Courses TMMA
Colleagues

- Sofie Beerens
- Johan Van Bauwel
- Bart Tanghe
- Wim Dams
- Lars Struyf
- Peter Arras
- Dirk Van Merode
Courses

- C for Embedded Systems
- DSP
- Embedded Communication
- Embedded Operating Systems
- Embedded Software
- Multicore Programming
- MCAD
- ECAD
C for embedded systems

Sofie Beerens
Prerequisites

- Beginners course: first semester of first year
- A basic knowledge of common mathematical methods
- No programming knowledge is required
Objectives

• Introduce basic programming principles:
  ○ division of a problem into smaller sub problems
Objectives

- Introduce basic programming principles:
  - convert (sub) problems into algorithms before coding

Think before coding

Flowcharts
Objectives

• Understand and use C syntax:
  o Predict the outcome of programs written in C syntax
  o Create a well structured program in C code containing functions
  o Choice of appropriate datatypes
  o Use file handling in C
  o Perform bit operations
Why C?

- C is a flexible and well-structured language
- designed to:
  - provide low-level access to memory
  - provide language constructs that map efficiently to machine instruction
- available on a very wide range of platforms, from embedded microcontrollers to supercomputers.
Course material

• Textbook: C for Embedded Systems
  o For each chapter/subject:
    – Objectives
    – Theoretical explanation
    – Code examples
    – Pitfalls, do’s and don’ts
    – Exercises
  lectures

  practicum / individual work

• Visual Studio Express 2013 for Desktop
Content

• Basic description of programming languages

• Dynamic data structures like lists
Content

1. Programming languages
2. Program design
3. Programming in C: an introduction
4. Basic concepts of C programming
5. Controlling the program flow
6. Functions
7. Arrays
8. Strings
9. Multidimensional arrays
10. Sorting and searching arrays
Content

11. Pointers
12. Comma operator, const, typedef, enumerations and bit operations
13. The C preprocessor
14. File handling in C
15. Structures
16. Command line arguments
17. Dynamic memory allocation
18. Dynamic data structures
Hello world

- Create a new visual studio project
- Write C code
- Compile the code
- Run the executable
- Verify the output
Create new project

[Image of Microsoft Visual Studio Express 2013 for Windows Desktop interface, highlighting the 'New Project...' option with the keyboard shortcut Ctrl+Shift+N.]
New Project Window

Choose project name and project dir.
Application Wizard

- Press ‘next’
Application Wizard

- Select ‘console application’
- Deselect ‘Precompiled header’
- Deselect Select SDL
- Select ‘Empty project’
- Press ‘Finish’
Solution explorer

Solution 'HelloWorld' (1 project)

- External Dependencies
- Header Files
- Resource Files
- Source Files

Search Solution Explorer (Ctrl+§)
Create new source file

- Right click on “Source Files”, followed by “Add -> New Item”
- Select “Code” and enter a file name

!! Default extension is “.cpp” (for a C++ file).
Make sure you save your file as a “*.c” file
Create new source file
Create new source file

Add New Item - HelloWorld

- C++ File (.cpp)
- Header File (.h)

Name: HelloWorld.c
Location: C:\Users\u0088734\Desktop\Temp\HelloWorld\HelloWorld\n
Click here to go online and find templates.

Type: Visual C++
Creates a file containing C++ source code

Add
Writing Hello World program
Hello World!

/*
 * HelloWorld.c
 * Our first C program
 */

#include <stdio.h>

int main(void)
{
    printf("Hello world\n");
    return 0;
}

comments
preprocessor directive
main
statement
statement
Hello World: create exe file

```
1>------ Build started: Project: HelloWorld, Configuration: Debug Win32 ------
1>   HelloWorld.c
1>   HelloWorld.vcxproj -> C:\Users\u0088734\Desktop\Temp\HelloWorld\Debug\HelloWorld.exe

1>------ Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ------
```
Hello World: run program

- cmd window opens shortly and closes again after execution

=> add breakpoint
Hello World: run program

Hello world
Other options to keep console

- Use “Start Without Debugging”
- Add a line “getchar();” before “return 0;”
Learning objectives

- Students are able to design DSP algorithms using C and/or a higher level language
- Students are able to test the algorithms in a sensible manner
- Students can calculate the impulse response of a LTI-system
- Students can calculate a convolution sum and a DFT
- Students can analyze frequency content of digital signals using the DFT/FFT
- Students are able to create a transfer function and are able to draw a pole-zero plot using the Z-transform
- Students possess knowledge of the various topics treated in this course
- **To make students enthousiast about DSP!**
Teaching methodology

