Programming Project

- **<u>Objective</u>: Achieve practical skills within imperative programming & microprocessor system**

- **<u>Elements</u> of the course include:**

  - microcontroller architecture
  - registers in a microprocessor
  - structural programming in C
  - hardware implementation in a μP
  - documentation
  - fixed point format
  - analysis of program
  - real time application
  - software workflow

Programming Project

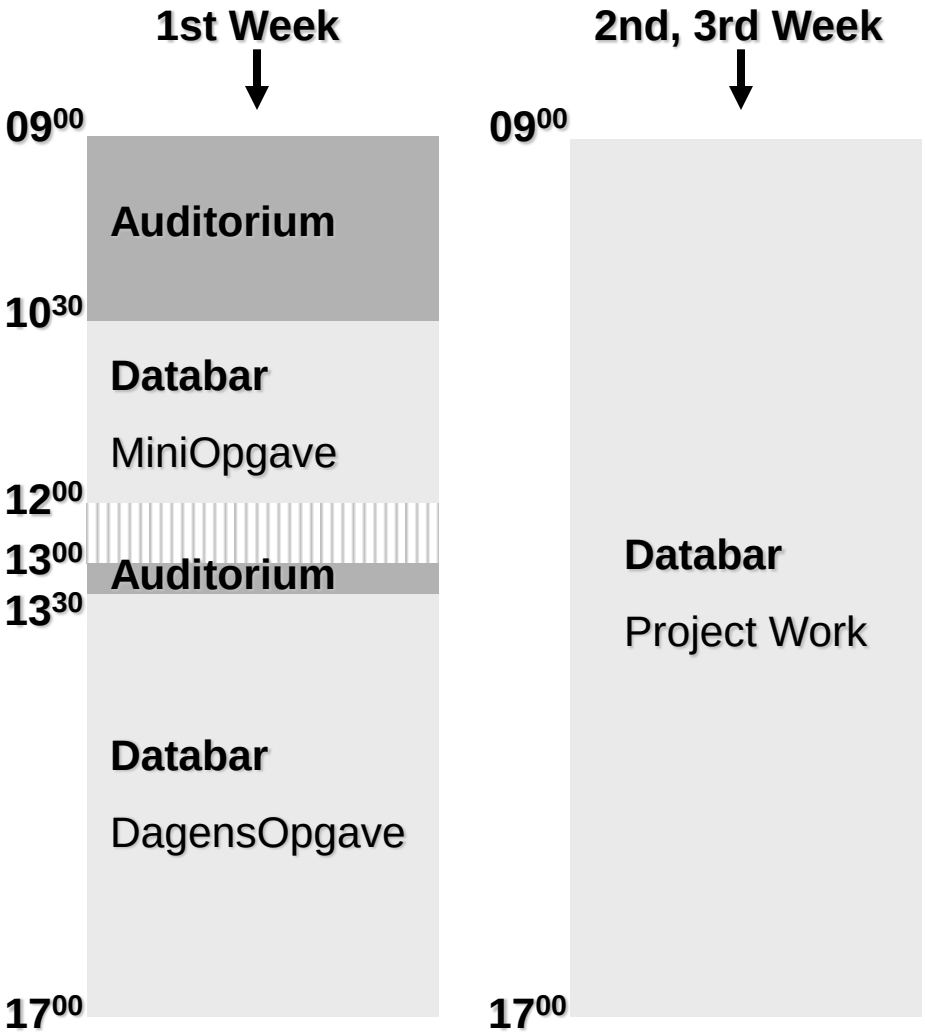
<u>At the end</u> of the course you should be able to:

- Identify and list the basic elements of the architecture of a microcontroller
- Describe and explain basic mathematical operations in fixed point format
- Apply and demonstrate the use of registers in a microprocessor
- Analyse a medium-size programming problem
- Apply C language and structural programming in C
- Implement and synthesize a program targeting a microprocessor
- Design and realize an application in real time
- Organize, plan and document the workflow of a software project
- Write a technical report including references and citations

Course Organization

- 1st week (Introduction):
  - Applied C Language
  - Program Structuring
  - Microprocessor Architecture
  - UART, Timer and I/Os
  - Introduction to Project Work
- 2nd week (Project Work):
  - Analysis
  - Block Diagram
  - Program Specifications
  - Coding/Debugging
- 3rd week: (Project Work)
  - Coding/Debugging (cont.)
  - Documentation
  - Report

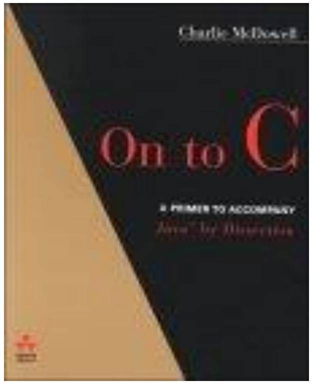
3 persons per group, which loan a HW kit
Auditorium: 341/22. Databar: 341/015-9
Do not leave your HW unattended!



