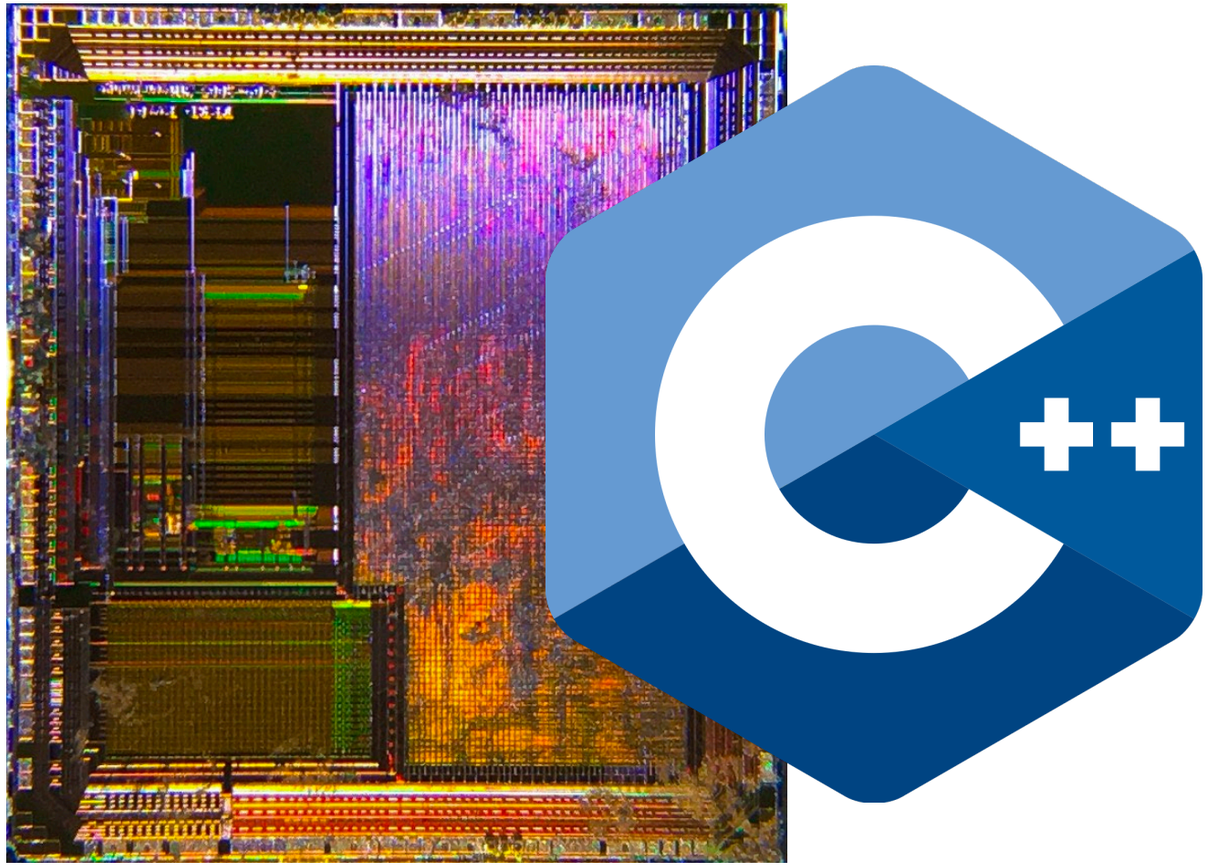
Embedded Real-Time Software

