Embedded Software Lab
(Labor Eingebettete Software)
565040

Introduction

Christine Jakobs, Martin Richter

Operating Systems Group, TU Chemnitz

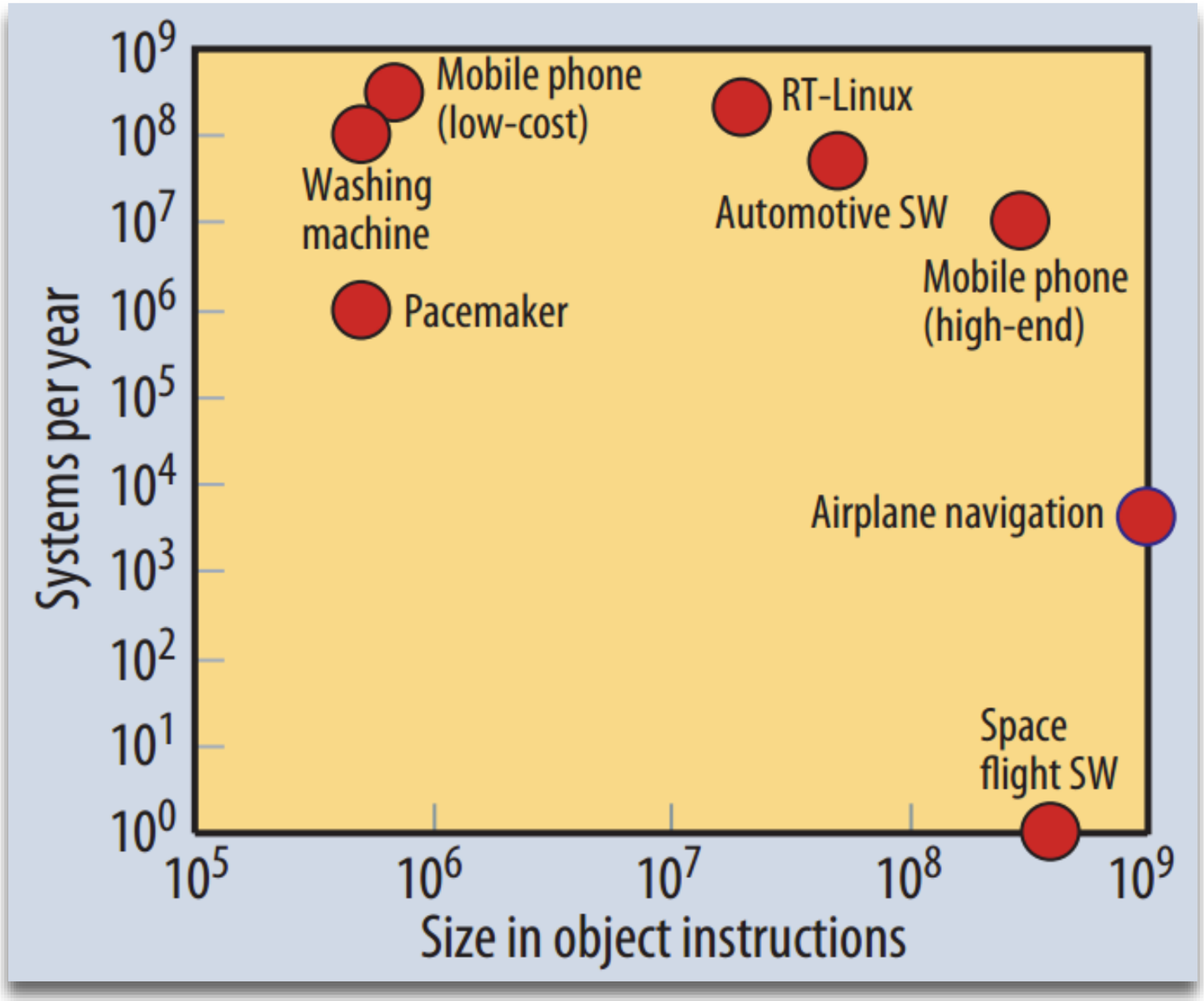
Embedded Systems

- Computer system in a context
  - Specific dedicated task (vs. all-purpose computer)
  - Often hardware/software co-design
  - Optimized design based on application
  - Often non-visible for user
  - Often real-time systems
    (max. response time ≤ deadline)
  - High cost pressure - low memory size, simple CPUs
  - Energy efficiency

