Embedded Systems

1 - Introduction

© Lothar Thiele
Computer Engineering and Networks Laboratory
Lecture Organization
Organization

WWW: https://www.tec.ee.ethz.ch/education/lectures/embedded-systems.html
Lecture: Lothar Thiele, thiele@ethz.ch
Coordination: Rehan Ahmed, rehan.ahmed@tik.ee.ethz.ch
References:


What will you learn?

- Theoretical foundations and principles of the analysis and design of embedded systems.
- Practical aspects of embedded system design, mainly software design.

The course has three components:

- **Lecture**: Communicate principles and practical aspects of embedded systems.
- **Exercise**: Use paper and pencil to deepen your understanding of analysis and design principles.
- **Laboratory (ES-Lab)**: Introduction into practical aspects of embedded systems design. Use of state-of-the-art hardware and design tools.
When and where?

### Schedule

<table>
<thead>
<tr>
<th></th>
<th>When</th>
<th>Where</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lectures</td>
<td>Wed. 13:15 - 15:00</td>
<td>ETF C1</td>
</tr>
<tr>
<td>Exercises</td>
<td>Wed. 15:15 - 17:00</td>
<td>ETF C1</td>
</tr>
<tr>
<td>Labs</td>
<td>Wed. 15:15 - 17:00</td>
<td>ETZ D61.1/2, D96</td>
</tr>
<tr>
<td></td>
<td>Wed. 17:15 - 19:00</td>
<td></td>
</tr>
</tbody>
</table>

### Timetable

<table>
<thead>
<tr>
<th>Date</th>
<th>Lecture</th>
<th>Exercise</th>
<th>Lab</th>
</tr>
</thead>
<tbody>
<tr>
<td>21.2.2018</td>
<td>1. Introduction (PDF, 8.2 MB) ↓</td>
<td></td>
<td>0. Prelab [RJ, XH, YC]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Supplementary Material →</td>
</tr>
<tr>
<td>28.2.2018</td>
<td>2. Software Development (PDF, 1.8 MB) ↓</td>
<td>3. Hardware-Software Interface ↓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.3.2018</td>
<td>3. Hardware-Software Interface ↓</td>
<td>1. SPI, I/O, Polling vs Interrupt [RA]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Please register!

- https://www.tec.ee.ethz.ch/education/lectures/embedded-systems.html

Laboratory exercises will be conducted in teams of 2 people. Due to the large number of students, there will be two successive ES-Lab sessions at the days of the ES-Lab (see timetable): One from 3pm - 5pm and one from 5pm - 7pm. Both sessions will take place in rooms D61.1 / D61.2 and D96.

You are required to specify your lab session preference using the following form: Lab Registration.

This form has to be filled by 22.02.2018 for your preference to be considered.

**ES-Material**

The lab registration form can also be used to order print out of the course material (lecture slides and ES-Lab material). Printed copies are sold for CHF 10 and are distributed in the second lecture (on 28.02.2018).
What you get after the break...
Be careful and please do not ...
You have to return the board at the end!
Embedded Systems - Impact
Often, the main reason for buying is not information processing

**Embedded Systems**

Embedded systems (ES) = information processing systems embedded into a larger product

Examples:

![Embedded Systems Examples](image-url)
Many Names – Similar Meanings

- Internet of Everything
- Smarter Planet
- Machine to Machine (M2M)
- The Fog
- Internet of Things (IoT)
- Industry 4.0
- The Industrial Internet
- TSensors (Trillion Sensors)
- Cyber-Physical Systems

© Edward Lee
The Hype Cycle

The Hype Cycle

Internet of Things

Trough of Disillusionment

Use feedback to influence the dynamics of the physical world by taking smart decisions in the cyber world.
Predictability & Dependability

CPS = cyber-physical system

“It is essential to predict how a CPS is going to behave under any circumstances [...] before it is deployed.”\textsuperscript{Maj14}