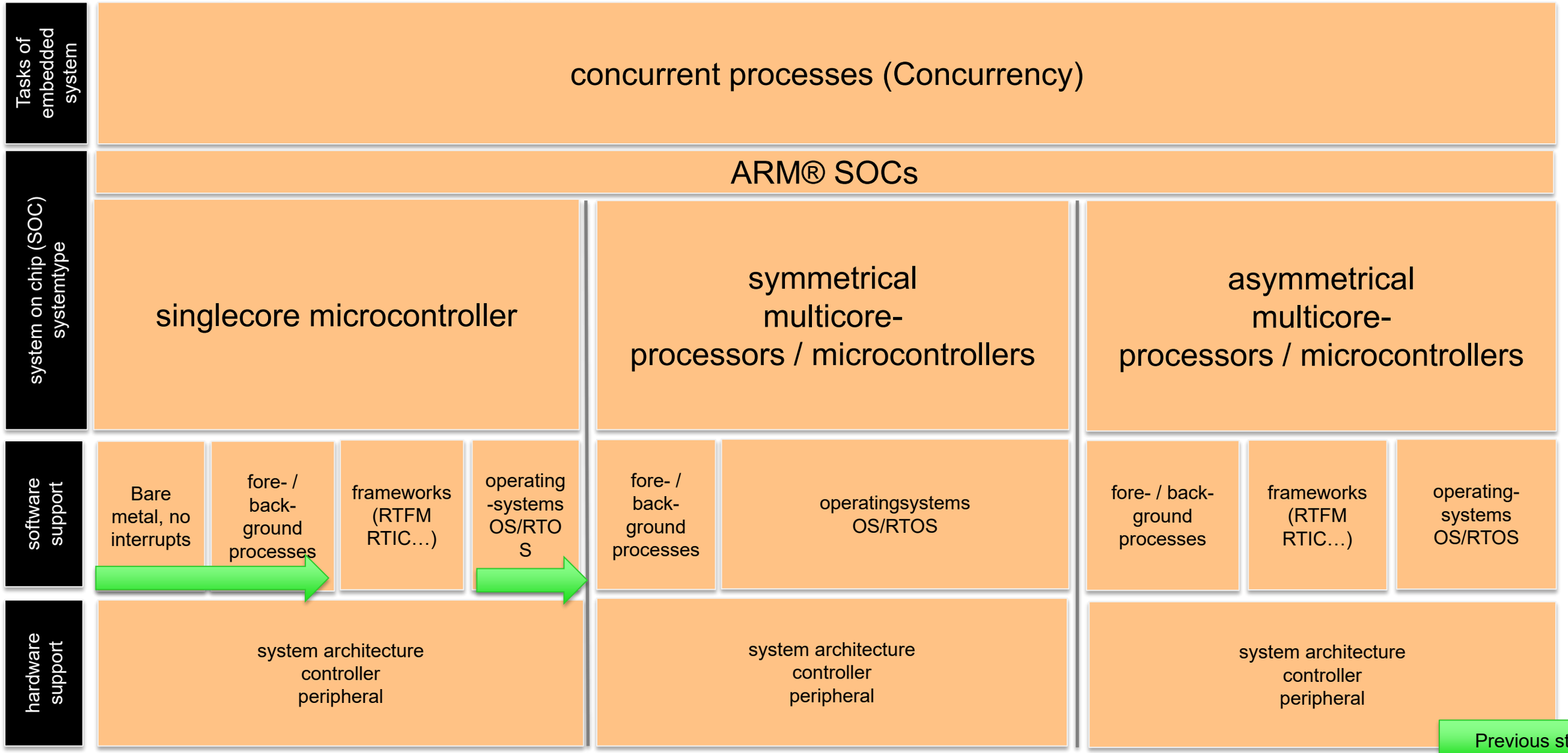
Introduction to object oriented programming on Microcontroller



