Welcome to
Concepts and Software Design for CPS

- Class documents: they will be posted on Moodle

- Lecture Instructor: Marco Caccamo
- mcaccamo@tum.de
- Office: Gebäude 1 – Raum 2110 (Mechanical Engineering)
- Office Hour: Wednesday at 6.45pm (Immediately, after lecture)

- Lab Instructors: Raphael Trumpp, Binqi Sun
- Email: raphael.trumpp@tum.de, binqi.sun@tum.de

- The course includes weekly lecture and Lab exercise.

- Examination: Written exam (100%) and laboratory assignments (fail/pass)

Lab

- ~~All labs are in U150, basement of mechanical engineering.~~
- **You need to register on TUMonline for the course "Tutorial Concepts and Software Design for Cyber-Physical Systems" (Course No. 4355). Registration is open now!**
- We offer three identical Lab sessions every week. You will attend one of them.
  - Thursdays 10:30 – 12:00 (Session 1)
  - Thursdays 13:00 – 14:30 (Session 2)
  - Thursdays 14:30 – 16:00 (Session 3)
- During the registration, please order the sessions according to your preference. You will be automatically assigned to a session based on your preference and your registration priority.

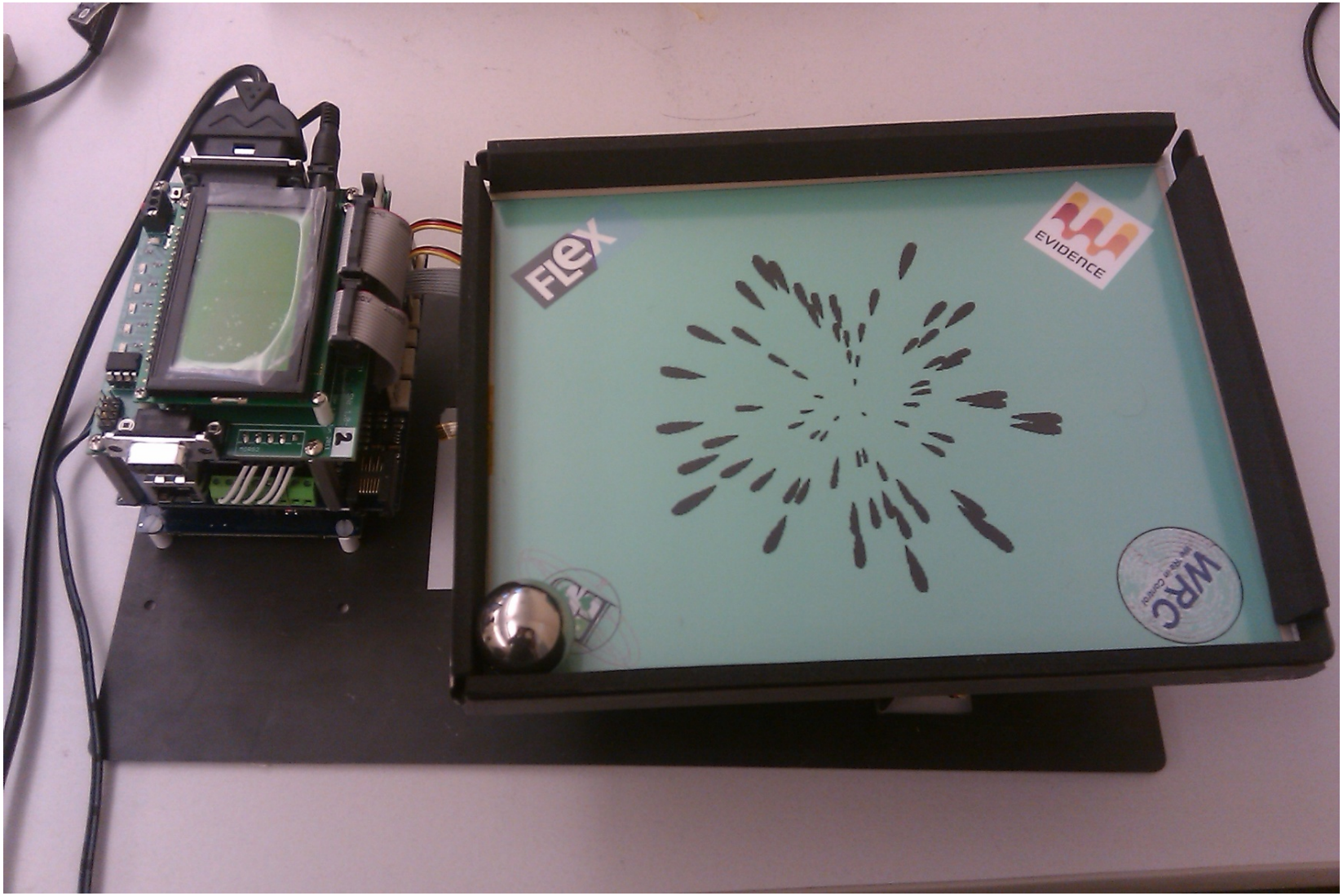
Organize Yourself into Teams for the Lab