30010 Course Plan

1st week – Introduction, ver 1. Preliminary!					
	Thursday (31.5.)	Friday (1.5.)	Monday (4.6.)	Wednesday (6.6.)	Thursday (7.6.)
Theme background 09:00-10:20 (341/23)	**Course Introduction C, Structural, HW & Compiler**	**C (cont.), Structural C, Array, Pointer, Structure**	**Microprocessor Architecture (1): Intro., I/O, Register**	**Microprocessor Architecture (2): Display, UART**	**Introduction to the Project Work**
Exemplarize 10:30-12:00(341/015-9) (JMGM + LC, MB, JL, 10:15-12:15)	**Exercise 1**: Learn/Use Compiler: Edit, Compile, Debug, Download & Execute program targeting µP.	**Exercise 3**: Application of fixed point arithmetic. Using arrays & structures.	**Exercise 5**: Learn how to use the µP registers to read/write to the I/0. (Ref:Gpio,Irq,Timer).	**Exercise 7 (+8)**: Use the LCDs to show text and scroll, using Timer & IRQ	
Assignment 13:00-17:00(341/015-9) (JMGM + LC, MB, JL, 13:15-16:15)	**Exercise 2 Work**: Draw window in putty with ANSI-Escape codes. Stucturing with multifiles: .h, .c	**Exercise 4 Work**: Mini project: ball in movement within a window	**Exercise 6 Work**: Application of the onboard Timer with IRQ for implemen-ting a stop watch.	**Exercise 8 (cont.)** Use the onboard ADC	**Project Work**: Discussion of Project Work at group level
2nd week – Project Work					
	Friday (8.6.)	Monday (11.6.)	Tuesday (12.6.)	Wednesday (13.6.)	Thursday (14.6.)
09:00-12:00 (1.5x??) 13:00-17:00 (1.5x??)	**Milestone: Preliminary Review**				
3rd week – Project Work					
	Friday (15.6.)	Monday (18.6.)	Tuesday (19.6.)	Wednesday (20.6.)	Thursday (21.6.)
09:00-17:00		**Milestone: Status Review**			**Written Exam 2 h., d. 21/6 (TBC) Report Delivery & HW Delivery**

Literature

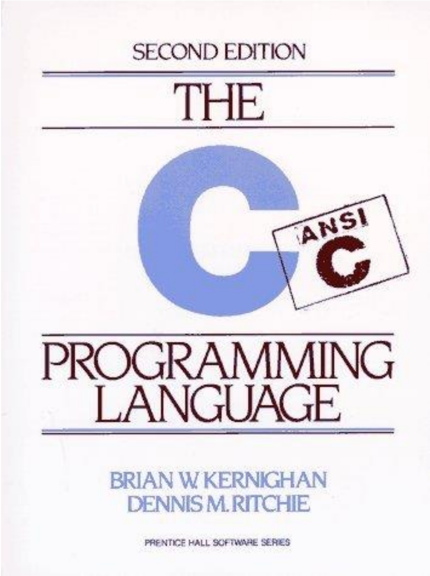


C Language:

- Java to C: A Primer, C. McDowell & Jørgen Villadsen, Polyteknisk Forlag, 2013
- On to C, C. McDowell, 1st ed., Addison Wesley, 2001
- ANSI C for programmers on UNIX systems, T. Love, Cambridge Univ., 1996 (*)
- Programming Language, 2nd ed, Kernighan and Ritchie, Prentice Hall, 1988

Hardware:

- User Manual for the STM32 NUCLEO F302R8 development board (*)
- Programming Manual for STM32F/L3-4+ Series Cortex®-M4 (*)
- Reference manual STM32F302 advanced ARM®-based 32-bit MCUs (*)
- Datasheet for the STM32F302x8 ARM Cortex MCU (*)
- MBED extension board (*)



Web:

- Campus Net

Recommended Courses:

- Equivalent to ”Indledende programmering” (02102)

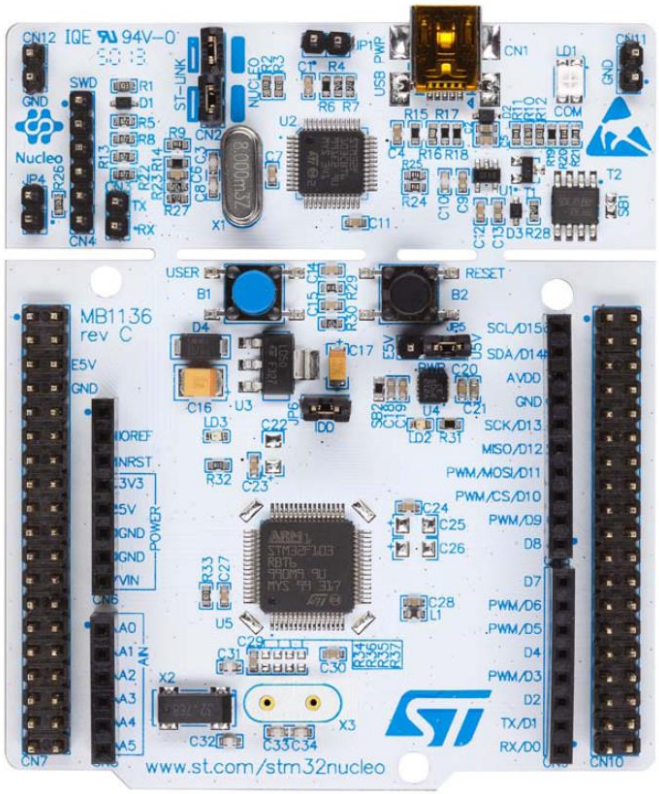
(*) .pdf file downloadable from Campusnet