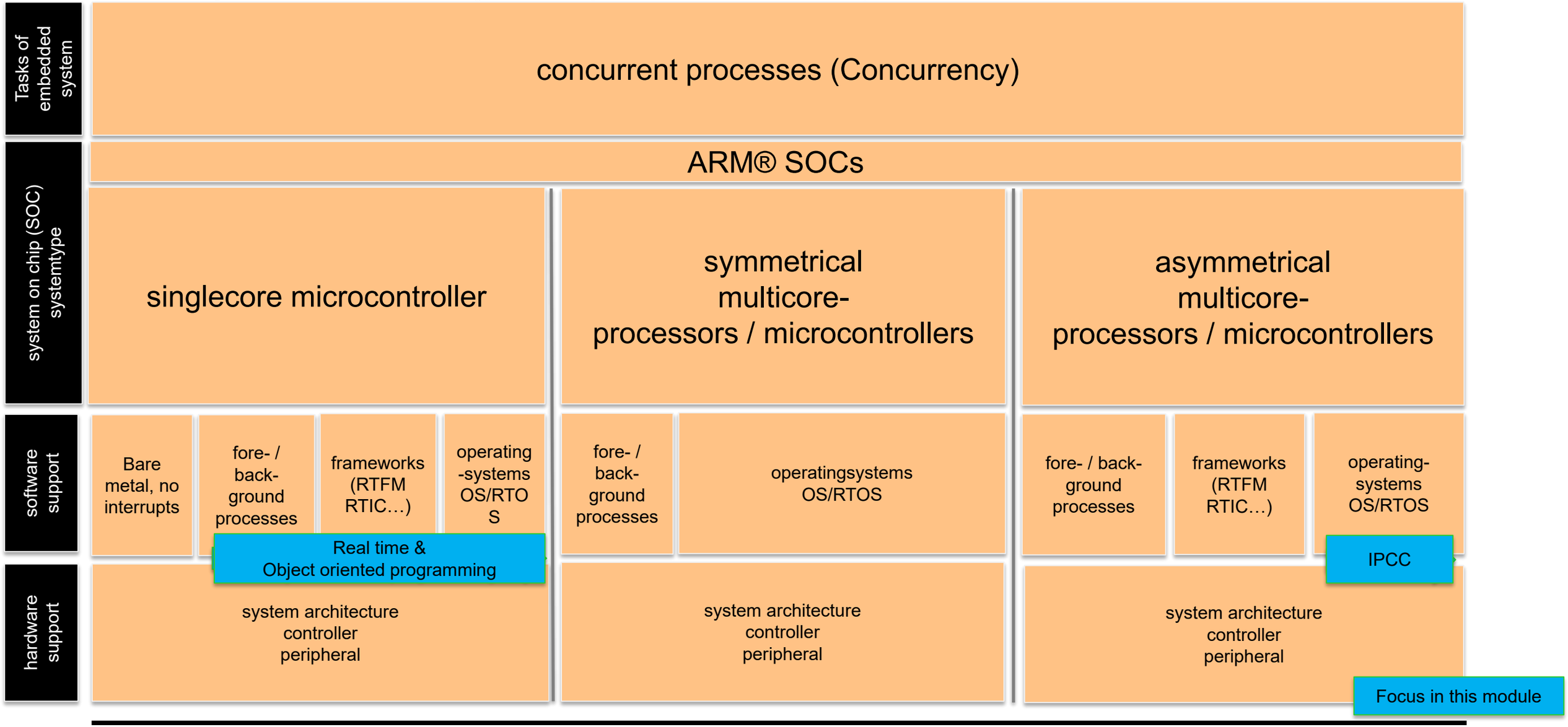
19th September 2023 , Prof. Dominique-Stephan Kunz

Overview on topics, approach and CPU technology for Part 1 and Part 3

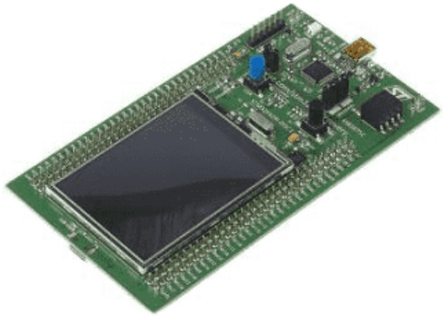
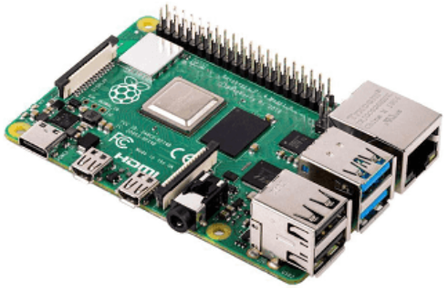
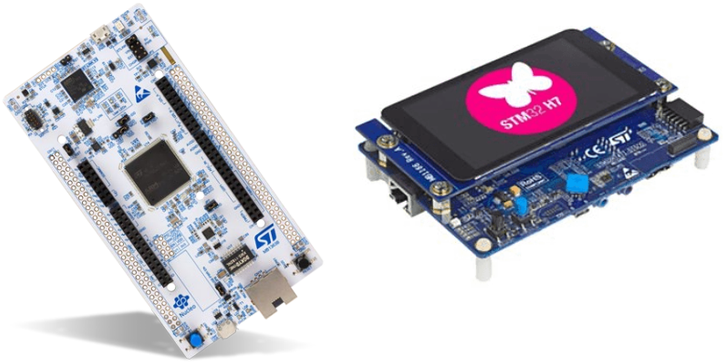
Landscape of embedded Systems (no ASICS or FPGA)



Landscape of embedded Systems (no ASICS or FPGA)



Types of microcontrollers / microprocessors

singlecore microcontroller	symmetrical multicore-processors / microcontrollers	Focus in this module asymmetrical multicore-processors / microcontrollers
Cortex M4	ARM Cortex A72	Cortex M7 + Cortex M4
STM32F429	A72	STM32H7x5/7X7
		