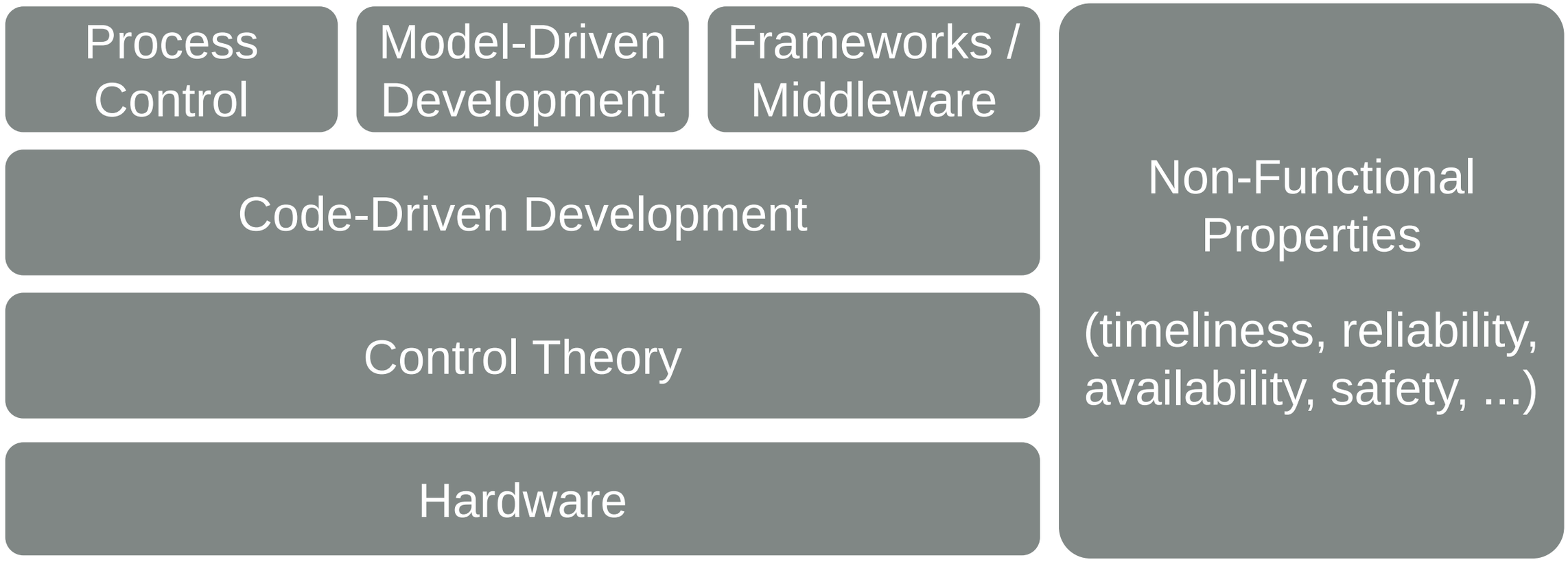
- Everywhere: Cell phones, printers, automobiles, aviation products, household devices, medical devices, children toys, …

Embedded Systems

- Ebert and Jones, 2009
  - Worldwide market of 160 billion €
  - ~30 embedded processors per person in developed countries
  - 98% of all produced microprocessors for embedded applications
- New emerging trends
  - Internet of Things (IoT)
  - Cyber-Physical Systems (CPS)



Design of Software for Embedded Systems



- Course 565050
  - Mandatory precondition for this lab
  - Gives you the broad overview of embedded software development
  - Now we dig deeper ....

Embedded Software Lab

„Das Praktikum beschäftigt sich mit der angewandten Modellierung, Analyse und Entwicklung eingebetteter Software-Systeme. Dabei werden folgende Teilaspekte behandelt :

• Umgang mit Werkzeugen zur Entwicklung eingebetteter Software (Cross Compiler, grafische Entwicklungsumgebungen, Debugger, Werkzeuge für den Erstellungsprozess)

• Modellierung und Analyse von Algorithmen und Architekturen für eingebettete Systeme

• Fehlerbehebung und Laufzeitanalyse für eingebettete Software

Qualifikationsziele: Spezialisierte anwendungsnahe Fähigkeiten zum Entwurf und der Entwicklung eingebetteter Software“