” *K&R is, so far as one can tells, the best book ever write*

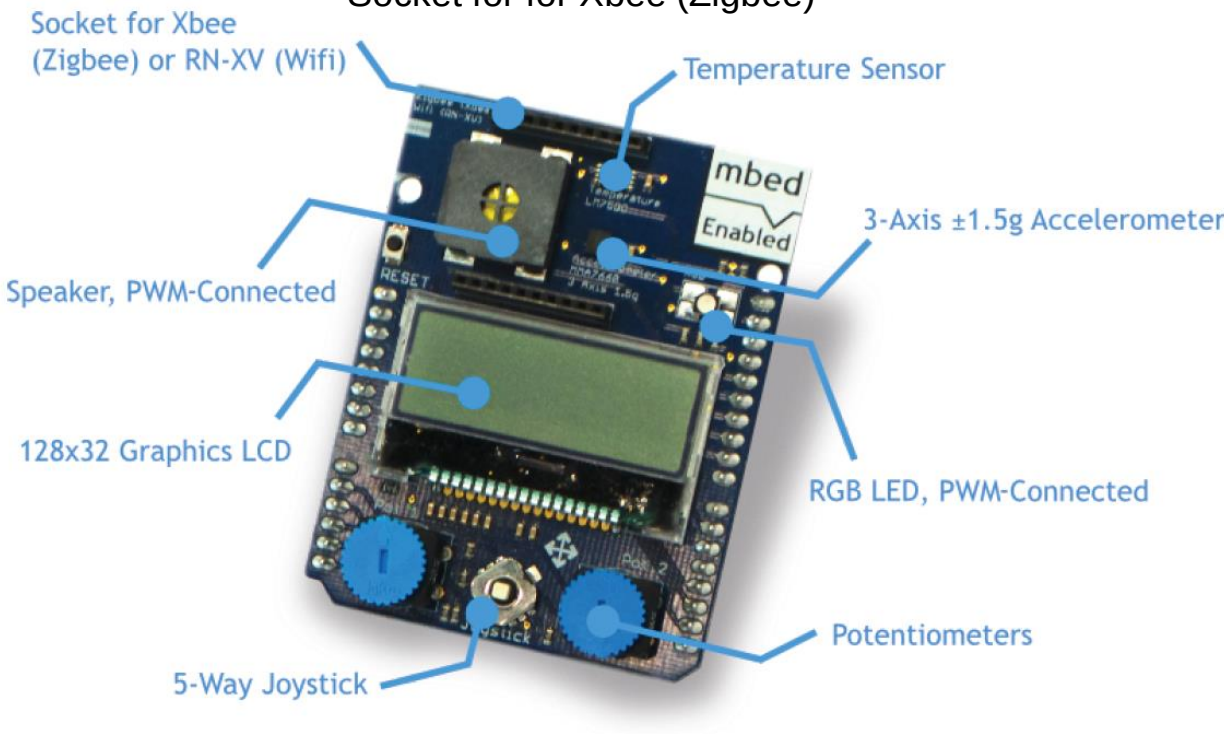
Hardware Kit

USB Port

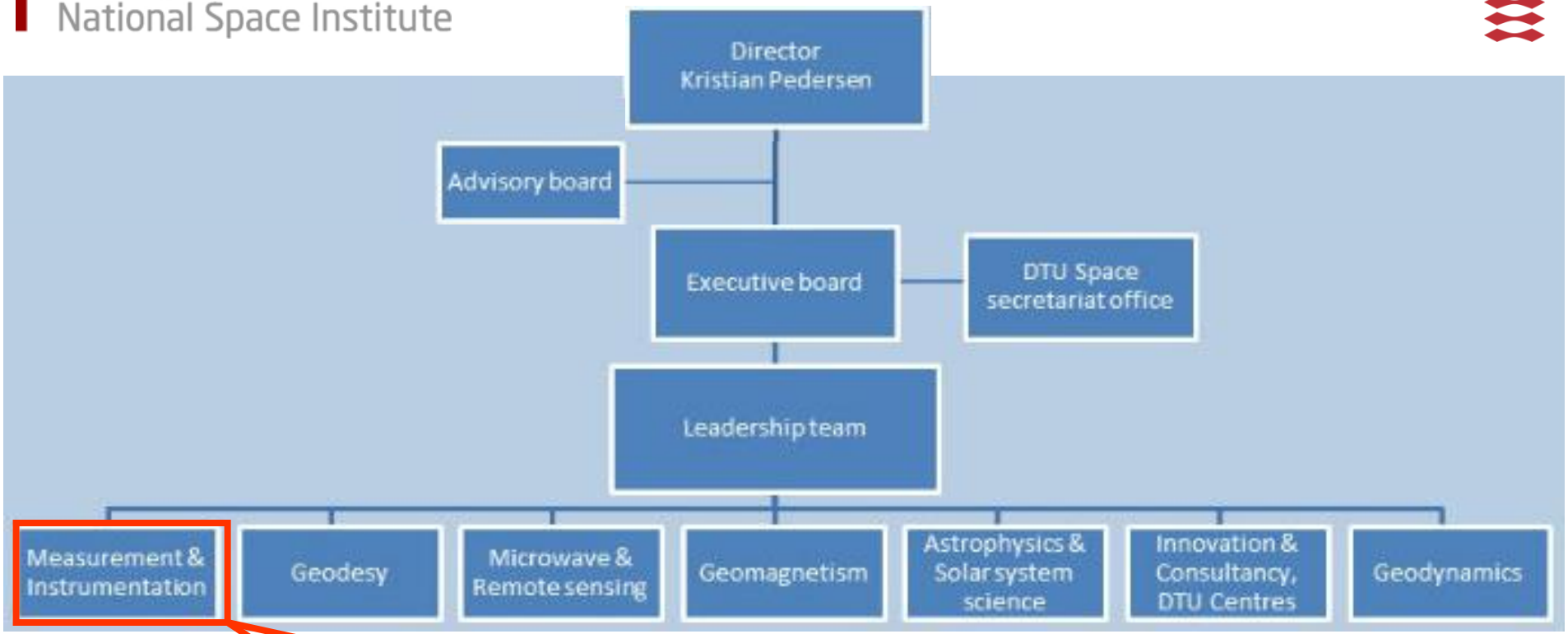
NUCLEO-F302R8 board
(ARM Cortex STM32F302R8 MCU)



