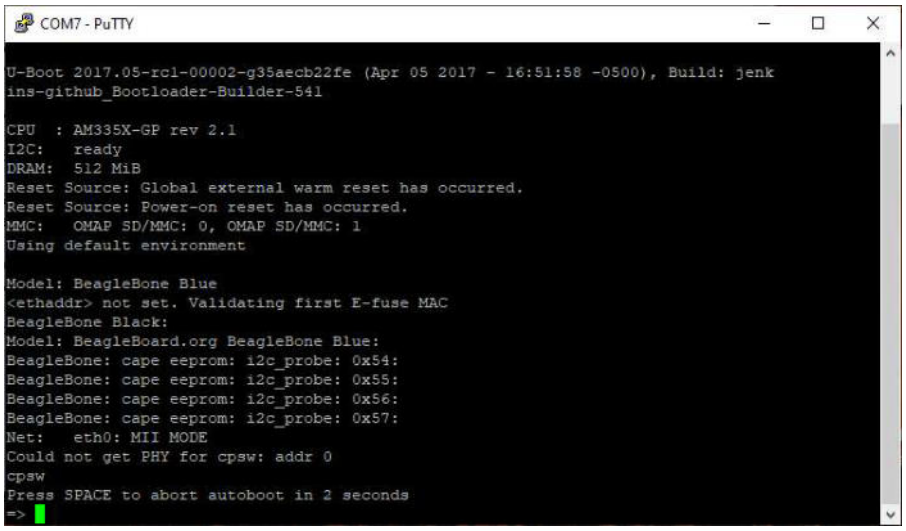
How to read from the console?



TX and RX are labelled from the perspective of the UART Cable! Connect the cable/adapter's RX to the pin labelled TX (Pin 4) and the adapter's TX to RX (Pin 3). Please also note that you need to power the BeagleBone Blue device by using the generic 5V USB port or the 12V power port!
You can then connect to the BeagleBone Blue device via serial console with a baudrate of 115200.

4

Booting on Beaglebone



5

U-Boot Commands

```
#   help
?       - alias for 'help'
autoscr - run script from memory
base    - print or set address offset
bdinfo  - print Board Info structure
boot    - boot default, i.e., run 'bootcmd'
bootd   - boot default, i.e., run 'bootcmd'
bootelf - Boot from an ELF image in memory
bootm   - boot application image from memory
bootp   - boot image via network using BootP/TFTP protocol
bootvx  - Boot vxWorks from an ELF image
clock   - Set Processor Clock
cmp     - memory compare
coninfo - print console devices and information
cp      - memory copy
crc32   - checksum calculation
date    - get/set/reset date & time
dboot   - Digi ConnectCore modules boot commands
dcache  - enable or disable data cache
dhcp    - invoke DHCP client to obtain IP/boot params
```

6

Environment

```
# printenv
bootdelay=1
baudrate=115200
ipaddr=192.168.0.2
serverip=192.168.0.1
netmask=255.255.255.0
bootfile="uImage"
filesize=B81A24
bootcmd=nand read 80200000 280000 400000;bootm 80200000
bootargs=console=ttyS2,115200n8 root=/dev/mtdblock4
         rw rootfstyp=jffs2 nohz=off
…
…

# setenv mytestvariable abcdefg
# printenv mytestvariable
mytestenv=abcdefg

# saveenv
```

7

Starting Linux and Applications

And then?

Initialization manager – systemd is a suite of components that is used to initialize and configure a Linux system. There are utility applications used to monitor and control the system and there are init scripts for the different services within the system.

```
# /etc/systemd/system/delta-startup.service
[Unit]
Description=delta robot application

[Service]
Type=simple
ExecStart=/bin/bash -c 'sudo /usr/bin/screen -dmS delta /home/ost/bin/start.sh'
RemainAfterExit=yes

[Install]
WantedBy=multi-user.target
```

8

Kernel Configuration

The kernel configuration is kept in a file called .config in the top directory of the kernel source tree. If you have just expanded the kernel source code, there will be no .config file, so it needs to be created from scratch by basing it on the "default configuration," taken from a running kernel version, or taken from a distribution kernel release.

$ make defconfig

9

Menuconfig

The kernel contains almost two thousand different configuration options, so being asked for every individual one will take a very long time. Luckily, there is an easier way to configure a kernel: base the configuration on a pre-built configuration. Every kernel version comes with a "default" kernel configuration. This configuration is loosely based on the defaults that the kernel maintainer of that architecture feels are the best options to be used. In some cases, it is merely the configuration that is used by the kernel maintainer himself for his personal machines. This is true for the i386 architecture, where the default kernel configuration matches closely what Linus Torvalds uses for his main development machine.

