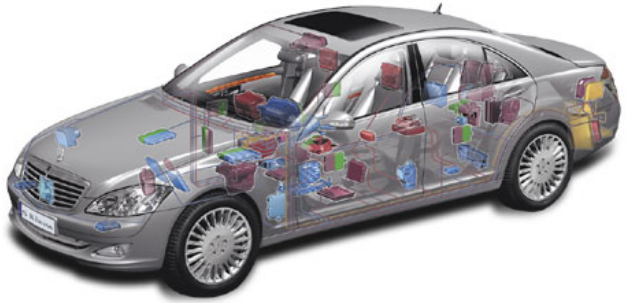
Embedded Software

CSE2425

1. Introduction

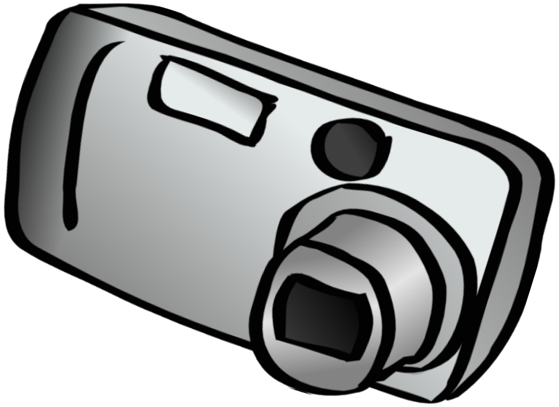


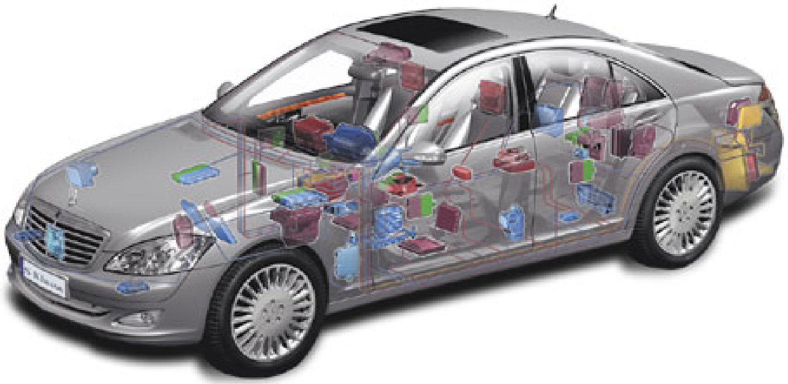
Koen Langendoen
Qing Wang
Embedded & Networked Systems Group

Embedded System – Definition

- Many different definitions, some of them:
  - A computer system with a dedicated function within a larger mechanical or electrical system
  - …, often with real-time computing constraints
  - A computing system that fulfills the task of monitoring and controlling the technical context
  - Without the computing system, the whole system is useless

Examples





Embedded Software – Definition

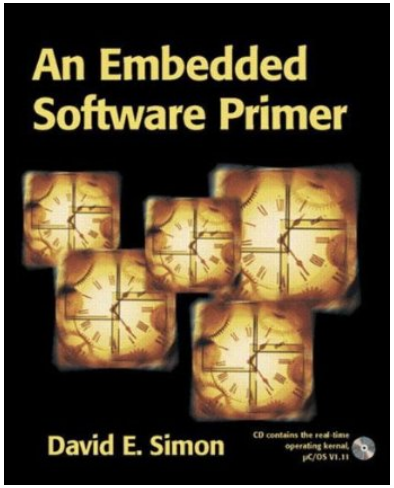
- Many different definitions, some of them:
  - Computer **software** with a dedicated function within a larger mechanical or electrical system
  - …, often with real-time computing constraints
  - A computer **program** that fulfills the task of monitoring and controlling the technical context
  - Without the right **firmware**, the whole system is useless

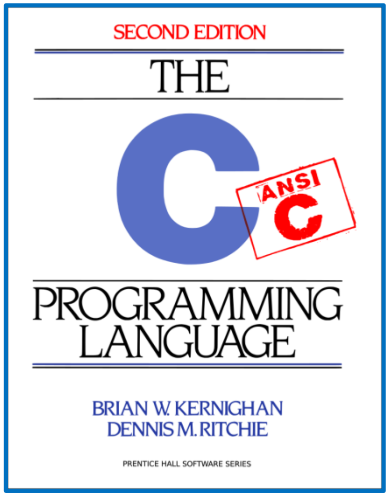
In this course …

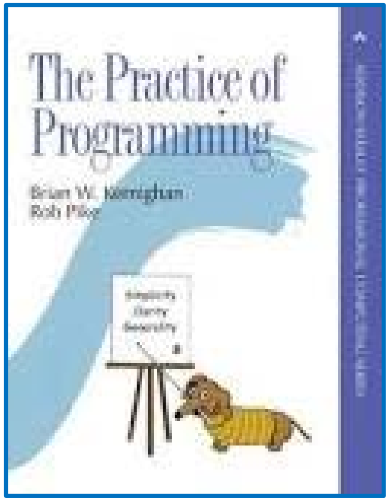
- You will learn about:
  - Programming of embedded system
  - Real-time programming with RTOSs
- We will explore:
  - Principles of “good” embedded systems design
  - Time and complexity
- You will engage in low-level programming:
  - C language
  - STM32F103C8T6 microcontroller platform