32F429IDISCOVERY	Raspberry Pi 4	

Structure of the lessons

Motivation for C++	•Motivation for C++ in the embedded sector •The difficult topics of C++ in Embedded Systems
C++ Standard	• Overview C, C++ modern C++ • Hello World • Build-Process • Pre-processor • Data types
Boards used in this part	•Microcontroller board Nucleo-STM32F746 •Sensorboard: IKS01A3
First project and ST Environment	•STM32CUBE IDE- Approach •Board Support Package (BSP) •HAL Delay •Frist project: Blinky in C
Enumeration types	
C in C++ projects in STM32CUBEIDE	•First C++ project
Self study	

Learning Objectives

- You will learn what could be the motivation to use C++ on microcontrollers
- You will get an overview of C and C++ standard.
- You will look at the basic program build process.
- You will cover the different data types of C, C++, modern C++ and Java and you will be able to explain this.
- You will cover why not to use the standard data types, but the commonly accepted data types and you will know how to enable and use them.
- You will see which boards we are going to use in this part of the module.
- You will learn how the STMCUBEIDE wizard works and where the information of the board support package can be found.
- You will learn what difference of the enumeration type in C and C++ class is.
- You will see how to setup your first C++ project so you can apply what you have learned today.

Structure of the lessons

Motivation for C++
- Motivation for C++ in the embedded sector
- The difficult topics of C++ in Embedded Systems

C++ Standard
- Overview C, C++ modern C++
- Hello World
- Build-Process
- Pre-processor
- Data types

Boards used in this part
- Microcontroller board Nucleo-STM32F746
- Sensorboard: IKS01A3

First project and ST Environment
- STM32CUBE IDE- Approach
- Board Support Package (BSP)
- HAL Delay
- Frist project: Blinky in C

Enumeration types

C in C++ projects in STM32CUBEIDE
- First C++ project

Self study

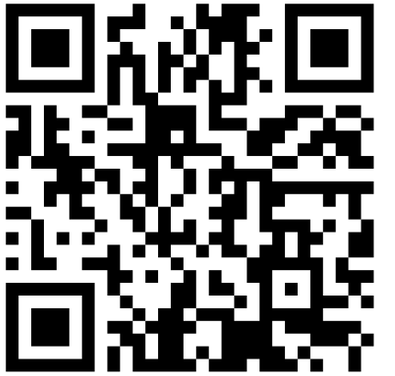
Motivation for C++ in the embedded sector

Motivation for C++ in the embedded area

Answer 2 Questions:

- What could be the pro and cons to use C++ on microcontroller?

https://padlet.com/dominiquekunz1/mse-hs23-what-could-be-the-pro-and-cons-to-use-oq1kt24b8srrtj8z



- Reasons for not to using C++ on Microcontrollers?

https://padlet.com/dominiquekunz1/mse-hs23-reasons-for-not-to-using-c-on-microcor-qlnbi7c6dnxkg9b9

Why C++ is ready for embedded

- The further development of the standard ensures safer and more efficient machine code than with old C++ compilers and C++ versions.
- Reusability through technology and concepts better becomes increasingly important as you no longer do everything yourself from scratch.

- Prerequisites:
  - Knowledge: how to use the C++ properly to make the code as small or smaller than C.
    - const, constexpr, consteval etc.
  - Knowledge: where to watch out in C++ to avoid runtime issues
  - Knowledge and use of mechanisms to make code more understandable and less error-prone.

The difficult topics of C++ in Embedded Systems

The difficult topics of C++

The riskiest issue is dynamic memory allocation, which can occur more often in C++.

Technologies like containers, but also the string class can make very strong use of dynamic memory allocation.

The problem lies in the fragmentation of the memory.

At the PC a MMU provides the automatic defragmentation.

Later more to this topic.

For now, everything with C++ without dynamic memory allocation.

Structure of the lessons

Motivation for C++
- Motivation for C++ in the embedded sector
- The difficult topics of C++ in Embedded Systems

C++ Standard
- Overview C, C++ modern C++
- Hello World
- Build-Process
- Pre-processor
- Data types

Boards used in this part
- Microcontroller board Nucleo-STM32F746
- Sensorboard: IKS01A3

First project and ST Environment
- STM32CUBE IDE- Approach
- Board Support Package (BSP)
- HAL Delay
- Frist project: Blinky in C

Enumeration types

C in C++ projects in STM32CUBEIDE
- First C++ project

Self study

Overview C, C++ modern C++

C: history

C is a procedural language, over 50 years old and still receiving extensions and corrections

K & R C	ANSI C	ANSI ISO C	C99	C11	C18
1978 Kernighan & Ritchie	1989	1990	1999	2011	2018

Linux is largely based on C code.