- Prerequisites: C programming, simulation software skills, Linux basics
- Theory: lectures + exercises
- Labs: lab assignments

- Theory: 50%, Labs: 50%
Teaching methodology

- Lab assignments: decreasing support, increasing independency
Teaching methodology

- Lectures: 12 weeks, 2hrs/week
  => 24 contact hours
- Labs: 12 weeks, 3hrs/week
  => 36 contact hours
- Students will have to prepare the lab assignments, study the theory, ...
  labs: practical test
Learning tools

- Course text
- Lab assignments, library API & UDOO reference manual
- Various on-line resources
- Supplementary reading material: several outstanding books (cfr. next slide)
Learning tools

- Oppenheim & Schafer: “Discrete Time Signal Processing”
- Lyons: “Understanding Digital Signal Processing”
- Smith: “The Scientist & Engineer’s Guide to Digital Signal Processing”
- Proakis: “Digital Signal Processing”
- Orfanidis: “Introduction to Signal Processing”
- Analog Devices (Walt Kester): “Data Conversion Handbook”
Course text

- Signals and systems
- Sampling
- Convolution
- DFT
- FFT
- FIR filters
- IIR filters
- Filter Design
- Filter Structures
- Z-transform
- DSP software & hardware
- Multirate DSP
Lab assignments

- Signals and systems, sampling
- Convolution (1D and 2D)
- DFT (incl. windowing and zeropadding)
- FFT
- Digital filters: FIR and IIR
- Filter structures (DF, DF2, cascade)
- The Goertzel algorithm (standard and optimized)
- Basic image processing (color inversion, RGB to grayscale)
- Median image filtering
- Edge detection (using Laplacian, Sobel, Prewitt, ...)
- Histogram equalization
- Audio filters on .WAV files
Demo
Embedded OS

Bart Tanghe
Scratch on Raspberry Pi

- Presentation & demo
Embedded SW

Wim Dams
Embedded Software Course

Bare metal C on ARM Cortex M4
Objectives of the Embedded Software Course

- Develop an understanding of the technologies behind an embedded system
  - Software components: RTOS, HAL Drivers, Libs
  - Hardware Modules: USB, Ethernet,
  - Interaction between software and hardware
  - Build system, compiler settings, performance
ARM Cortex M core

- ARM Cortex-M processors have been licensed to over 175 ARM partners (vendors) and benefits from the widest third-party tools, RTOS and middleware support of any architecture. Which makes it the best choice for embedded applications.
The evaluation board: STM32F4DISCOVERY

- Coded in "Bare metal" C
- Embedded ST-LINK/V2 (USB->JTAG)
- LEDs, PushButton
- USB (Host, Device, OTG)
- Motion Sensor
- MEMS audio sensor
The controller: STM32F407VGT6

- ARM™ Cortex-M4 core (168MHz/210DMIPS)
- Single Cycle DSP MAC & FPU
- USB, Ethernet MAC, DMA, 6xUSART, 2xCAN, 3xI²C, 3xI²S, 3xSPI, SDIO/MMC
- 1 Mbytes Flash
- 192 Kbytes SRAM
Optional: Base Board (STM32F4DIS-BB)

- Interfaces for:
  - Serial Port (RS232 levels)
  - Ethernet (Phy)
  - MicroSD
  - TFT LCD (sold separately)
  - Camera (sold separately)
Software environment

- CooCox CoIDE
  - Free
  - Based on Eclipse but better GUI (less options)
  - GCC toolchain
  - Integrated debugger
  - No Simulator (not needed)
STM32CubeMX

- Firmware for STM32 microcontrollers
- Generates boilerplate code based on wizard
- CoIDE is not supported. Manual import is needed

STM32Cube Drivers

Manuals and Datasheets

STM32Cube Middleware

IDE Specific Projects
Configured Drivers and Middleware
Pin Configuration Report
Hardware Initialisation Code
Power Consumption Estimates
Course Overview

• Labs (Hands On) 2,5h x 12 = 30h
  o Blinky (GPIO)
  o Hello World (UART, Semihosting, USB CDC)
  o Timer (Interrupt)
  o DMA
  o Embedded TCP/IP lwIP
  o RTOS (FreeRTOS)
Course Overview

- Project during remaining labs (+homework)
  - Student chooses their own project (functional).
  - Student needs to implement a middleware library (e.g. lwIP or FatFS or FreeRTOS).
  - Teacher guards complexity, cost, ...
Course Overview

- Results of a Project (on a previous MCU)

Ethernet to DMX

Audio Player
Course Overview

- Lectures 30h
  - Development environment
  - GPIO
  - Semihosting
  - Uart Communication
  - USB Communication
  - Interrupts (Systick, GPIO)
  - lwIP