MBED extension board
128x32 Graphics LCD
5 way joystick
2 x Potentiometers
Speaker (PWM Connected)
3 Axis +/1 1.5g Accelerometer
RGB LED (PWM connected)
Temperature sensor
Socket for for Xbee (Zigbee)



DTU Space
National Space Institute



Jose M.G. Merayo
Professor
Course Responsible
jmgm@space.dtu.dk
tlf. 4525 3452

Lukas Alexander Mads Christensen
Exercise Coordinator
lamc@space.dtu.dk

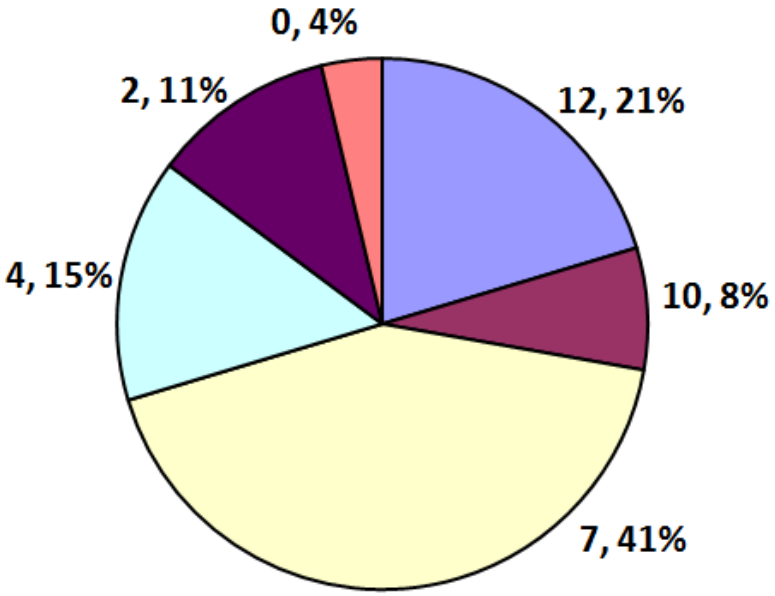
Johannes Linde
Course Assistant
jlinde@space.dtu.dk

Mathias Bredholt
Course Assistant
s144040@student.dtu.dk

Mads Frimann Madsen
Course Assistant
mfma@dtu.dk

Course Evaluation

- **Delivery of Course Project Report, including *Journal* of the exercises in the appendix and Written exam:**
  - **1 report per group (weight 2/3)**:
    ~1/3 per student, each student specifies what he/she did.
  - **Individual multipole choice exam (weight 1/3)**:
    last day of course, 2 hours and based on project work
  - **Karakter**: 12-skala, Ekstern censur
- **Evaluation is based on the different elements included in the course:**
  - **SW Design, Development & Implementation**
    - **Layers (10%)**
    - ***.h/*.c (10%)**
    - **Structure (10%)**
    - **Extras (10%)**
  - **Software Documentation**
    - **Req. Specs. (10%)**
    - **Flow Charts (10%)**
  - **Report**
    - **Modules (10%)**
    - **Structure (10%)**
    - **Manual (10%)**
  - **Code Execution (10%)**



Low/High Level Language

- **Assembly Language (Low Level Language)**
  + Programs that manipulate a lot of Hardware
  + Initializing peripherial devices
  + Programs that have to execute fast
  - Large programs (slow, tedious)

- **C (High Level Language)**
  + Larger programs
  + Very widely used in the industry (see Ingeniøren)
  + Good stepping stone to “higher level” languages (fx. object oriented)
  + Very easy to learn if you know assembly programming language
  + Foundation for UNIX/LINUX, C++, Java
  + Programming of electronic based equipment

- Why use more than one programming language?

Programming Languages and C

- **Some programming languages**

(B)	**C**	C++
SmallTalk	Assembler	BASIC/COMAL
APL	JAVA	PASCAL
LISP	COBOL	FORTRAN

- **C Characteristics**

  - General purpose
  - Low level (“Run on the metal”)
  - UNIX, compilers
  - Simple
  - Portable (ANSI C)
  - High performance (Compiler)

Programming Process

THINK

Break design into modules
(assembler/C)

EDIT

Syntax High lighting

MAKE

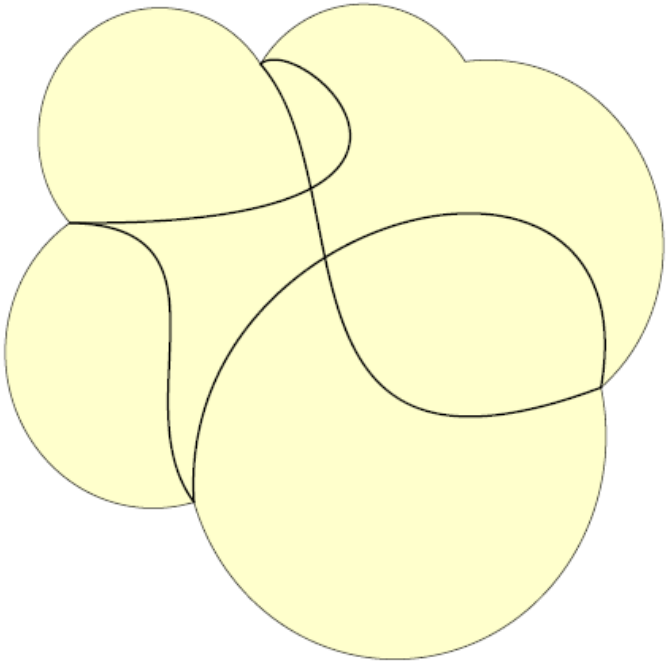
1) Preprocess
2) Compile
Assembler and C modules into .OBJ files
3) Link .OBJ files to .EXE file