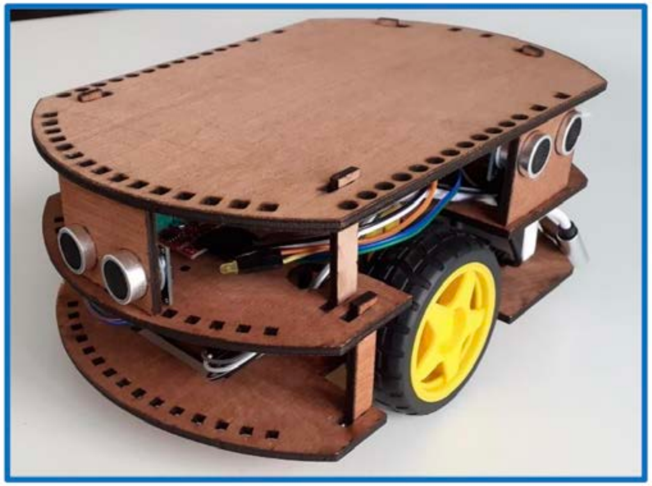
Course setup

CSE2425	2021-2022
Credit points	5 EC
Lectures	11
Exam	Chap 1, 4-10 + lect. notes C, FSM
Lab work	C + Robot



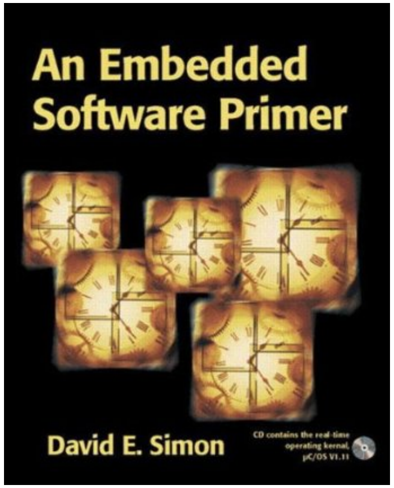




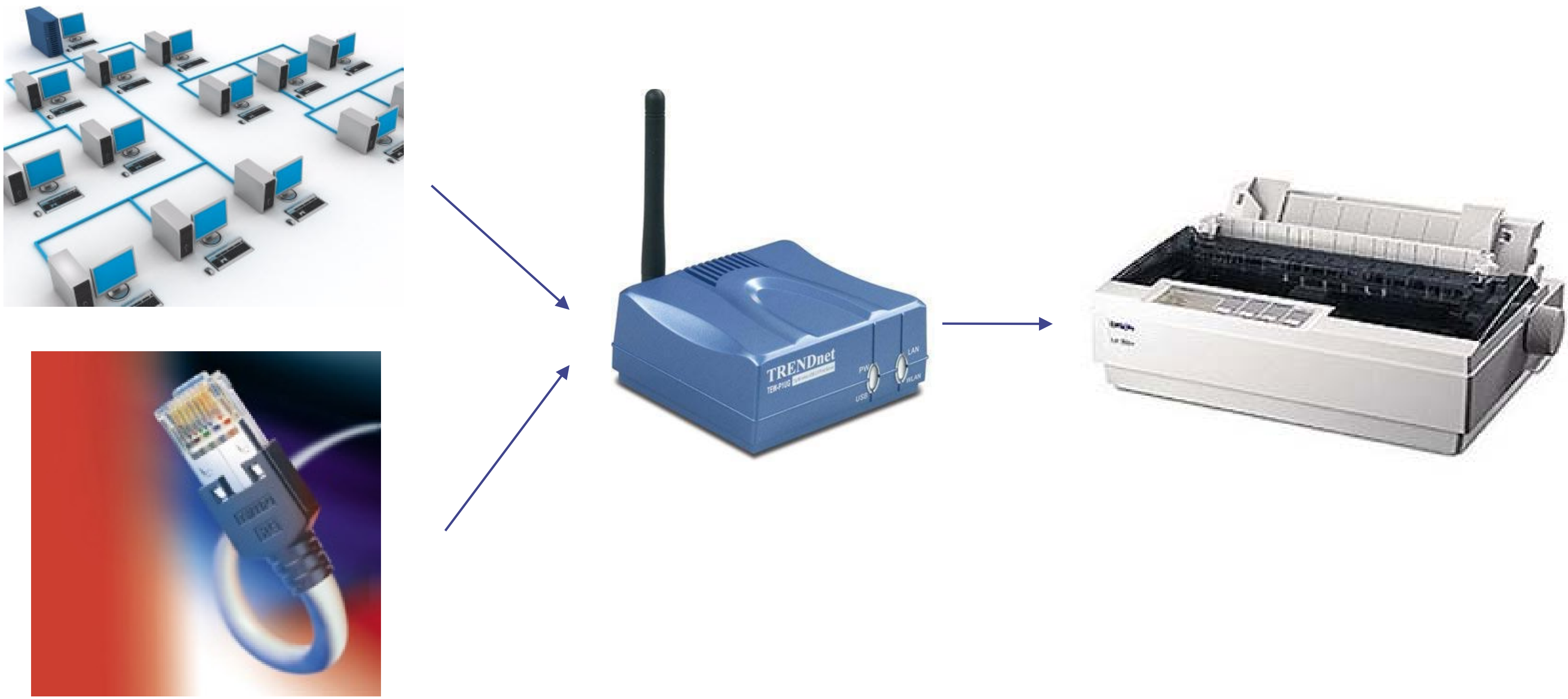


The book

- Chapter 1 – Introduction to embedded systems (today)
- Chapter 4 – Interrupts
- Chapter 5 – Survey of software architectures
- Chapter 6 – Introduction to RTOS
- Chapter 7 – More OS services
- Chapter 8 – Basic design with RTOS
- Chapter 9 – Toolchain
- Chapter 10 – Debugging