- $ make menuconfig
- $ make

10

Device Tree

Is a data structure describing the hardware components of a particular computer so that the operating system's kernel can use and manage those components, including the CPU or CPUs, the memory, the buses and the integrated peripherals. Device trees are used on various platforms, notably ARM and PPC.

x86 uses auto configuration protocols, such as ACPI to discover hardware.

Is a tree of named nodes with properties. Nodes can contain child nodes.

Device tree files are text files for easy management but must be compiled into a device tree blob.

11

Example

```
/ {
        soc {
                i2c@40003000 {
                        compatible = "nordic,nrf-twim";
                        label = "I2C_0";
                        reg = <0x40003000 0x1000>;
                    ~   apds9960@39 {
                                compatible = "avago,apds9960";
                                label = "APDS9960";
                                reg = <0x39>;
                        };
                    ~   ti_hdc@43 {
                                compatible = "ti,hdc", "ti,hdc1010";
                                label = "HDC1010";
                                reg = <0x43>;
                        };
                    ~   mma8652fc@1d {
                                compatible = "nxp,fxos8700", "nxp,mma8652fc";
                                label = "MMA8652FC";
                                reg = <0x1d>;
                        };
                };
        };
};
```

12

Device Tree for Beaglebone Blue

```
#include "am33xx.dtsi"
#include "am335x-bone-common-universal-pins.dtsi"
#include <dt-bindings/interrupt-controller/irq.h>

/ {
        model = "TI AM335x BeagleBone Blue";
        compatible = "ti,am335x-bone-blue", "ti,am33xx";

        chosen {
                stdout-path = &uart0;
                base_dtb = "am335x-boneblue.dts";
        };

        cpus {
                cpu@0 {
                        cpu0-supply = <&dcdc2_reg>;
                };
        };

        memory@80000000 {
                device_type = "memory";
                reg = <0x80000000 0x20000000>; /* 512 MB */
        };
```

13

What about the UART1 on Beaglebone Blue?

https://github.com/RobertCNelson/linux-stable-rcn-ee/blob/4.19.94-ti-rt-r59/arch/arm/boot/dts/am335x-boneblue.dts

```
&uart0 {
        pinctrl-names = "default";
        pinctrl-0 = <&uart0_pins>;
        status = "okay";
};

&uart1 {
        status = "okay";
};

&uart2 {
        status = "okay";
};
```

sudo microcom /dev/ttyS1

14

GDB QUICK REFERENCE GDB Version 5

Essential Commands

`gdb` *program* [*core*]	debug *program* [using coredump *core*]
`b` [*file*`:`]*function*	set breakpoint at *function* [in *file*]
`run` [*arglist*]	start your program [with *arglist*]
`bt`	backtrace: display program stack
`p` *expr*	display the value of an expression
`c`	continue running your program
`n`	next line, stepping over function calls
`s`	next line, stepping into function calls

Starting GDB

`gdb`	start GDB, with no debugging files
`gdb` *program*	begin debugging *program*
`gdb` *program core*	debug coredump *core* produced by *program*
`gdb --help`	describe command line options

Stopping GDB

`quit`	exit GDB; also `q` or `EOF` (eg `C-d`)
`INTERRUPT`	(eg `C-c`) terminate current command, or send to running process

Getting Help

`help`	list classes of commands
`help` *class*	one-line descriptions for commands in *class*
`help` *command*	describe *command*

Executing your Program

`run` *arglist*	start your program with *arglist*
`run`	start your program with current argument list
`run ... <`*inf* `>`*outf*	start your program with input, output redirected
`kill`	kill running program
`tty` *dev*	use *dev* as stdin and stdout for next `run`
`set args` *arglist*	specify *arglist* for next `run`
`set args`	specify empty argument list
`show args`	display argument list
`show env`	show all environment variables
`show env` *var*	show value of environment variable *var*
`set env` *var string*	set environment variable *var*
`unset env` *var*	remove *var* from environment

Shell Commands

`cd` *dir*	change working directory to *dir*
`pwd`	Print working directory
`make ...`	call "`make`"
`shell` *cmd*	execute arbitrary shell command string