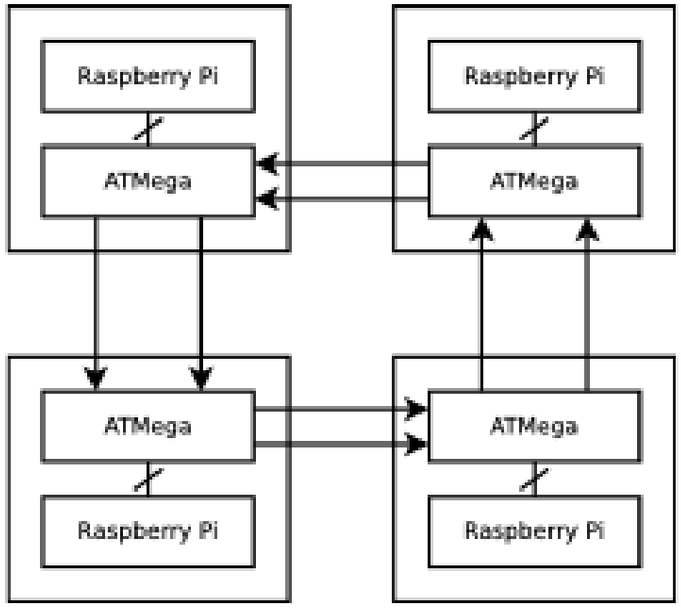
- Applied development and analysis of software for embedded systems
- Focus on tools, architectures, fault finding, and runtime analysis

- Qualification goals
  - Specialized knowledge in embedded software development
  - Working close to applications

- Still very broad
- Complexity issues with industry use cases

RASPNet

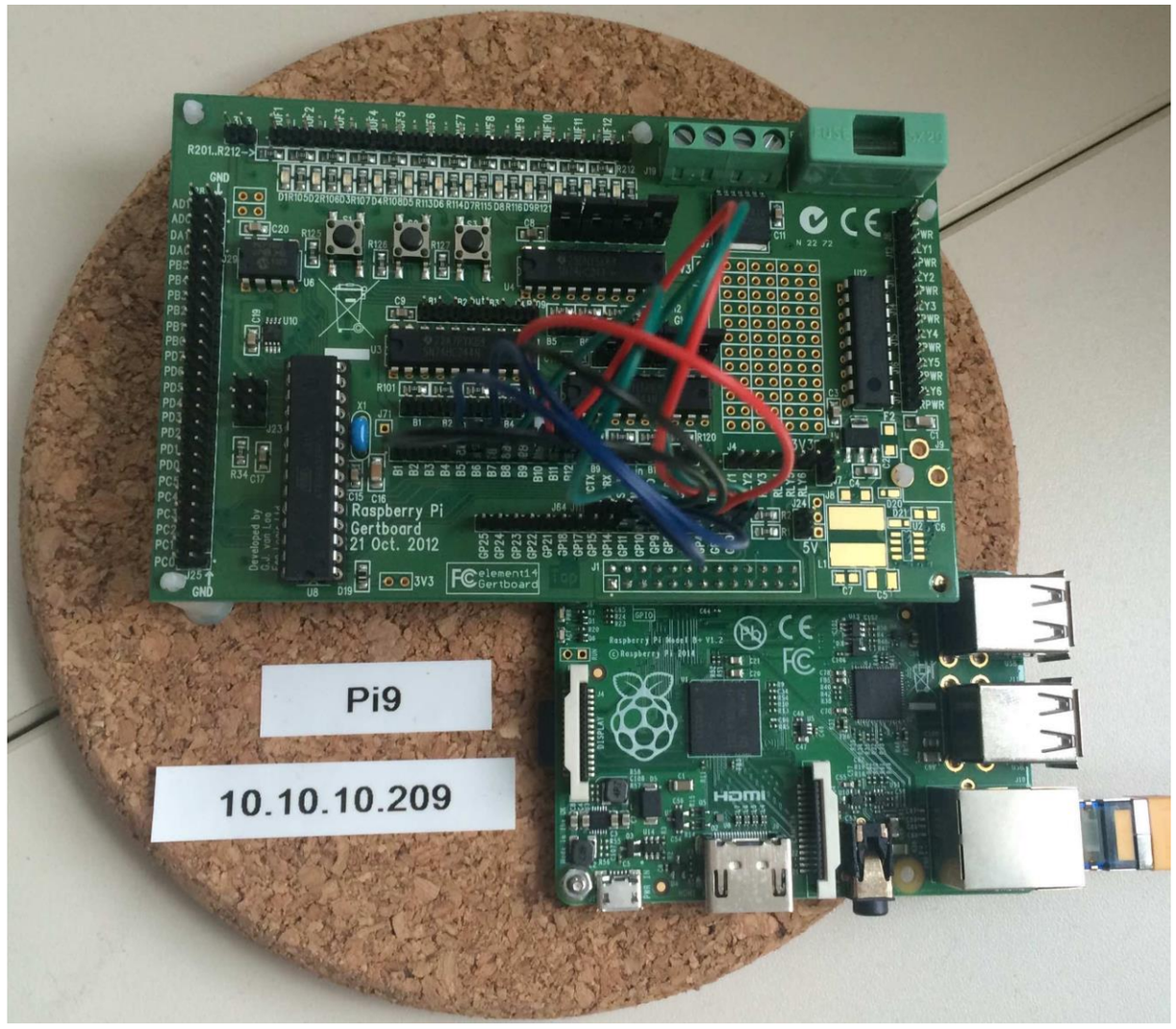
- Serial real-time protocol, for education purposes only
- Overall task: Running RASPNet on a microcontroller
  - Bare metal implementation on 8-Bit ATMega in C
  - Restricted resources during execution
  - Restricted tools and capabilities for debugging and testing
  - Hard real-time constraints
  - Ultimate goal of interoperability