ES Example – Telegraph

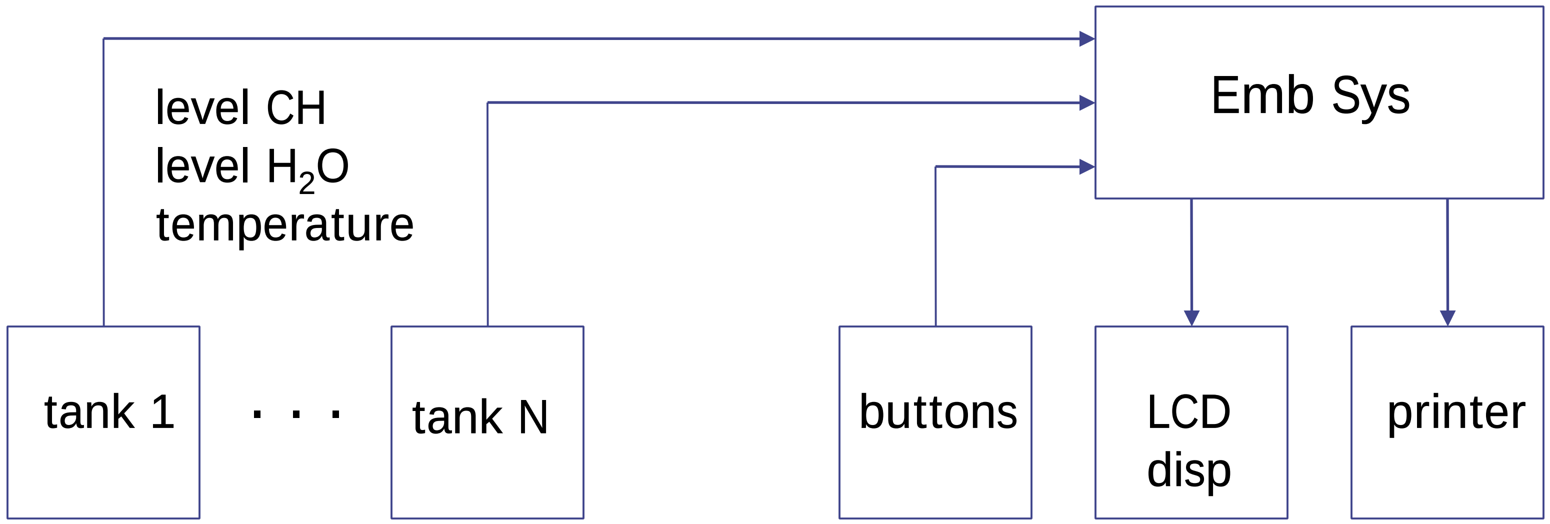


ES Example – Telegraph

- Out-of-order data
- Negotiate with multiple clients (print jobs) + status reqs.
- Adapt to different printers
- Response time to certain requests
- Data throughput / buffering
- Debugging and software updates