[ ] surround optional arguments ... show one or more arguments

Breakpoints and Watchpoints

`break` [*file*`:`]*line* `b` [*file*`:`]*line*	set breakpoint at *line* number [in *file*] eg: `break main.c:37`
`break` [*file*`:`]*func*	set breakpoint at *func* [in *file*]
`break +`*offset*	set break at *offset* lines from current stop
`break -`*offset*	
`break *`*addr*	set breakpoint at address *addr*
`break`	set breakpoint at next instruction
`break ... if` *expr*	break conditionally on nonzero *expr*
`cond` *n* [*expr*]	new conditional expression on breakpoint *n*; make unconditional if no *expr*
`tbreak ...`	temporary break; disable when reached
`rbreak` *regex*	break on all functions matching *regex*
`watch` *expr*	set a watchpoint for expression *expr*
`catch` *event*	break at *event*, which may be `catch`, `throw`, `exec`, `fork`, `vfork`, `load`, or `unload`.
`info break`	show defined breakpoints
`info watch`	show defined watchpoints
`clear`	delete breakpoints at next instruction
`clear` [*file*`:`]*fun*	delete breakpoints at entry to *fun*()
`clear` [*file*`:`]*line*	delete breakpoints on source line
`delete` [*n*]	delete breakpoints [or breakpoint *n*]
`disable` [*n*]	disable breakpoints [or breakpoint *n*]
`enable` [*n*]	enable breakpoints [or breakpoint *n*]
`enable once` [*n*]	enable breakpoints [or breakpoint *n*]; disable again when reached
`enable del` [*n*]	enable breakpoints [or breakpoint *n*]; delete when reached
`ignore` *n count*	ignore breakpoint *n*, *count* times
`commands` *n* [`silent`] *command-list*	execute GDB *command-list* every time breakpoint *n* is reached. [`silent` suppresses default display]
`end`	end of *command-list*

Program Stack

`backtrace` [*n*] `bt` [*n*]	print trace of all frames in stack; or of *n* frames—innermost if *n*>0, outermost if *n*<0
`frame` [*n*]	select frame number *n* or frame at address *n*; if no *n*, display current frame
`up` *n*	select frame *n* frames up
`down` *n*	select frame *n* frames down
`info frame` [*addr*]	describe selected frame, or frame at *addr*
`info args`	arguments of selected frame
`info locals`	local variables of selected frame
`info reg` [*rn*]...	register values [for regs *rn*] in selected frame; `all-reg` includes floating point
`info all-reg` [*rn*]	

Execution Control

`continue` [*count*] `c` [*count*]	continue running; if *count* specified, ignore this breakpoint next *count* times
`step` [*count*] `s` [*count*]	execute until another line reached; repeat *count* times if specified
`stepi` [*count*] `si` [*count*]	step by machine instructions rather than source lines
`next` [*count*] `n` [*count*]	execute next line, including any function calls
`nexti` [*count*] `ni` [*count*]	next machine instruction rather than source line
`until` [*location*]	run until next instruction (or *location*)
`finish`	run until selected stack frame returns
`return` [*expr*]	pop selected stack frame without executing [setting return value]
`signal` *num*	resume execution with signal *s* (none if 0)
`jump` *line*	resume execution at specified *line* number or *address*
`jump *`*address*	
`set var=`*expr*	evaluate *expr* without displaying it; use for altering program variables

Display

`print` [`/`*f*] [*expr*] `p` [`/`*f*] [*expr*]	show value of *expr* [or last value `$`] according to format *f*:
`x`	hexadecimal
`d`	signed decimal
`u`	unsigned decimal
`o`	octal
`t`	binary
`a`	address, absolute and relative
`c`	character
`f`	floating point
`call` [`/`*f*] *expr*	like `print` but does not display `void`
`x` [`/`*Nuf*] *expr*	examine memory at address *expr*; optional format spec follows slash
N	count of how many units to display
u	unit size; one of `b` individual bytes `h` halfwords (two bytes) `w` words (four bytes) `g` giant words (eight bytes)
f	printing format. Any `print` format, or `s` null-terminated string `i` machine instructions
`disassem` [*addr*]	display memory as machine instructions

Automatic Display

`display` [`/`*f*] *expr*	show value of *expr* each time program stops [according to format *f*]
`display`	display all enabled expressions on list
`undisplay` *n*	remove number(s) *n* from list of automatically displayed expressions
`disable disp` *n*	disable display for expression(s) number *n*
`enable disp` *n*	enable display for expression(s) number *n*
`info display`	numbered list of display expressions

Expressions

expr	an expression in C, C++, or Modula-2 (including function calls), or:
addr`@`*len*	an array of *len* elements beginning at *addr*
file`::`*nm*	a variable or function *nm* defined in *file*
{*type*}*addr*	read memory at *addr* as specified *type*
`$`	most recent displayed value
`$`*n*	*n*th displayed value
`$$`	displayed value previous to $
`$$`*n*	*n*th displayed value back from $
`$_`	last address examined with `x`
`$__`	value at address $_
`$`*var*	convenience variable; assign any value
`show values` [*n*]	show last 10 values [or surrounding $*n*]
`show conv`	display all convenience variables

Symbol Table

`info address` *s*	show where symbol *s* is stored
`info func` [*regex*]	show names, types of defined functions (all, or matching *regex*)
`info var` [*regex*]	show names, types of global variables (all, or matching *regex*)
`whatis` [*expr*]	show data type of *expr* [or `$`] without evaluating; `ptype` gives more detail
`ptype` [*expr*]	
`ptype` *type*	describe type, struct, union, or enum