TEST

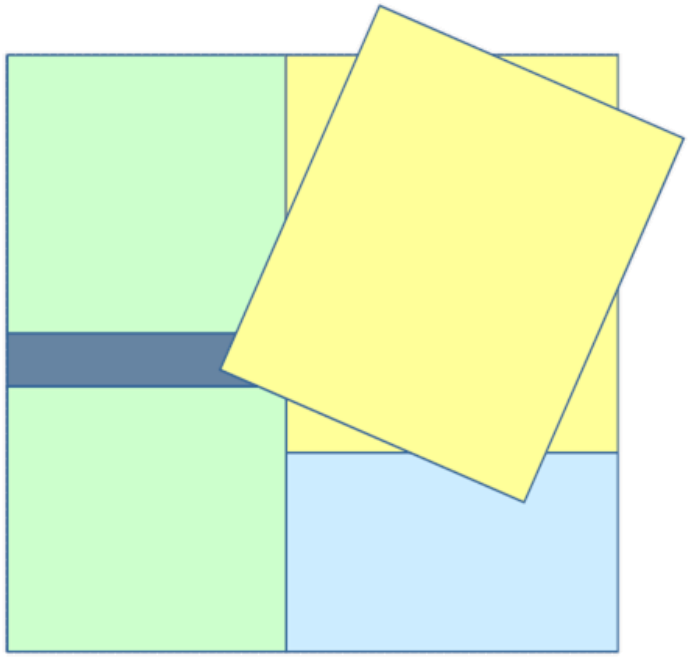
Debug

Programming Architecture Abstraction

Software System



Structured Software System



Requirements → Software Architecture → Low Level Design → Implementation → Test

Program structure in C

calc.h:

```
#define NUMBER '0'
void push(double);
double pop(void);
int getop(char []);
int getch(void);
void ungetch(int);
```

main.c:

```
#include <stdio.h>
#include <stdlib.h>
#include "calc.h"
#define MAXOP 100
main() {
    ...
}
```

getop.c:

```
#include <stdio.h>
#include <ctype.h>
#include "calc.h"
getop() {
    ...
}
```

stack.c:

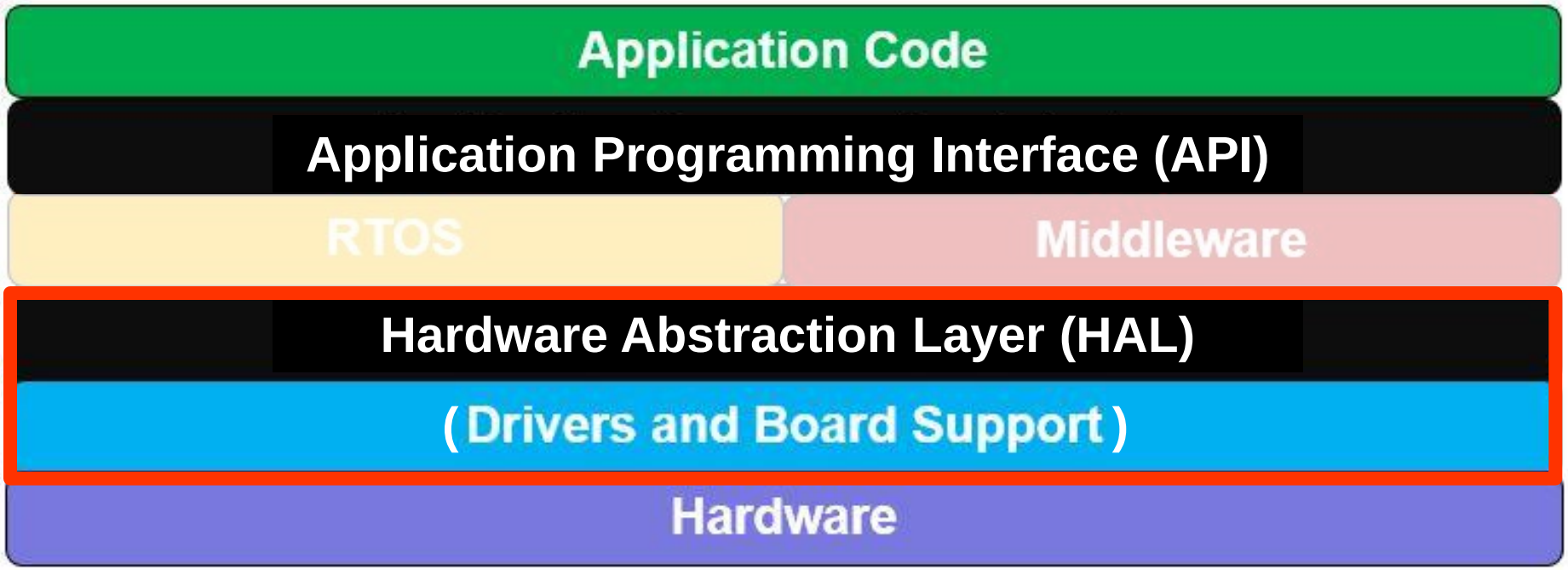
```
#include <stdio.h>
#include "calc.h"
#define MAXVAL 100
int sp = 0;

void push(double) {
    ...
}
double pop(void) {
    ...
}
```

getch.c:

```
#include <stdio.h>
#define BUFSIZE 100
#include "calc.h"
int bufp = 0;
int getch(void) {
    ...
}
void ungetch(int) {
    ...
}
```