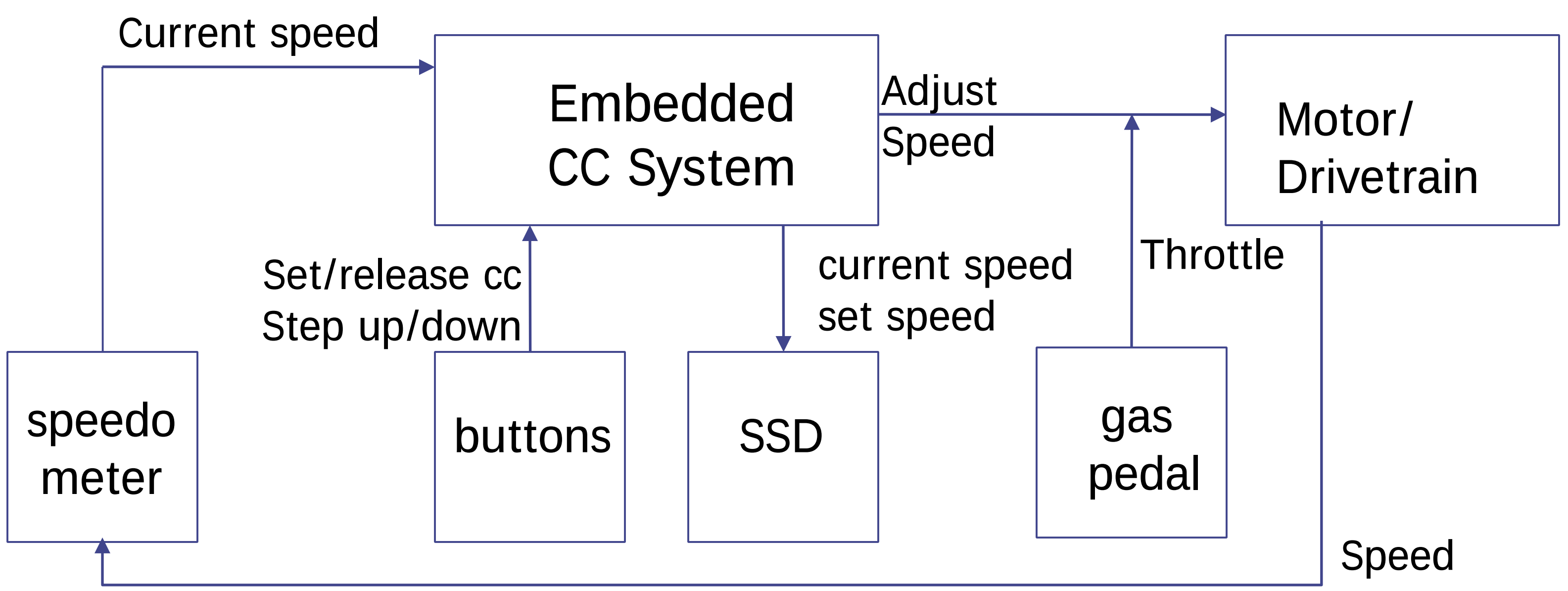
Telegraph is more complex than anticipated!

Underground Tank Monitoring Sys.



- Guard levels, detect leaks
- Extremely low-cost design (proc)
- Very simple arithmetic CPU - response time problem
- Model of normal drainage vs. leaking drainage

Cruise Control System



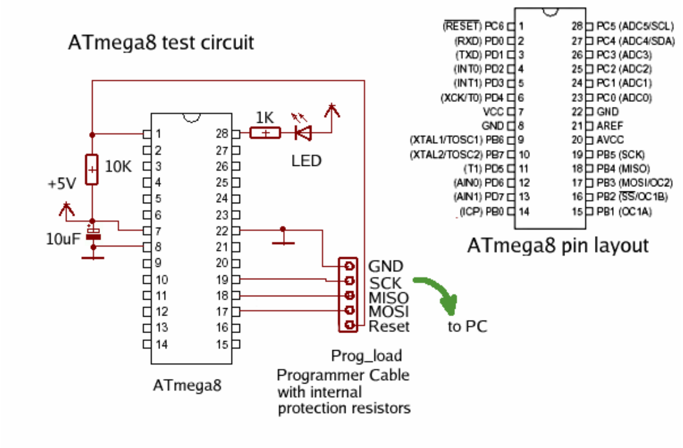
- Stabilize car speed when engaged
- Extremely low processor cycle budget
- Small control loop jitter due to other activities
- Reliable operation

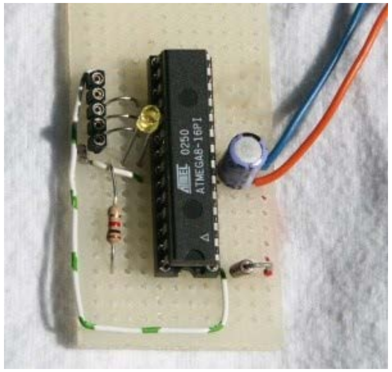
Characteristics of Embedded Sys.

- No / restricted user interface
- Specific connectors for sensors/actuators
- Restricted memory size and processing power
- Predictable timing behavior
- Suitable for extreme operation environments

Typical Platform for ES

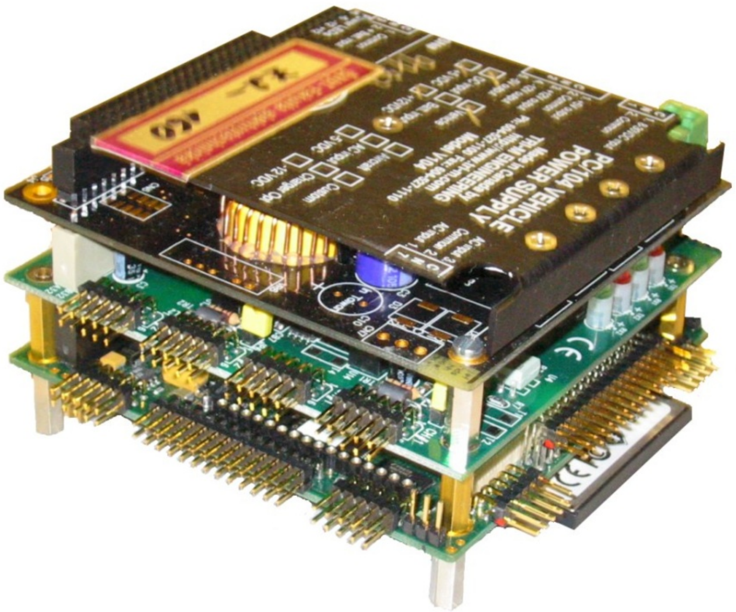
- Microcontroller
  - 8 bit RISC Processor
  - EEPROM & RAM
  - UART (serial line)
  - Timer
  - A/D converter
  - Digital I/O Lines