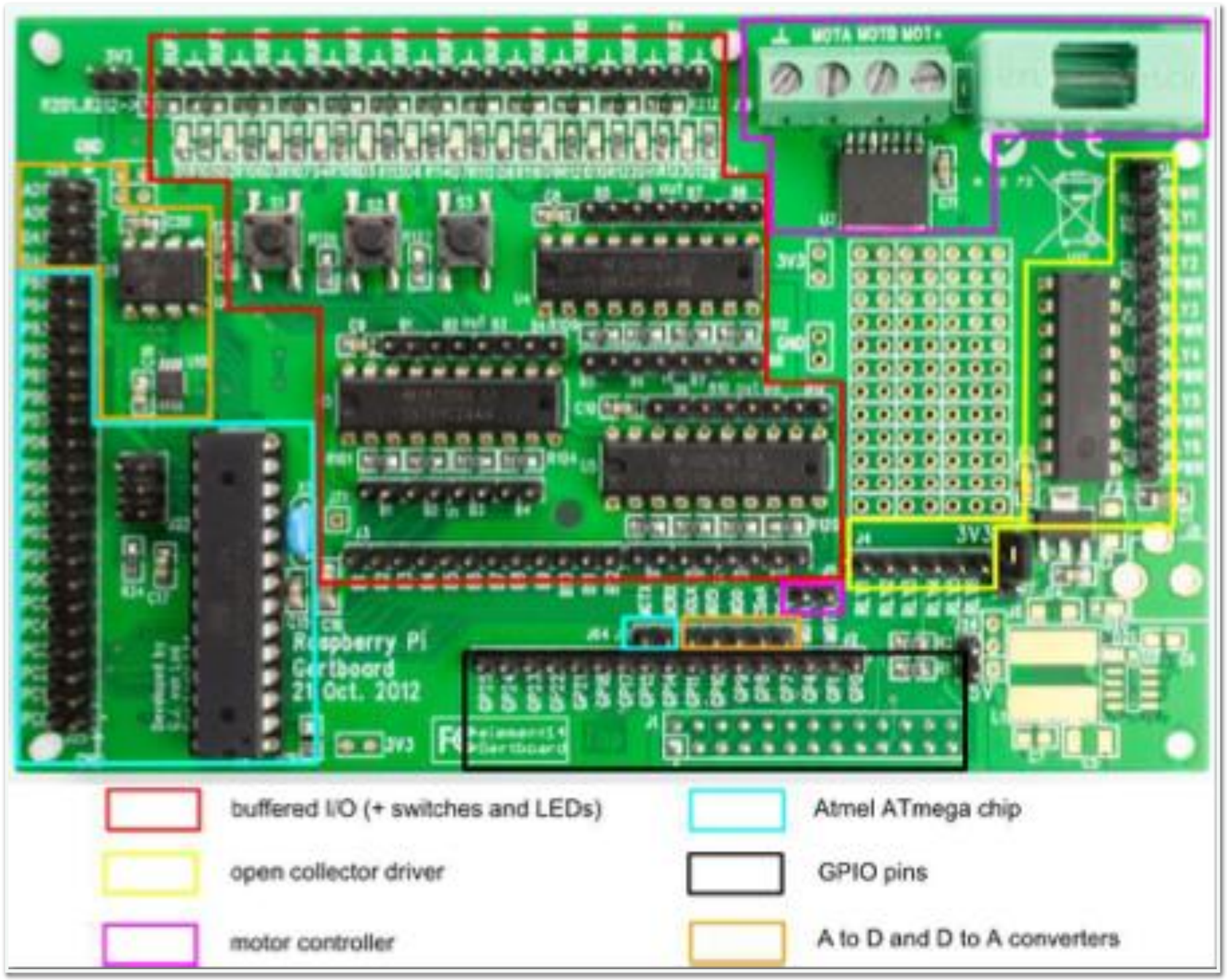
- Development work guided by bi-weekly task sets
- Protocol description and add-on sources will be available on-line
- Beyond that, you are on your own ...