“CPS must operate dependably, safely, securely, efficiently and in real-time.”\textsuperscript{Raj10}

Efficiency & Specialization

- Embedded systems must be **efficient**:  
  - *Energy* efficient  
  - *Code-size* and *data memory* efficient  
  - *Run-time* efficient  
  - *Weight* efficient  
  - *Cost* efficient

Embedded Systems are often **specialized** towards a certain application or application domain:

- Knowledge about the expected behavior and the system environment at design time is exploited to **minimize resource usage** and to **maximize predictability and reliability**.
Reactivity & Timing

Embedded systems are often reactive:

- Reactive systems must react to stimuli from the system environment:

  “A reactive system is one which is in continual interaction with its environment and executes at a pace determined by that environment” [Bergé, 1995]

Embedded systems often must meet real-time constraints:

- For hard real-time systems, right answers arriving too late are wrong. All other time-constraints are called soft. A guaranteed system response has to be explained without statistical arguments.

  “A real-time constraint is called hard, if not meeting that constraint could result in a catastrophe” [Kopetz, 1997].
## Comparison

**Embedded Systems:**
- Few applications that are known at design-time.
- Not programmable by end user.
- Fixed run-time requirements (additional computing power often not useful).
- Typical criteria:
  - cost
  - power consumption
  - size and weight
  - dependability
  - worst-case speed

**General Purpose Computing**
- Broad class of applications.
- Programmable by end user.
- Faster is better.
- Typical criteria:
  - cost
  - power consumption
  - average speed
Lecture Overview

1. Introduction to Embedded Systems
2. Software Development
3. Hardware-Software Interface
4. Programming Paradigms
5. Embedded Operating Systems
6. Real-time Scheduling
7. Shared Resources
8. Hardware Components
9. Power and Energy
10. Architecture Synthesis
Components and Requirements by Example
Components and Requirements by Example
- Hardware System Architecture -
High-Level Block Diagram View

**low power CPU**
- enabling power to the rest of the system
- battery charging and voltage measurement
- wireless radio (boot and operate)
- detect and check expansion boards

**higher performance CPU**
- sensor reading and motor control
- flight control
- telemetry (including the battery voltage)
- additional user development
- USB connection

**UART:**
- communication protocol (Universal Asynchronous Receiver/Transmitter)
- exchange of data packets to and from interfaces (wireless, USB)
High-Level Block Diagram View

**Acronyms:**
- Wkup: Wakeup signal
- GPIO: General-purpose input/output signal
- SPI: Serial Peripheral Interface Bus
- I2C: Inter-Integrated Circuit (Bus)
- PWM: Pulse-width modulated Signal
- VCC: power-supply

**EEPROM:**
- electrically erasable programmable read-only memory
- used for firmware (part of data and software that usually is not changed, configuration data)
- can not be easily overwritten in comparison to Flash

**Flash memory:**
- non-volatile random-access memory for program and data
High-Level Physical View

- **nRF51822**
  - 16MHz Cortex-M0
  - 16kB RAM, 256KB Flash
  - BLE and NRF radio

- **10DOF IMU**
  - 3-axis accelerometer
  - 3-axis gyro
  - 3-axis magnetometer
  - Pressure sensor

- **STM32F405**
  - 168MHz Cortex-M4
  - 196kB RAM, 1MB Flash

- **Push button**

- **Power supplies and battery charger**

- **RF power amplifier**

- **Motor driver**

- **Expansion port**

- **EEPROM**

- **µUSB port**

- **USB Data to STM32**

- **Charge/VBAT/VCC**

- **Wkup/OW/GPIO**

- **PWM**

- **I2C**

- **SPI/I2C/GPIO/PWM**

Crazyflie 2.0 system architecture
High-Level Physical View

10DOF IMU
- 3-axis accelerometer
- 3-axis gyro
- 3-axis magnetometer
- Pressure sensor

STM32F405
- 168MHz Cortex-M4
- 196kB RAM, 1MB Flash

Motor driver

EEPROM

Crazyflie 2.0 system architecture
Low-Level Schematic Diagram View

Motors
High-Level Software View

- The software is built on top of a *real-time operating system* “FreeRTOS”.
- We will use the same operating system in the ES-Lab ... .