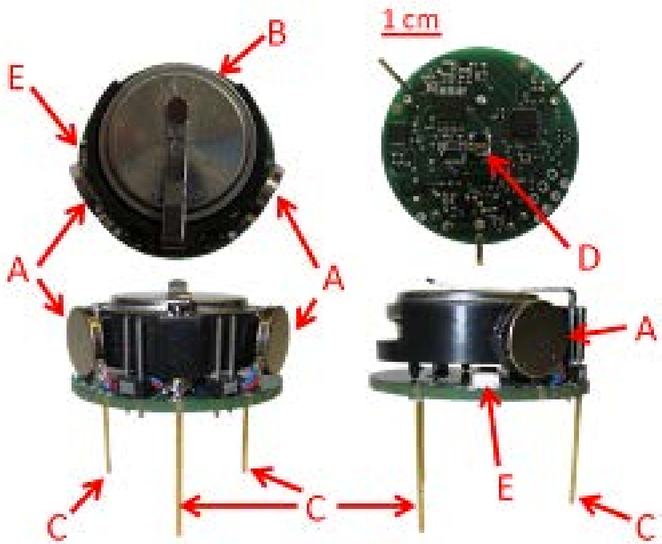


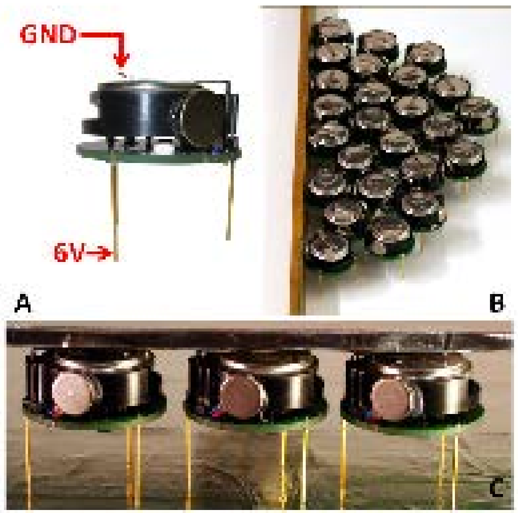
Typical Platform for ES

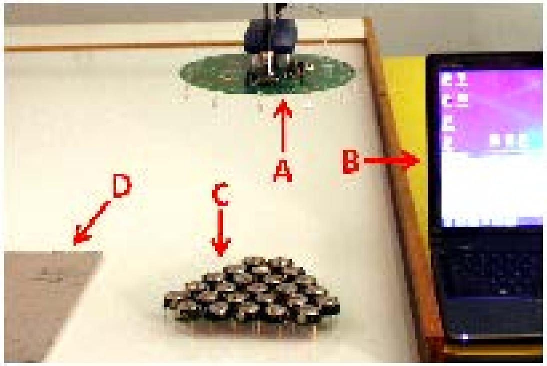
- PC/104
  - Typical PC platform
  - Flash, RAM, Drives
  - Many possible connectors and interfaces
  - Many available OSs



A different example - kilobot







Another Typical Platform for ES

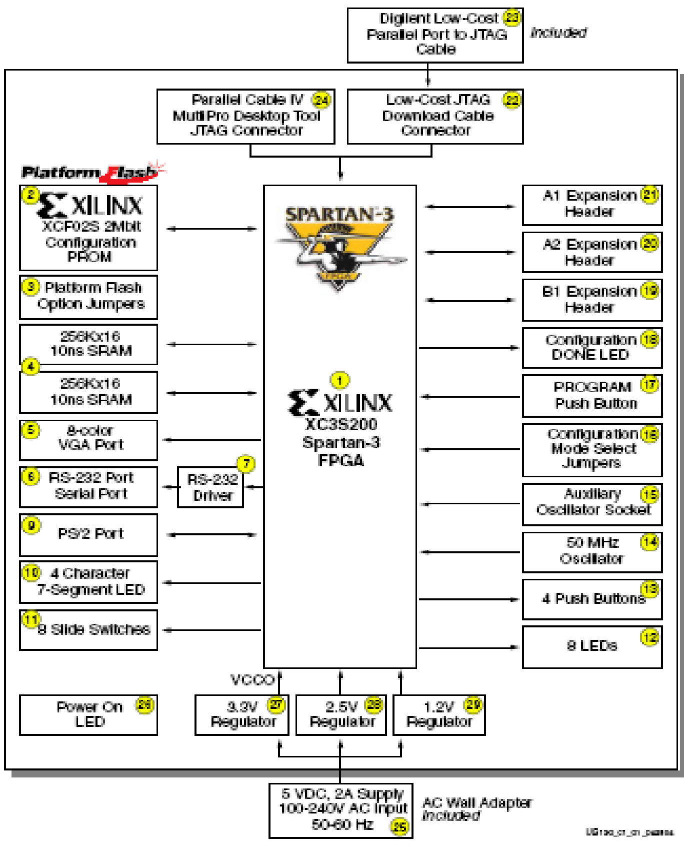
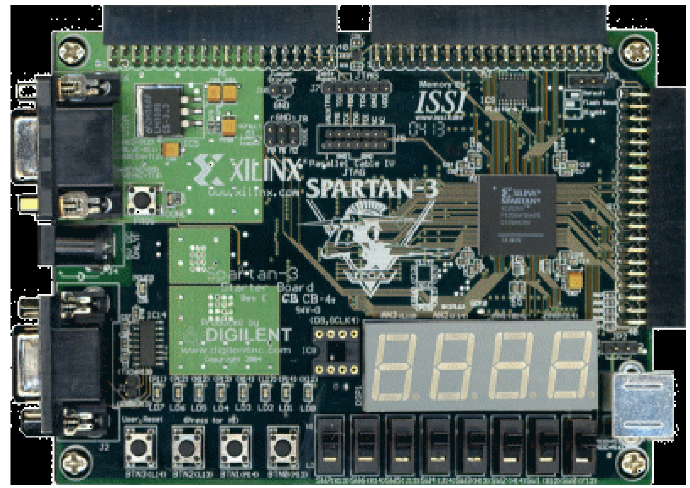