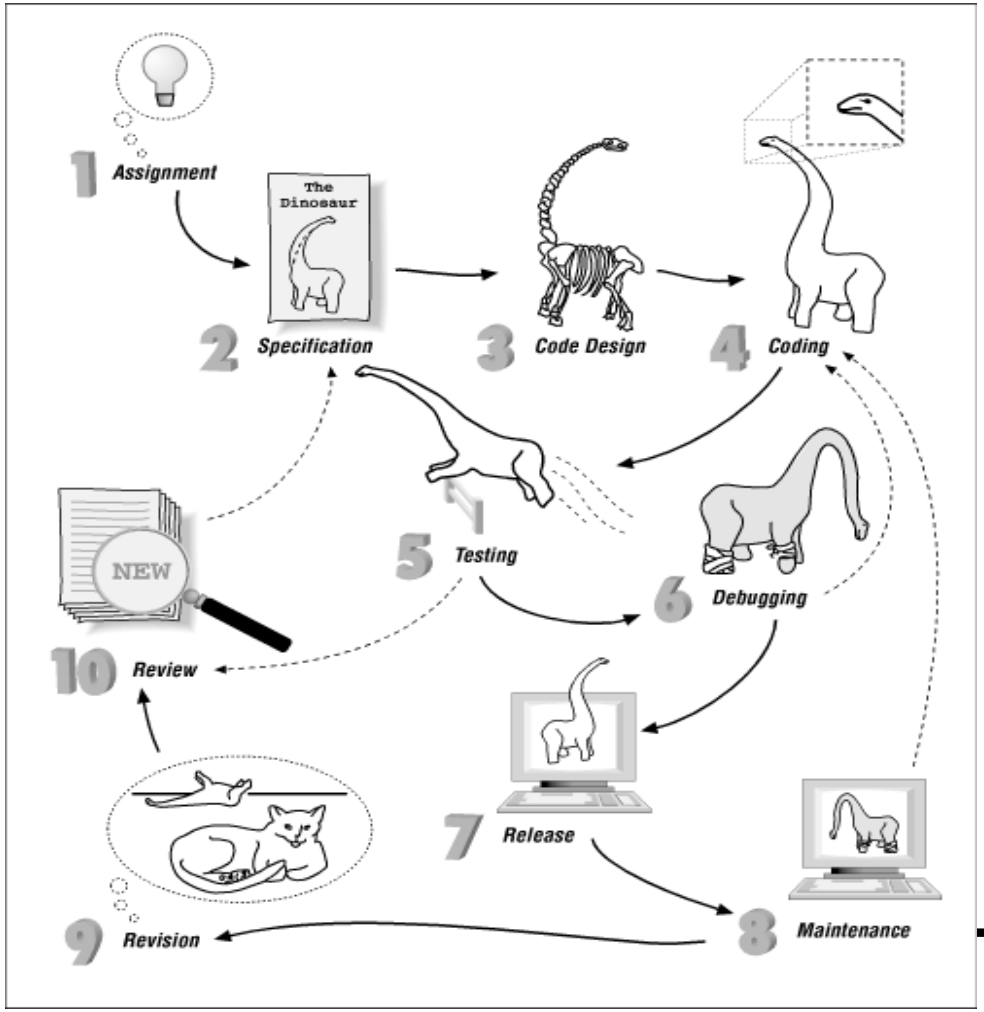
`#include <sio.h>` : Compiler standard library
`#include "egen.h"`: project own library

Programming Architecture Abstraction Model



Source: https://www.beningo.com/embedded-basics-apis-vs-hals/

Software life cycle



- Software …

  is born, …

  grows up, …

  becomes mature, …

  and finally dies, ….

  only to be replaced by a newer, younger product

Basic structure of a C program

```
#include "30010_io.h"
#include <sio.h>

#define ESC '\x1B' //or #define ESC 27

int funk1()
   {
    ...
   }
void proc2()
   {
    ...
   }
char funk3()
   {
    ...
   }
void main()
   {
    char tal=25;
    ...
    funk3();
    ...
    funk1();
    ...
    proc2();
   }
```

Libraries

Parameters

type returned by the call

Functions & Procedures

main

due to µP we are dealing with. In other OS, it can be `int main()`

variable declaration & initialization

"hello world" version in a µP

```
#include "stm32f30x_conf.h"
#include "30010_io.h"

void main ()
   {
    init_usb_uart( 9600 ); // Init USB
    printf ("hello world\n");
    while(1){}
   }
```

Note: C is case sensitive!