- First lab meeting will take place remotely this Thursday (21.10) via **zoom** at times mentioned in the previous slide.
- Zoom link and password for Lab sessions are posted on Moodle.
- Please register for the tutorials on TUMonline **now**! Successful registrations and your corresponding session will be announced at 10:00pm today.
- All material related to the lab sessions will be posted in Moodle
- ~~Students can go to lab at any times to finish their work~~

Amazing Ball Embedded Board
(If COVID-19 permits, next semester)

- **Embedded device for MW2411 Lab:**
- Built around the 16-bit dsPIC33F microcontroller from Microchip
  - 12.8 MHz clock
  - 259KB of flash program memory
  - 30,720 bytes of SRAM
- A 128x64 pixels LCD module
- A 4-wire touchscreen
- Digital to analog and analog to digital converters
- Servo motors
- Serial port
- Joystick port
- And more

Amazing Ball Embedded Board



Final exam and Lab projects

- **Date for Final exam**: **Feb 23 at 5pm**

- The course examination includes:
  - Written final exam (weighting 100%, processing time 180 min, permitted aids are a handwritten, double-sided A4 sheet and a scientific calculator)
  - Laboratory projects (pass/fail). In the course of this, students work on programming tasks that demonstrate that they can design and analyze time-sensitive software for cyber-physical applications.

- You need a **passing grade** in the Lab assignments in order to register to the written exam.

Reference Books (optional)

- This class covers a wide range of selected topics that are important for practicing embedded system software engineers. There is no single textbook that even comes close to covering all the material covered in this class. As a result, there is no mandatory textbook.

- Two following books are good reference books, they are **optional** for the course and they cover some of the teaching material.

- 1) POSIX.4 Programmers Guide
- Programming for the Real World
- Bill Gallmeister
- http://www.oreilly.com/catalog/posix4/index.html

- 2) HARD REAL-TIME COMPUTING SYSTEMS:
- Predictable Scheduling Algorithms and Applications
- Giorgio C. Buttazzo
- http://shop.store.yahoo.com/softpro/0-387-23137-4.html

MW2411 : The Big Picture - 1

- Embedded real time computing systems enable us to:
  - manage the vast power generation and distribution networks.
  - control industrial processes for chemicals, fuel, medicine, and manufactured products.
  - control automobiles, ships, trains and airplanes.
  - conduct video conferencing over the Internet and interactive electronic commerce.
  - send vehicles high into space and deep into the sea to seek new knowledge.

- It is a challenging and exciting area in either the design of embedded hardware or the software systems.

- MW2411 will focus on the software engineering aspect.

MW2411 : The Big Picture - 2

- MW2411 is designed to help you master the software engineering aspects of embedded software systems.
  - The basics of micro-computer and how it is interfaced with the external world
    - how to program A/D and D/A cards
    - interrupts, serial communication, etc.
    - how to drive servo motors, sample a touch-screen
  - Tools of the trade
    - RTOS, timers and periodic tasks
    - programming with multi-tasking real-time operating system
  - Theory of the trade: How to analyze and design concurrent real time tasks
    - periodic tasks, aperiodic tasks, synchronization etc.
- MW2411 also teaches the key concepts of the common application domains
    - signal processing
    - feedback control
  - so that you can work with communication and control engineers effectively.