Figure 1-1: **Xilinx Spartan-3 Starter Kit Board Block Diagram**



- FPGA
  - Build your own hardware (I/O)
  - High performance
  - High-level programming

Embedded Systems Boom

- Provides functionality (intelligence) of almost everything
- Annual growth 25-60% (Emb Linux > 60%)
- 100 x PC market
- Accounts for 25-40% costs in automotive
- Very large societal dependence
- Very high performance demands
- More and more integration of systems

www.linuxdevices.com

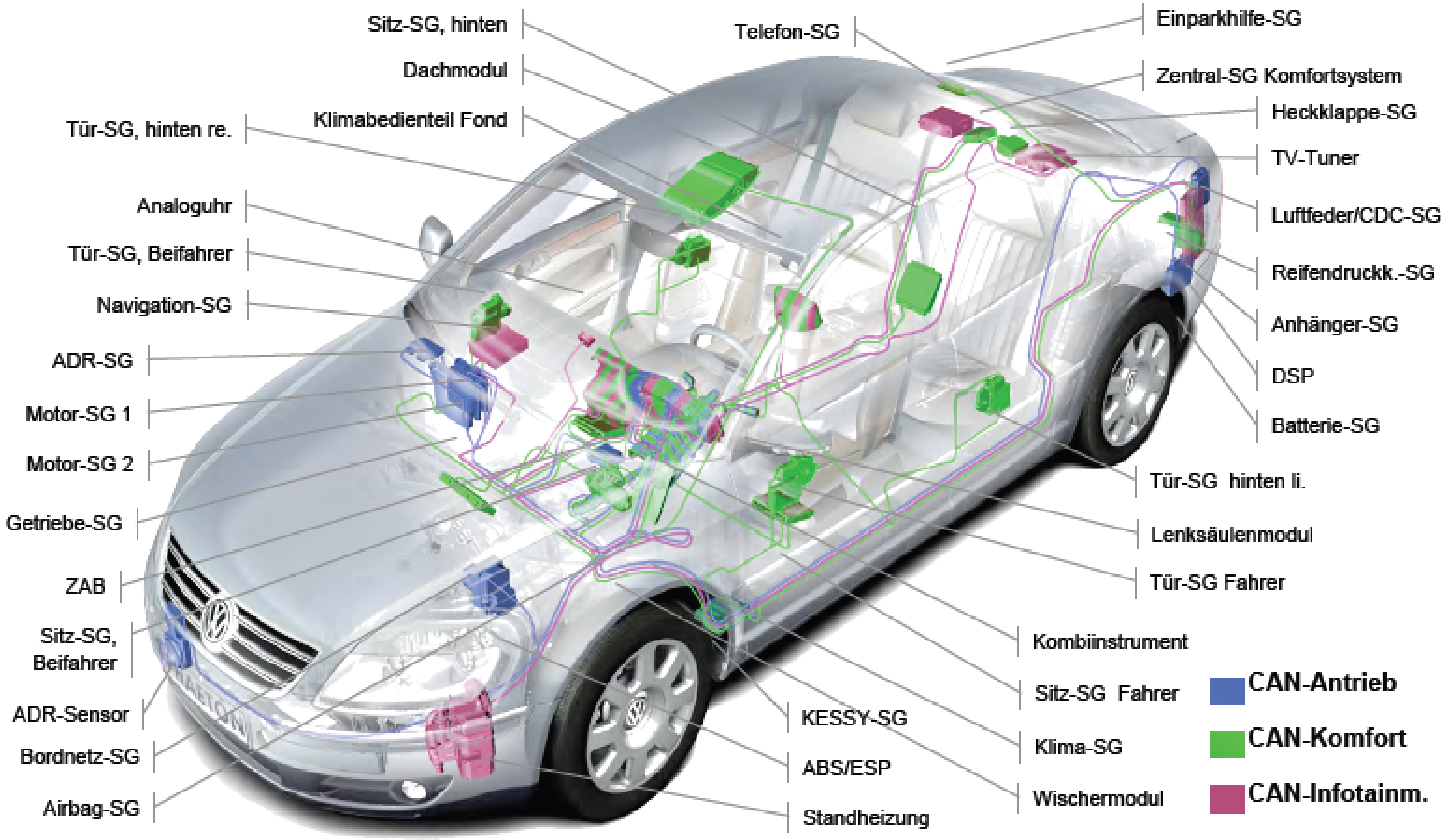

Embedded Software Boom

- Software
  - is more and more executed on standard hardware
- Accounts to a large extent for the
  - Product functionality
  - Intelligence / smartness
  - User ergonomics & look and feel
- Has an increasing added value
- Increased volume and complexity

50% Development Cost for Software alone!