Hardware

- 14 boards
  - RaspberryPi B+
    - Standard Rasbian installation
    - AVR compiler tool chain
    - Used as development environment
  - Gertboard v2.0
    - Extension board
    - Buffered I/O
    - Pushbuttons
    - Open collector drivers
    - Motor controller
    - 28pin ATMega microcontroller
    - D/A, A/D converters
    - Target for your implementation efforts



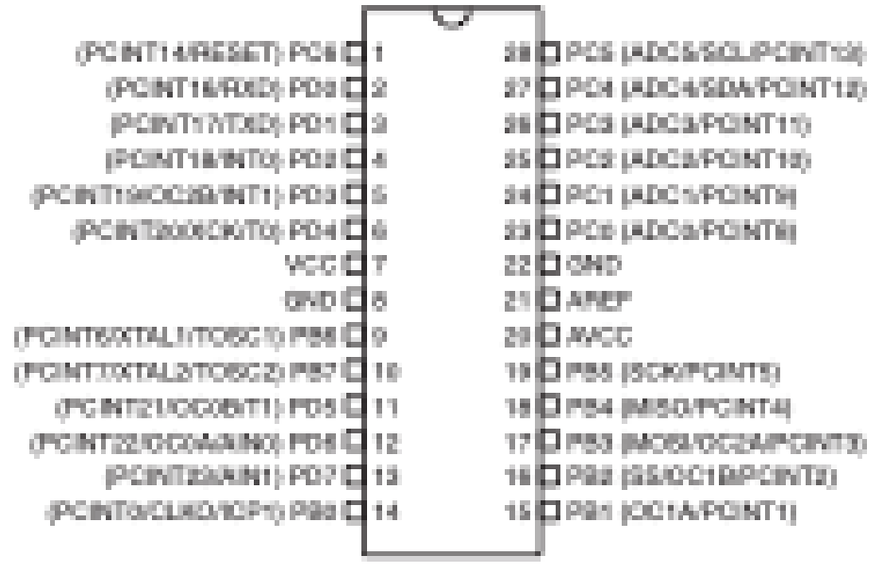
Gertboard



[Gertboard Manual]

Microcontroller@Gertboard

- Atmel AVR microcontroller
  - ATmega328/P, 8 bit RISC Harvard, 12 MHz, 3.3V (!) power from the RaspberryPi
  - 32kByte Flash (code), 1kByte EEPROM (storage), 2kByte SRAM (data)
  - 2x 8-bit timer, 1x 16-bit timer, support for interrupt on pin change
  - Same hardware as on the popular Arduino hardware
  - Upload of software via SPI bus
  - Input / output monitoring over serial connection
  - Check Gertboard manual for further information
- Task descriptions contain a wiring plan
  - Your responsibility to not break the hardware
  - Beside that, toying around is fine



Dates

- Initial lecture: Today (17.10.2019)
- Lab 1 (24.10.2019)
  - Get together
  - Sub-project 1 starts: Hello ATMega!
- Lab 2 (07.11.2019)
  - Presentation of sub-project 1
  - Sub-project 2 starts: RASPNet Layer 1+2
- Lab 3 (21.11.2019)
  - Presentation of sub-project 2
  - Sub-project 3 starts: RASPNet Layer 3
- Lab 4 (05.12.2019)
  - Presentation sub-project 3
  - Sub-project starts: Organize Project
- Lab 5 (19.12.2019)
  - Presentation of gaps
  - Sub-project 4 starts:RASPNet Layer 4
- Lab 6 (09.01.2020)
  - Presentation of sub-project 4
  - Sub-project 5 starts: PlugFest
- Lab 7 (?)
  - Final presentation

BUT: There will probably be changes depending on the overall progress!