The Most Important Skill

MW2411 is an important course. However, learning how to learn effectively is the most important skill that you can acquire during college years:

- In engineering, new technologies and new knowledge are generated rapidly. Every 3-5 years, we need to significantly revise our course material to reflect the changes.
- You must become an effective and efficient learner to get ahead.

Preparation for Exam

- Not all materials are equally important. Educators will try to make sure that
  - Key concepts, results and techniques constitute the major part of the exam
  - There is a reasonable distribution across the subject areas.

- How do I know what are the key concepts, results and techniques?
  - Educators will try their best to tell you that “this is really fundamental/important/ critical during the lectures”
  - They will try to make you work on them in the lab.
  - They will repeat them and review them to make sure that you hear them more than once.
  - They will test you if you have mastered them.

Background knowledge: brief review

- Brief review
  - Basic data types
  - binary/decimal/hexadecimal
  - Bitwise operations

Data Types

- uint8_t 8-bit unsigned integer
- uint16_t 16-bit unsigned integer
- uint32_t 32-bit unsigned integer
- uint64_t 64-bit unsigned integer
- int8_t 8-bit signed integer
- int16_t 16-bit signed integer
- int32_t 32-bit signed integer
- int64_t 64-bit signed integer
- char 8-bit (signed) character
- double 32-bit floating point value

- Should I be careful when choosing the data type?

Constants: binary / decimal / hexadecimal

- What is the difference between these assignments?

```
i =       42;
i =     0x2a;
i = 0b101010;
```

Constants: binary / decimal / hexadecimal

- You should be able to convert between binary and hexadecimal quickly.

decimal	hexadecimal	binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Bitwise Operations (on unsigned char*)*

- ~ Bitwise inverse (one's complement)
  ~00101111 → 11010000
- & Bitwise AND
  01010101 & 11110000 → 01010000
- | Bitwise OR
  01010101 | 11110000 → 11110101
- ^ Bitwise XOR (exclusive OR)
  01010101 ^ 11110000 → 10100101
- << Left shift
  00101111 << 2 → 10111100
- >> Right shift
  00101111 >> 2 → 00001011

Bitwise Examples

- x = y << 2; // Shift the value of y by 2 bits to the left and store
  // the result in x
- x = x | 0x01; // Set the least significant bit in x to 1
- x |= 0x01; // Set the least significant bit in x to 1
- x &= 0xfe; // Set the least significant bit in x to 0
- x &= ~0x01; // Set the least significant bit in x to 0
- x ^= 0x01; // ???

Bits and Masks

- A *n*-bit value will have bits numbered consecutively from 0 through *n* - 1. Bit 0 is the least significant bit in the value. Bit *n* - 1 is the most significant bit.

- The **BV(*i*)** macro returns a mask consisting of all 0s, except for bit *i* which is 1. Mathematically, **BV(*i*)** is equivalent to 2^i.

Macro	Binary	Hex	Decimal
BV(0)	0000000000000001	0x0001	$2^0 = 1$
BV(1)	0000000000000010	0x0002	$2^1 = 2$
BV(2)	0000000000000100	0x0004	$2^2 = 4$
BV(3)	0000000000001000	0x0008	$2^3 = 8$
BV(4)	0000000000010000	0x0010	$2^4 = 16$
BV(5)	0000000000100000	0x0020	$2^5 = 32$
BV(6)	0000000001000000	0x0040	$2^6 = 64$
BV(7)	0000000010000000	0x0080	$2^7 = 128$
....			
BV(15)	1000000000000000	0x8000	$2^{15} = 32768$

Bit Manipulation Examples with BV()

- `x |= BV(4);` // Set bit 4 in x to 1
- `x &= ~BV(4);` // Set bit 4 in x to 0
- `x ^= BV(4);` // Toggle bit 4 in x

- `x |= BV(2) | BV(4) | BV(5);` // Set bits 2, 4, and 5 in x to 1
- `x &= ~(BV(2) | BV(4) | BV(5));` // Set bits 2, 4, and 5 in x to 0
- `x ^= BV(2) | BV(4) | BV(5);` // Toggle bits 2, 4, and 5 in x

- `x &= ~BV(6);` // This line sets bit 6 in x to 0

Class exercise: bit manipulation

- Let's assume that X and Y are unsigned:
- X=10011101
- Y=11111000

- X |= (Y & BV(3)) >> 3 << 6;

- What are the final values of X and Y?