90% of the Innovations Coming from Electronics & Software

CAN-Netw. Devices in a VW Phaeton



Embedded Software Crisis

- Functionality migrates from HW to SW
- Standard cores combined with FPGAs, rather than ASICs
- Programming-centred design (incl. HDLs)
- TV, mobile, car, .. 10+ MLOC code, exp. growth!
- Despite SW engineering: 1 – 10 bug / KLOC
- 100 Billion $ / yr on bugs (Mars Polar Lander, Mars Climate Orbiter, Ariane 5, Patriot, USS Yorktown, Therac-25, ...)

A new Embedded Software crisis?

Embedded Programming

- More difficult than “classical” programming
  - Interaction with hardware
  - Real-time issues (timing)
  - Concurrency (multiple threads, scheduling, deadlock)
  - Need to understand underlying RTOS principles
  - Event-driven programming (interrupts)
- Lots of (novice) errors (hence the crisis)
- That’s why we have this course already in 2nd year!

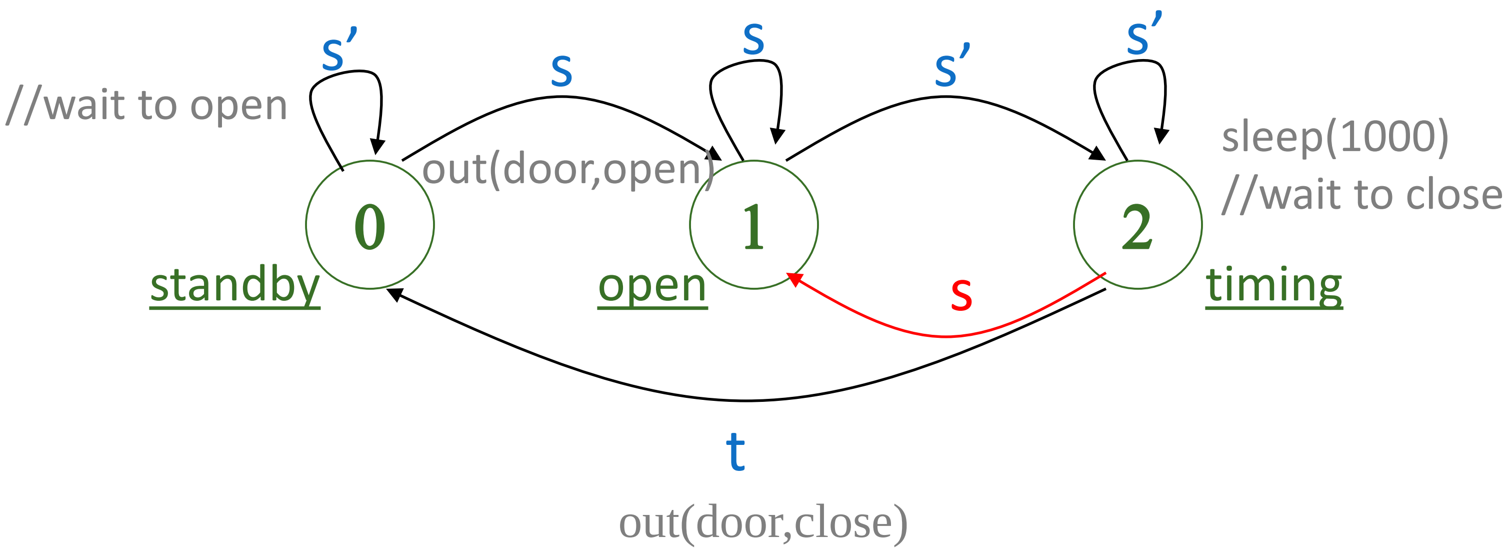
Embedded Programming Example

- Automatic sliding gate task (thread):

```
for (;;) {
  // wait to open
  while (inp(sensor) != 1) ;
  out(door,OPEN);
  // wait to close
  while (inp(sensor) == 1) ;
  sleep(1000);
  // close after timeout
  out(door,CLOSE);
}
```

- Any issues with this code?

Specification: Finite State Machine



- Red arc missing from the specification
- Door can slam in your face!