Note: It is FORBIDDEN to use *Global variables* in this Course

Data Types in C

type	undertype	size (bits)	range	
char				
	unsigned char	8	0 ->	255
	char	8	-128 ->	127
int (short)				
	unsigned int	16	0 ->	65536
	int	16	-32767 ->	32767
	unsigned long	32	0 ->	4294967296
	long	32	-2147483648 ->	2147483647
float				
	float	32	|3.4e-38| ->	|3.4e+38|
	double	64	|1.7e-308| ->	|1.7e+308|
	long double	80	|3.4e-4932| ->	|3.4e+4932|
pointer				
	near	16	64K	
	far or huge	32	4G	

Escape sequences

\ followed by 1+ characters

\a	alert (bell) character
\b	backspace
\f	formfeed
\n	newline
\r	carriage return
\t	horizontal tab
\v	vertical tab
\\	backslash
\?	question mark
\’	single quote
\”	double quote
\ooo	octal number
\xhh	hexadecimal number
\0	null character, used as string terminator

Declaration & Initialization

```
#define VTAB '\xb'  // ASCII vertical tab (hex) or
#define VTAB '\013' // ASCII vertical tab (decimal)

#define BELL '\x7'  // ASCII bell tab (hex) or
#define BELL '\007' // ASCII bell tab (decimal)

#define FALSE 0
#define TRUE 1

char key;           // extern variables
char slut=FALSE;

void main ()
   {
    char ch;        // local variables
    char flag_a, flag_b;
    char tal=25;
    char tegn='A';  // ASCII tegn
    char bell='\x07'; // escape sequence
    char return='\r';

    char buffer[20]; // Array
    char format[13]="\n\nResultat: ";
    char message[]="Fejl";

    ... ... ... ...
   }
```

C operators

Arithmetic

Addition	+	a=c+d
Subtraction	-	a=c-d
Multiplication	*	a=c*d
Division	/	a=c/d
Modulus	%	a=c%d
Sign minus	-	a=-d
Sign plus	+	a=+d

Bitwise

AND	&	a=c&d
OR	|	a=c|d
XOR	^	a=c^d
NOT	~	a=~d
SHIFT LEFT	<<	a=a<<n
SHIFT RIGHT	>>	a=a>>n

(n is number of bits to be shifted)

Type Converting / Type Casting

```
void main()
   {
    char c1, c2='A';
    int i1, i2=7913;
    float f1, f2=7.913;

    int i=2, s=3, resultat;
    long i=10;
    float f=3.33;

    // In assignments

    i1=c2;   //i1='A'=0x45=65
    c1=i1;   //c1='A';
    c1=i2;   //7913=0x1ee9->c1=0xe9=-23
    f1=i2;   //f1=7913.0
    i2=f2;   //i2=7
    c2=-100; //-100=0x9c
    i2=c2;   //i2=-100 or i2=0x09c=156

    // In expressions

    resultat=f*i+l/s;
       // a) g=f*i=3.33*2.0=6.66 (i->float, g-> type float)
       // b) m=l/s=10/3    =3    (s->long,  m-> type long)
       // c) h=g+m=6.66+3.0=9.66 (s->float, m-> type float)
       // resultat=9 (type int)
   }
```

C operators (cont.)

Relational

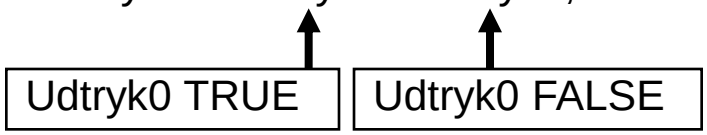
Equal to	==
Not equal to	!=
Larger than	>
Less than	<
Larger than or equal to	>=
Less than or equal to	<=

Boolean

Logical AND	&&
Logical OR	| |
Logical NOT	!

Conditional Expression

Udtryk0 ? Udtryk1 : Udtryk2;



Assignment

a += b	is identical to	a = a + b
a -= b	-"-	a = a - b
a *= b	-"-	a = a * b
a /= b	-"-	a = a / b
a %= b	-"-	a = a % b
a &= b	-"-	a = a & b
a |= b	-"-	a = a | b
a ^= b	-"-	a = a ^ b
a <<= b	-"-	a = a << b
a >>= b	-"-	a = a >> b

Increment & Decrement

++n	is identical to	n = n + 1
--n	-"-	n = n - 1

++n is incremented before used
n++ is incremented after used

`printf()`: formatted output to a device

It has the following **arguments**:

- **Control String**
- **Variable(s)**

```
#include "30010_io.h"
#include <sio.h>

void main()
   {
    long l=987654321L;
    float f=12.3456789;
    double d=-0.0000334499;
    char t[]="abcdefghij";

    init_uart...

    printf("\nUdskrift med printf\n");
    printf("\nl = :%-20ld:",l);
    printf("\nf = :%20.5f:",f);
    printf("\nd = :%20.10e:",d);
    printf("\ns = :%20s:",t);
   }
-------------------------------------
Udskrift med printf
l = :987654321           :
f = :             12.34567:
d = :   -3.3449900000e-05:
s = :          abcdefghij:
```

Format Specifier Symbol	Print
%d (%3d, %03d)	decimal integer
%u	unsigned integer
%ld	long decimal integer
%p	pointer value
%f (%6.2f)	floating point format
	floating point round off to two digits of decimal point, total of 6 digits
%e	exponential format floating point
%c	ASCII character for value
%s	string
%x or %X	hex value of integer
-	left justified
\n \f ‘ ’ \t \r	newline, formfeed, space, htab, return

Control Flow

Statements

```
i=a+b/j;
```

Block Statements

```
{
 i=0;
 j=5*i+2;
}
```

If-else Statements

```
if (udtryk) statement;

if (udtryk) statement1;
else statement2;

if (udtryk1)
     if (udtryk2) statement1;
     else statement2;

if (udtryk1) statement1;
else if  (udtryk2) statement2;
else if  (udtryk3) statement3;
else statement4;
```