Successful because:

- many platforms are supported
- Performant code
- Small code size
- Many tools

Widely used in: embedded systems, (decreasing on PC)

a. o.: https://de.wikipedia.org/wiki/C_(Programmiersprache)

C++ and modern C++: history

C++ is an object-oriented language, over 30 years old and still receiving enhancements and fixes.

C++98
1998

- Templates
- STL with Container und algorithms
- Strings (class)
- I/O Streams

C++11
2011

- Move Semantik
- Auto and decltype
- Lambda expressions
- Constexpr
- Multithreading
- Smart Pointer
- Hash-tables

C++14
(small update)
2014

- Reader-writer locks
- Generic lambdas
- Generalized constexpr Funktionen

C++17 (medium update)
2017

- Fold expressions
- Constexpr if
- Strucutrd binding
- std::string_view
- Algorithmen in der STL für parallele Ausführung.
- FilesystemBibliothek
- std_:any,std::optional und std::variant

C++20
2020

- Coroutinen
- Module
- Concepts
- Ranges library

C++23
(mittleres Update)
2023

- Ranges library extended
- Die Views library extended
- String class receives constrains
- Stacktrace library (like in Boost library)
- Container: flat_*

C++26
2026

- constexpr extended
- Static storage for braced initializers
- Placeholder variables with no name
- Hazard Pointers

All advantages of C.

Widely used in: PC applications, increasingly used in embedded systems.

C++ and modern C++: history

The versions from C++11 on are often called modern C++.
So that the code is really modern C++ one should use the new elements from C++11 onward.

C++98	C++11	C++14	C++17	C++20	C++23	C++26
1998	2011	(small update) 2014	(medium update) 2017	2020	(mittleres Update) 2023	2026

Detailed features and Compiler support can be found on:
https://en.cppreference.com/w/cpp/compiler_support

C and C++ on microcontroller

C: full range of languages applicable

- C compilers mostly supported all versions and functions of the C standard.

C++/modern C++:

- Which version of the C++ standard does the C++ compiler support?
- Is there a Memory Management Unit (MMU)?
  - Yes, no restrictions
  - No, Usable with restrictions (on microcontrollers)

Hello World

«Hello World» on PC

C

```c
#include <stdio.h>

int main() {
   //Output
   printf("Hello, World!\n");
   /* return */
   return 0;
}
```

C++

```cpp
#include <iostream>

int main() {
   // Output
   std::cout << "Hello, World!" << std::endl;
   printf("printf");
   /* return */
   return 0;
}
```

Java

```java
package com.company;

public class Main {

   public static void main(String[] args) {
      //Ausgabe
      System.out.println("Hello World");
      /* Rückgabe */
   }
}
```

What is C <-> C++ specific?

main is a class and a method

System is a class

Build-Process

Build-Process

The build process is the translation of the source code into executable code.

The compiler interprets the source code line by line.

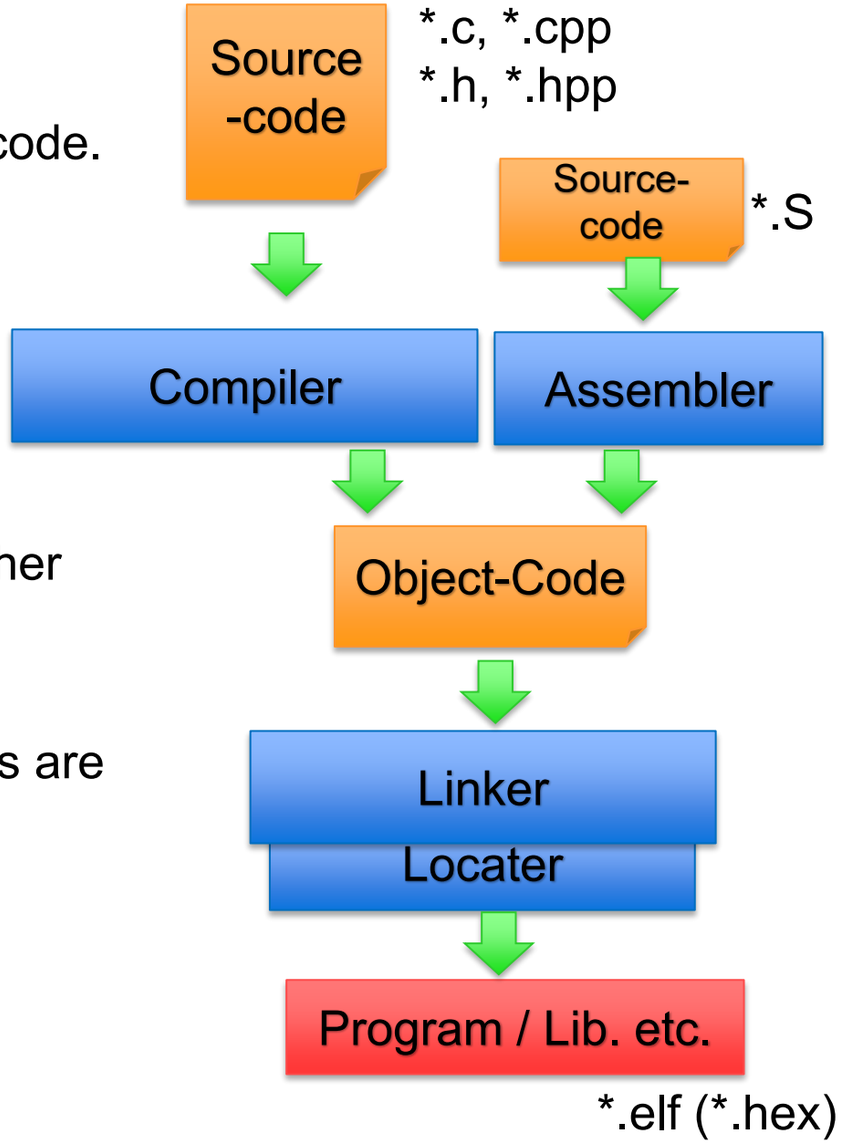
As soon as an error occurs, it terminates the translation.