Door Controller in VHDL

- VHDL: FSM in entity door_controller
- Advantages
  - Separate hardware: no sharing of a processor (no scheduling, no priorities)
  - Fast and synchronous programming model: high frequency clocked process with simple polling for *s* and *t*
- Disadvantages
  - VHDL too cumbersome / prohibitive for large applications
  - Lots of legacy code written in C

A VHDL Solution

```
process -- fsm
begin
  wait until rising_edge(clk);
  case state is
    when S0 => if (s = ‘1’) then
                 state <= S1;
    when S1 => if (s = ‘0’) then
                 state <= S2;
    when S2 => if (s = ‘1’) then – red arc in FSM
                 state <= S1;
               if (t = ‘1’ and s = ‘0’) then
                 state <= S0;
  end case;
  door <= ‘1’ when (state != S0) else ‘0’;
  timer_enable <= ‘1’ when (state = S2) else ‘0’;
end process;
```

A C Implementation

- C: FSM in a task door_controller
- Advantages
  - simple (sequential) programming model
- Disadvantages
  - can't be invoked periodically by a high-frequency clock (timer) because of polling overhead
  - busy waiting (polling) is not an option (see above) -> concurrent (event) programming (e.g., using interrupts and semaphores)

- So the while loops in the example code are wrong
- Only use a delay that is not based on busy wait
- Ergo: interrupt programming, using an RTOS

A better (but not ideal) C Solution

```
void isr_sensor(void)                       // process sensor IRQ
{
  OS_Post(semaphore_event_on_s);            // signal s changed
}

void task_door_controller(void)
{
  for (;;) {
    OS_Pend(semaphore_event_on_s);          // wait for s = 1
    out(door,OPEN);
    do {
      OS_Pend(semaphore_event_on_s); // wait for s = 0
      OS_Delay(1000);
    } while (inp(sensor) != 0);             // timeout
    out(door,CLOSE);
  }
}
```

Issues

- Efficient, no busy waiting any more (OS_Pend, OS_Delay)
- Still, code is not correct: interrupts (entering/leaving persons within delay period are not properly handled, and are only accumulated in semaphore (wrong)
- Cannot afford to just “sit” in a delay, AND ...
- The ability to simultaneously wait for two events (s or t):

```
void isr_sensor_and_timer(void) { // handle both IRQs
  OS_Post(s_or_t);                // either s or t
}                                 // changed
```

Alternative C Solution

```
void task_door_controller(void) {
  for (;;) {
    switch (state) {
      STDBY:  OS_Pend(s_or_t);          // wait for 0-1
              out(door,OPEN);
              state = OPEN;
      OPEN:   OS_Pend(s_or_t);          // wait for 1-0
              timer_enable();
              state = TIMING;
      TIMING: OS_Pend(s_or_t);          // wait 0-1 || t
              if (inp(sensor) == 0) {  // timeout
                out(door,CLOSE);
                timer_disable();
                state = STDBY;
              } else state = OPEN;
}}}
```

Course Organization

- Grade = 0.5 exam + 0.5 lab
- Lectures (hall Boole): weeks 2.1 – 2.8
  - Tuesday, 15:45 – 17.45
  - Thursday, 15:45 – 17.45
- C programming: weeks 2.2 – 2.4
  - Wednesday, 13:45 – 17.45
- Robot Lab: weeks 2.5 – 2.9
  - Wednesday, 13:45 – 17.45

queue.tudelft.nl
weblab.tudelft.nl

pick up: Dec 8, 08:45 – 12:30

return & demo

Example exam questions

The “Embedded Software Crisis” refers to the “year 2000” bug.

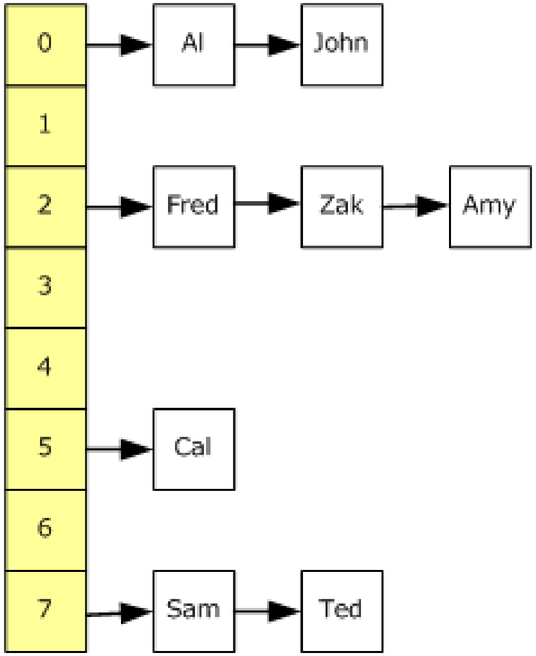
- true/false?

An embedded program can be coded as a finite state machine where interrupts trigger state transitions.

- true/false?

Lab: C programming

- Language
  - C-syntax, pointers, memory management, …
- Tools
  - Gdb, valgrind
- Assignment (**graded**)
  - Hash table with bucket lists



Lab: Robot
line follower

hidden agenda:
promote minor robotics

- Hardware [3mE]
  - Sensors: IR, ultrasonic ranging (2x)
  - Control: STM32F103C8T6
  - Actuators: motors (2x), LEDs

- Software
  - C
  - Arduino IDE
  - ROS – Robotic Operating System (**not!**)

Conclusion

- Embedded programming is not so easy
- Neither in C nor VHDL
  - Event programming needed: interrupts + RTOS support
  - Concurrency needed (seq. prog. model): RTOS support

- Learn the basics of interrupt programming & RTOS (in C)
- Learning is (lots of) programming!
- Lab: C (3 weeks) + Robot (5 weeks)

Sharing code is plagiarism
... and so is copying from the Internet / YouTube