GDB Scripts

`source` *script*	read, execute GDB commands from file *script*
`define` *cmd* *command-list*	create new GDB command *cmd*; execute script defined by *command-list*
`end`	end of *command-list*
`document` *cmd* *help-text*	create online documentation for new GDB command *cmd*
`end`	end of *help-text*

Signals

`handle` *signal act*	specify GDB actions for *signal*:
`print`	announce signal
`noprint`	be silent for signal
`stop`	halt execution on signal
`nostop`	do not halt execution
`pass`	allow your program to handle signal
`nopass`	do not allow your program to see signal
`info signals`	show table of signals, GDB action for each

Debugging Targets

`target` *type param*	connect to target machine, process, or file
`help target`	display available targets
`attach` *param*	connect to another process
`detach`	release target from GDB control

Controlling GDB

`set` *param value*	set one of GDB's internal parameters
`show` *param*	display current setting of parameter

Parameters understood by `set` and `show`:

`complaint` *limit*	number of messages on unusual symbols
`confirm` *on/off*	enable or disable cautionary queries
`editing` *on/off*	control `readline` command-line editing
`height` *lpp*	number of lines before pause in display
`language` *lang*	Language for GDB expressions (`auto`, `c` or `modula-2`)
`listsize` *n*	number of lines shown by `list`
`prompt` *str*	use *str* as GDB prompt
`radix` *base*	octal, decimal, or hex number representation
`verbose` *on/off*	control messages when loading symbols
`width` *cpl*	number of characters before line folded
`write` *on/off*	Allow or forbid patching binary, core files (when reopened with `exec` or `core`)
`history` ...	groups with the following options:
`h` ...	
`h exp` *off/on*	disable/enable `readline` history expansion
`h file` *filename*	file for recording GDB command history
`h size` *size*	number of commands kept in history list
`h save` *off/on*	control use of external file for command history
`print` ...	groups with the following options:
`p` ...	
`p address` *on/off*	print memory addresses in stacks, values
`p array` *off/on*	compact or attractive format for arrays
`p demangl` *on/off*	source (demangled) or internal form for C++ symbols
`p asm-dem` *on/off*	demangle C++ symbols in machine-instruction output
`p elements` *limit*	number of array elements to display
`p object` *on/off*	print C++ derived types for objects
`p pretty` *off/on*	struct display: compact or indented
`p union` *on/off*	display of union members
`p vtbl` *off/on*	display of C++ virtual function tables
`show commands`	show last 10 commands
`show commands` *n*	show 10 commands around number *n*
`show commands +`	show next 10 commands

Working Files

`file` [*file*]	use *file* for both symbols and executable; with no arg, discard both
`core` [*file*]	read *file* as coredump; or discard
`exec` [*file*]	use *file* as executable only; or discard
`symbol` [*file*]	use symbol table from *file*; or discard
`load` *file*	dynamically link *file* and add its symbols
`add-sym` *file addr*	read additional symbols from *file*, dynamically loaded at *addr*
`info files`	display working files and targets in use
`path` *dirs*	add *dirs* to front of path searched for executable and symbol files
`show path`	display executable and symbol file path
`info share`	list names of shared libraries currently loaded

Source Files

`dir` *names*	add directory *names* to front of source path
`dir`	clear source path
`show dir`	show current source path
`list`	show next ten lines of source
`list -`	show previous ten lines
`list` *lines*	display source surrounding *lines*, specified as:
[*file*:]*num*	line number [in named file]
[*file*:]*function*	beginning of function [in named file]
`+`*off*	*off* lines after last printed
`-`*off*	*off* lines previous to last printed
`*`*address*	line containing *address*
`list` *f*`,`*l*	from line *f* to line *l*
`info line` *num*	show starting, ending addresses of compiled code for source line *num*
`info source`	show name of current source file
`info sources`	list all source files in use
`forw` *regex*	search following source lines for *regex*
`rev` *regex*	search preceding source lines for *regex*

GDB under GNU Emacs

`M-x gdb`	run GDB under Emacs
`C-h m`	describe GDB mode
`M-s`	step one line (`step`)
`M-n`	next line (`next`)
`M-i`	step one instruction (`stepi`)
`C-c C-f`	finish current stack frame (`finish`)
`M-c`	continue (`cont`)
`M-u`	up *arg* frames (`up`)
`M-d`	down *arg* frames (`down`)
`C-x &`	copy number from point, insert at end
`C-x SPC`	(in source file) set break at point

GDB License

`show copying`	Display GNU General Public License
`show warranty`	There is NO WARRANTY for GDB. Display full no-warranty statement.