The Linker connects the object code (arises from a source file) with further translated program parts to a whole.

The Locater defines where in the memory the program and the variables are arranged (above all with Embedded systems relevant).

C, C++ is direct, few automatisms (no magic).

As long as you use only one file... no problem.



Pre-processor

Pre-processor of C and C++

Tasks:

Adds more files, substitutes symbolic constants and macros, conditional compilation.

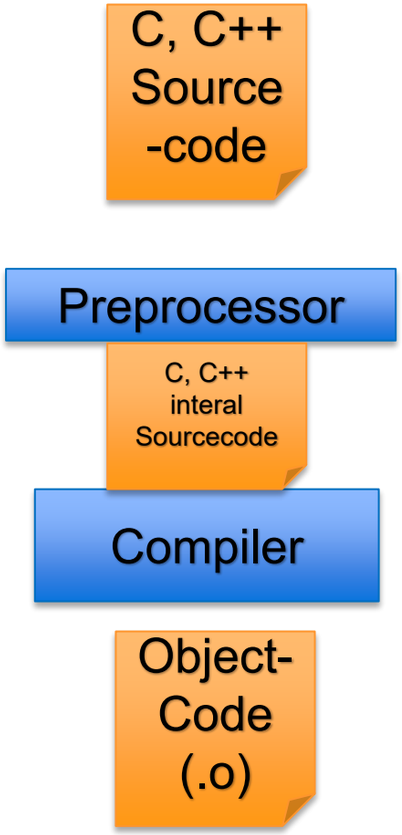
The preprocessor commands start with a #.

There are no differences between C and C++.

Sequence:

- Insert trigraph sequences (??= corresponds to #, ??:/ corresponds to \)
- Replace comments with whitespace
- Execute preprocessor statements: Expand macros, insert files, implement conditional compilation, etc.
- Replace escape sequences and string-literals (text characters enclosed in double-execution characters).
- Join adjacent string.

This results in the internal source code that is passed to the compiler.



Use pre-processor directives for:

- Store information in program code during compilation.

- Develop code for multiple platforms and use conditional compilation to adapt each to the target platform.
- Populate code differently for debug and release with conditional compilation.

- Create macros.

- Create constants.

- Mark functions as interrupt functions.

…

Pre-processor directives

- The directives start with a #

- The lines do not have to be terminated with a ;

- There may be only one directive per line
  - If a command is to extend over several lines, there must be a "\" at the end of each line.

  The last line must not end with a "\"!

Use preprocessor directives for:
Predefined symbols-. Store information in program code during compilation

Predefined constants :

__LINE__	contains the current line
__FILE__	contains the current file
__DATE__	date of compilation in the format "mmm sss yyy"
__TIME__	time of compilation in the format "hh:mm:ss“
__STDC__	is set if "ANSI Standard C" is used

e.g.:

```
std::cout<<(__TIME__);
```

```
const char compileinfo[] = {__DATE__};
```

Preprocessor directives: constant

```
#define  PI (3.141f)
```

Where in the code PI is used the code (3.141f) is copied in.

It behaves almost like a constant, but the value is copied.

```
const float PI = 3.141f;
```

Is stored in the program memory at one location and if it is used in the code it is fetched from this address.