Organization

- Lab slots
  - Question round in the beginning
  - Code review afterwards
  - Not showing up without noticing us: YOU ARE OUT!
  - Beside the official slots, you are expected to spend >100h with coding
  - Lab slots are mainly for presentation and discussion
  - The room (1/B208) is all yours in this semester
- Code
  - Project code must be managed in a Git repository
  - Grading based on code commenting, structuring, consideration of feedback and completeness (e.g. Makefile)
- Grade is composed of assessment of your final code as well as your performance in the lab slots

Help?!?

- Information exchange during the lab slots and in a discussion forum
- Supervisors will not solve your problems
  - Learn how to debug over serial connections
  - Learn how to write error-free C code from the very beginning
  - Learn to live without visual debuggers / development environments
- Reliable sources: Data sheets, manufacturer manuals, man pages
  - Proper citing in the code expected
  - Post your insights in the discussion forum
- Tricky sources: Blogs, code examples, Stack Overflow, forums, cheating

Skills

- Lab work relies heavily on self-motivation
- Need for existing capabilities, or willingness to get them very fast
  - Reasonably good programming skills in C
  - Basic understanding of how a CPU works, e.g. pipelining
  - Basic knowledge about (Unix) operating systems, e.g. memory-mapped I/O
  - Working with source code versioning systems, especially Git
- Skills being developed during this course
  - Reading and understanding detailed documentation of hardware chips
  - „Survive“ without graphical user interfaces
  - Debugging code that is out of your control
  - Low-level I/O programming

Clean Code – How to do it

- The code explains itself
  - Useful names of variables, functions, …
  - Avoid complex expressions
  - Correct indentation
- Use comments whenever the code is difficult to understand
- Avoid duplicated code
  - Each functionality is in exactly one function
  - One function has exactly one functionality
- Ensures reusability and maintainability

Clean Code – Bad Example

```
struct p {
    int a;
    char n[];
};

struct p *cp(int a, char n[]) {
    int i;
    struct p *np = malloc(sizeof(struct p));
    np->a=a;
    for(i=0;*(n+i)!=0;i++)
    *(np->n+i)=*(n+i);
    *(np->n+i)=0;
    printf("%s %d\n",np->n,np->a);
    return(np);
}
```

Clean Code – Bad Example

- What is this supposed to do?
  - Output?
  - Memory allocation for „some kind of structure“?
  - Side effects?
- What is happening?
- WTF?!?!!?

```
struct p {
    int a;
    char n[];
};

struct p *cp(int a, char n[]) {
    int i;
    struct p *np = malloc(sizeof(struct p));
    np->a=a;
    for(i=0;*(n+i)!=0;i++)
    *(np->n+i)=*(n+i);
    *(np->n+i)=0;
    printf("%s %d\n",np->n,np->a);
    return(np);
}
```

Clean Code – How to do it

```
struct Person {
    int age;
    char name[MAX_LEN];
};

struct Person * createPerson(int age, char name[]) {
    struct Person *newPerson = malloc(sizeof(struct Person));
    newPerson->age = age;
    strcpy(newPerson->name, name);
    return (newPerson);
}

void showPerson(struct Person person) {
    printf("%s %d\n", person.name, person.age);
}
```

Daily work

- Leave no traces on the Raspberry SD cards
  - Bad students will steal your code
  - The lab is a shared resource, clean up your mess
- Learn to love your Git repository
  - Create an according private repository on the TU Chemnitz GitLab service
  - Give Ms. Jakobs and Mr. Richter read-only access
  - Git command-line tools are available in the Rasbian installation
  - Make use of tags and branches
- Final version of your repository must support Doxygen documentation
  - Code modules, function signatures, source files

Lab 1

- Create a first binary, get familiar with the tools
  - Blinking of Gertboard LEDs
  - Driven by program running on the ATMega, independent from RaspberryPi operation
  - Demands wiring between ATMega pins and the Gertboard LEDs
  - Demands wiring between ATMega pins and the RaspberyPi for serial connection + flashing
- Task set 1 becomes official on the first lab slot
- Preview available online
- But first, you must be allowed to participate in this course ...

Course Restriction

- Only 14 seats
  - 14 RaspberryPi work stations, with monitor, mouse and keyboard
- Course only works with pre-existing knowledge
  - Study regulations expect 565050 being passed
  - Also needed: Skills in operating systems, real-time and low-level software development
  - There is no time to teach you these basics here
- Too many people in the room? Multiple choice test will happen now.
  - You are intentionally unprepared
  - Everybody gets according feedback by eMail
  - If you win, you will be able to open B/208 with your TUC card