Switch

```
switch (udtryk)
   {
    case konstantudtryk1: statement1; break;
    case konstantudtryk2: statement2; break;
    default: statement3; break;
   }
```

While

```
while (udtryk) statement;
```

Do while loop

```
do statement;
while (udtryk);
```

For loop

```
for (udtryk1a, udtryk1b; udtryk1b; udtryk3) statement;

udtryk1;
while (udtryk2)
   {
    statement;
    udtryk3;
    }
```

Comma operator (in for): `udtryk1 : j=0, i=5`

Exercise 1

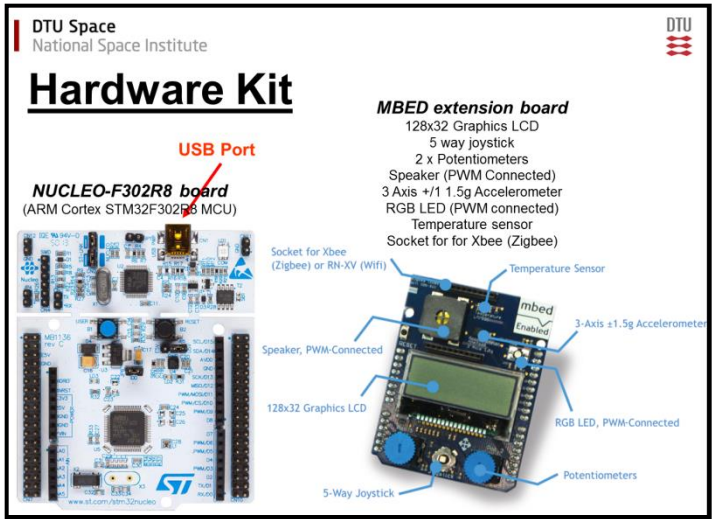
Installing the Hardware

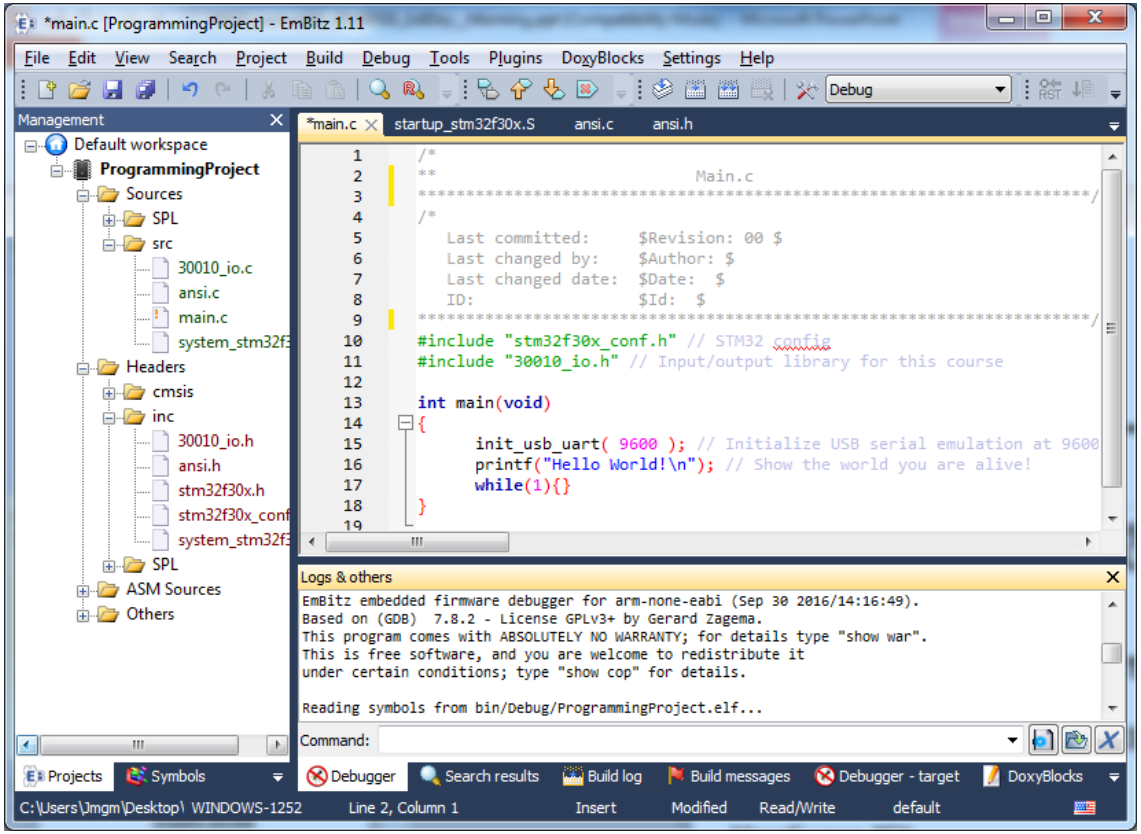
Familiarizing with Compiler

Starting with ARM STM32

Uploading Code to ARM STM32

Debugging with ARM STM32





Compiler support only with Windows PC !!

Programming Project

- **Exercise 2:**
  - **ANSI codes to control Putty**
  - **From CN: (30010\Exercises\2)**
  - **Draw a ”window” in an Putty window**
  - **Build the ANSI C functions library -> `ansi.c`**

- `fgcolor(int fg)`: **is used to change the foreground color**
- `bgcolor(int bg)`: **is used to change the background color**
- `color(int fg, int bg)`: **is used to change the foreground & background color with one function call, which minimizes bandwidth on the serial channel**