- DMA (Uart)
- RTOS
- Start-up/Boot code
- Linker scripts
- Coding Standards (CERT C, MISRA C)
- Coding Style
- ARM Cortex-M Core
Demonstration

- Http webserver based on Hands On 5 (lwIP)
  - LED3
  - ADC
Goals Of The Course

- Give students an understanding about multicore programming
- Give students some background on multicore hardware
- Learn students how to program multicore hardware with OpenCL
Goals Of The Course

- Theory: 12 weeks x 1h
- Labs: 8 weeks x 3h

- Material
  - PowerPoint presentations
  - Lab assignments

- Complementary reading material
  - OpenCL 1.2 specification
  - Heterogeneous Computing with OpenCL v2
  - Internet
Course Overview

- Introduction
  - Parallel Computing
  - OpenCL
- GPU Architectures
- OpenCL
  - Buffers
  - Images
  - Memory
  - Threading HW
  - Optimizations
  - Nbody Optimizations
  - Extensions
  - Timing
  - Debugging
  - Multidevice
Parallel Computing

- Parallel computing
- Multicore hardware
- Determining parallelism in software
OpenCL

- What is OpenCL?
- OpenCL platforms
- OpenCL devices
  - Contexts
  - Queues
- Data transfers
- OpenCL programs and kernels
- Threads
- Memory model
- Address space
Useful Information

• OpenCL Specification
• Books
  o Heterogeneous Computing with OpenCL
    – Second Edition revised for OpenCL 1.2
  o OpenCL Programming Guide
  o OpenCL In Action
Content

- Goals
- Course Overview
- Labs
  - Hardware
  - Software
  - Lab Setup
  - Lab Exercises
  - Lab Projects

Multicore Programming
Hardware

- NVidia
  - GPU
    - GeForce 8400GS and up
    - Quadro NVS 295 and up
    - Tesla
  - Tegra2 and up

- AMD/ATI
  - GPU
    - Radeon HD4000 series and up
    - FireStream 9250 and up
    - FirePro V3750 and up
  - CPU
    - AMD x86 with SSE 2 and up

- Intel
  - CPU
    - Intel x86 with SSE 4.1 and up
Hardware

- **ARM**
  - GPU
    - ARM Mali 600 and up
  - CPU
    - Cortex-A7, A9, A15, A17, A53, A57

- **IBM**
  - Processor
    - Cell/B.E.

- **Altera**
  - FPGA
    - Cyclone V
    - Stratix V
    - Arria V
Software

- AMD
  - AMD Accelerated Parallel Programming (APP) SDK
  - CodeXL

- NVidia
  - CUDA

- Intel
  - Integrated Native Developer Experience
Start Up: Create Project

Multicore Programming
Start Up: Create Project
Start Up: Create Project

![Image of Visual Studio creating a new project]
Start Up: Add Library Files
Start Up: Add Library Files

Multicore Programming
Start Up: Add Library Files

![Image of a configuration settings window for OpenCL start properties]

Additional Include Directories

- Specifies one or more directories to add to the include path; separate with semi-colons if more than one.

Additional #using Directories
- <Edit...>

Other configuration options include:
- Suppress Startup Banner
- Warning Level
- Treat Warnings As Errors
- SDL checks
- Multi-processor Compilation

Configuration: Active(Debug)
Platform: Active(Win32)
Configuration Manager...

Multicore Programming
Start Up: Add Library Files
Start Up: Add Library Files

Multicore Programming
Start Up: Add Library Files

Additional Dependencies
- `opengl.lib`
Code Profiling

- CodeXL integrates in Visual Studio and can be used from there
Code Profiling

- The console window shows some extra output.
Code Profiling

Multicore Programming
Code Profiling

Multicore Programming
Questions?
Contact

Ing. Dirk Van Merode MSc.
Project Coordinator DESIRE
Thomas More | Campus De Nayer
Technology & Design
J. P. De Nayerlaan 5
2860 Sint-Katelijne-Waver

Belgium
Tel. + 32 15 31 69 44
Gsm + 32 496 26 84 15
dirk.vanmerode@thomasmore.be
Skype dirkvanmerode
www.thomasmore.be

Dr. Ing. Peter Arras MSc.
International Relations Officer
KU Leuven | Campus De Nayer
Faculty of engineering technology
J. P. De Nayerlaan 5
2860 Sint-Katelijne-Waver

Belgium
Tel. + 32 15 31 69 44
Gsm + 32 486 52 81 96
peter.arras@kuleuven.be
Skype pfjlarra
www.iiw.kuleuven.be