![Diagram showing the software stack with Crazyflie Software, FreeRTOS, FreeRTOS Hardware-Interface Code, and Hardware (STM32F103CB)]
High-Level Software View

The software architecture supports

- **real-time tasks** for motor control (gathering sensor values and pilot commands, sensor fusion, automatic control, driving motors using PWM (pulse width modulation, ... ) but also
- **non-real-time tasks** (maintenance and test, handling external events, pilot commands, ... ).

**Control System:**

PID controller (proportional–integral–derivative)

- Pilot commands (from wireless interface)
- Update frequencies (periodic task execution)
High-Level Software View

More *detailed block diagram* of the stabilization system:

- **MPU6050 Gyro**
  - Set to: 1000 deg/s
  - Sample rate: 8 kHz
  - Lowpass filter: 256 Hz

- **MPU6050 Accel**
  - Set to: 8 g
  - Acc sample rate: 1 kHz
  - Lowpass filtered at 260 Hz

- Sensor reading & analog-digital conversion on sensor component
- Transfer to processor

- Variance calculation and logic to take bias
- Bias

- Axis mapping
- Sampled value converted to deg/s

- First order lowpass filtered at 60 Hz
- Sampled value converted to g

- Sensor fusion filter

- Stabilization
  - Commander (pilot control)
  - Actuator output
- Motors

- Information extraction from sensors
- Automatic control
- Actuation
Components and Requirements by Example
- Processing Elements -
What can you do to increase performance?
From Computer Engineering 1

MOORE’S LAW

transistors

10,000,000,000

10,000,000

1,000,000

1,000


Intel 62-Core Xeon Phi
From Computer Engineering 1:

AMD multicore RYZEN
From Computer Engineering 1

Intel Xeon Phi 7290
(8 Milliarden Transistoren,
14nm Technologie,
~ 650mm² Fläche,
1.5 GHZ Taktfrequenz)
What can you do to decrease power consumption?
Recent developments:

- Specialize multicore processors towards real-time processing and low power consumption
- Target domains:

<table>
<thead>
<tr>
<th>Core Generation</th>
<th>Number of Processing Cores</th>
<th>GFLOPS/W</th>
<th>GOPS/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andey</td>
<td>256</td>
<td>25</td>
<td>75</td>
</tr>
<tr>
<td>Bostan (2014)</td>
<td>256</td>
<td>50</td>
<td>80</td>
</tr>
<tr>
<td>Coolidge (2015)</td>
<td>64/256/1024</td>
<td>75</td>
<td>115</td>
</tr>
</tbody>
</table>
Why does higher parallelism help in reducing power?
System-on-Chip

*Samsung Galaxy S6*

- Exynos 7420 System on a Chip (SoC)
- 8 ARM Cortex processing cores (4 x A57, 4 x A53)
- 30 nanometer: transistor gate width
Components and Requirements by Example
- Systems -
Zero Power Systems and Sensors
Zero Power Systems and Sensors


Trends ...

- *Embedded systems are communicating with each other*, with servers or with the cloud. Communication is increasingly wireless.

- **Higher degree of integration** on a single chip or integrated components:
  - Memory + processor + I/O-units + (wireless) communication.
  - Use of networks-on-chip for communication between units.
  - Use of homogeneous or heterogeneous multiprocessor systems on a chip (MPSoC).
  - Use of integrated microsystems that contain energy harvesting, energy storage, sensing, processing and communication (“zero power systems”).
  - The complexity and amount of software is increasing.

- *Low power and energy constraints* (portable or unattended devices) are increasingly important, as well as temperature constraints (overheating).

- There is increasing interest in *energy harvesting* to achieve long term autonomous operation.