In a 32-Bit architecture:

If the constant has a 64-Bit size then memory is saved, because it is stored once...

If the constant is 8-Bit and used only twice, more code is used because 32 bit address is stored where the constant is located.

Pre-processor directives: MACRO

```
#define  MAX(a,b) \
    ({ __typeof__ (a) _a = (a); \
       __typeof__ (b) _b = (b); \
     _a > _b ? _a : _b; })
```

Where MAX is used the code is copied in.
It looks like a method, but it is code.

The whole thing is called a macro because it implements a function.

Preprocessor directives: #include and #import

```
#include <stdio.h >
```

Loads the contents of the file stdio.h from the default include directory and adds it to the file

#include

- Instructs the preprocessor to copy the entire contents of the file into the source code

- #import
  - Same functionality as #include
  - With the addition that files loaded with #import cannot be loaded mor than once in the entire source code. (However, the gcc documentation warns not to rely on this).

- Use <filename> to search for the file in the installation directory.

- File name" is used to search in the current working directory

- Files may again contain preprocessor directives, e.g. another #include , #import command

Präprozessor Direktiven: #include

```
#include <stdio.h >
```

Prevent multiple `#include` with GUARDS: `#ifndef....#endif`

```
#ifndef __MODULNAME_H
#define __MODULNAME_H
//Header content

#endif
```

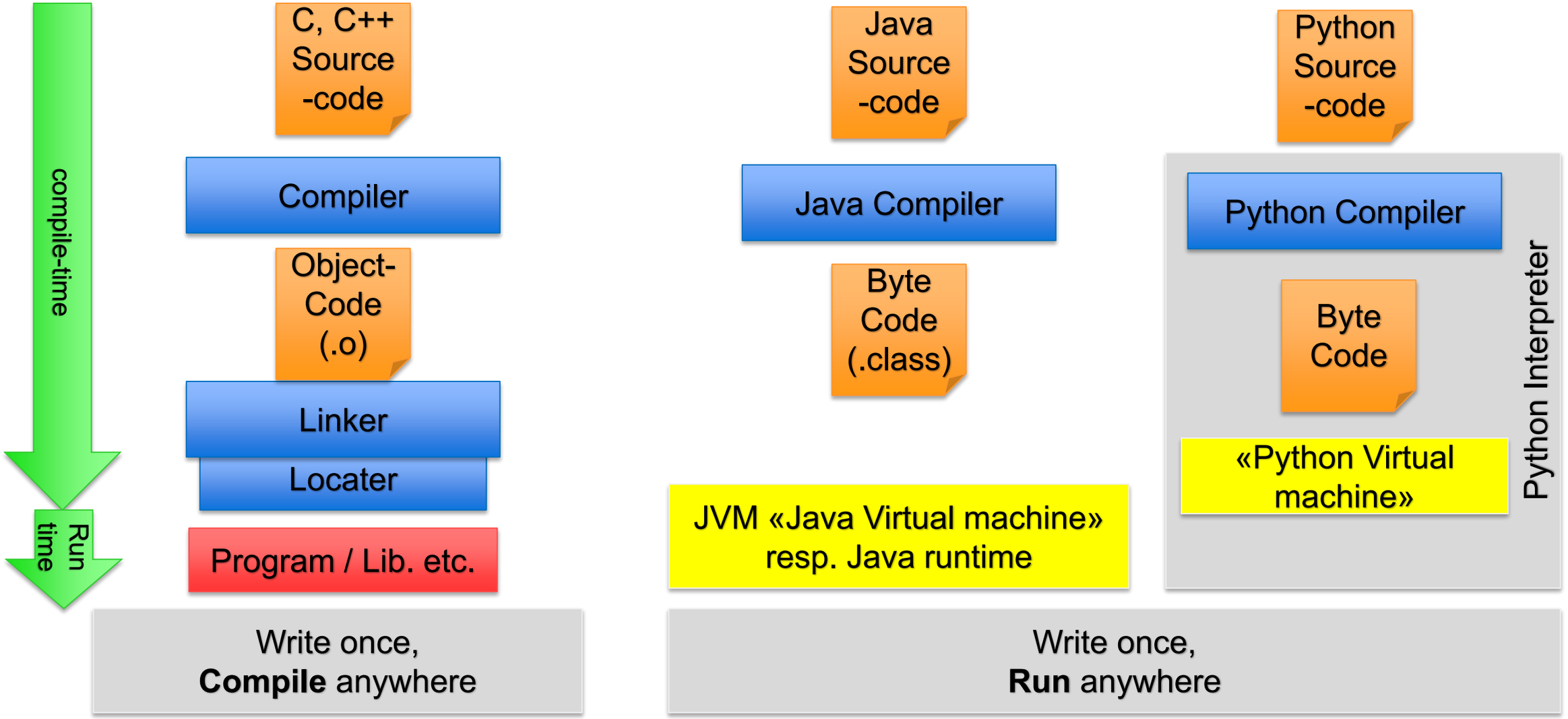
Alternative : `#pragma once`

File has the name: `modulname`

The module consists of:

`modulname.h` and `modulname.c` or `modulname.h` and `modulname.cpp`

C,C++ , Java , Python



Data types

Data types: numbers

C

Type name	Size / Byte	Range	System dependent
char unsigned char	1	-128..127, ASCII, 0..255	No
short unsigned short	2 2	$-2^7...2^7-1$ $0...2^{16}-1$	No
int unsigned int	2 (4) 2 (4)	$-2^{15}...2^{15}-1$ $0...2^{32}-1$	Yes
long unisgned log	4 (8) 4 (8)	$-2^{31}...2^{31}-1$ $0...2^{64}-1$	Yes
float	4	+/- 3.40282347 * 10^{38}	No
double	8	+/- 1.797693134 86231570 * 10^{308}	No

C++

Type name	Size / Byte	Range	System dependent
Bool	1	True, false	false
char unsigned char	1	-128..127, ASCII, 0..255	No
char16_t, char32_t	2, 4	unicode	No from C++11
short unsigned short	2 2	$-2^7...2^7-1$ $0...2^{16}-1$	No
int unsigned int	2 (4) 2 (4)	$-2^{15}...2^{15}-1$ $0...2^{32}-1$	Yes
long unisgned log	4 (8) 4 (8)	$-2^{31}...2^{31}-1$ $0...2^{64}-1$	Yes
long long unsigned long long	8 8	$-2^{63}...2^{63}-1$ $0...2^{128}-1$	No
float	4	+/-3.40282347 * 10^{38}	No
double	8	+/- 1.797693134862 31570 * 10^{308}	No

Java

Type name	Size / Byte	Range	Default value
boolean	1	true, false	false
char	2	all unicode-signs	\u0000
byte	1	$-2^7...2^7-1$	0
short	2	$-2^{15}...2^{15}-1$	0
int	4	$-2^{31}...2^{31}-1$	0
long	8	$-2^{63}...2^{63}-1$	0
float	4	+/- 3.40282347 * 10^{38}	0.0
double	8	+/- 1.79769313 486231570 * 10^{308}	0.0

Data types, Best Practice !!

The standard data type names are unsuitable for projects that should be portable, i.e.: reuse of code on different systems, because the data size varies depending on the system.

Solution:

The data types are used according to MISRA-C.

- In C either include **"stdint.h" (C99**)
  or rename the data types at one specific place in the project.

- In C++ either include **"cstdint"** or "iostream".

Only this way!

Note:

Projects with STM32CubeIDE, already support these data types.

Typname	Grösse / Byte	Werte-bereich	Grösse System-Abhängig
char int8_t uint8_t	1	ASCII -128..127, 0..255	Nein
char16_t, char32_t	2 4	Unicode	nein
int16_t uint16_t	2 2	$-2^{7}...2^{7}-1$ $0...2^{16}-1$	nein
int32_t uint32_t	4 4	$-2^{15}...2^{15}-1$ $0...2^{32}-1$	nein
int64_t uint64_t	8 8	$-2^{31}...2^{31}-1$ $0...2^{64}-1$	nein
int128_t uint128_t	16 16	$-2^{63}...2^{63}-1$ $0...2^{128}-1$	nein
float (oder float32_t)	4	+/- 3.40282347 * 10^{38}	nein
double (oder float64_t)	8	+/- 1.797693134 86231570 * 10^{308}	nein

Data types: String

In C:
Single characters (letters) are stored in the data type char.
If you want to store a string you have to store it in a collection of chars.
This is done with a field of chars (array)
char[100] //store 100 characters.

In C++ there is a string class like in Java, more about in the following lessons.

Structure of the lessons

Motivation for C++
- Motivation for C++ in the embedded sector
- The difficult topics of C++ in Embedded Systems

C++ Standard
- Overview C, C++ modern C++
- Hello World
- Build-Process
- Pre-processor
- Data types

Boards used in this part
- Microcontroller board Nucleo-STM32F746
- Sensorboard: IKS01A3

First project and ST Environment
- STM32CUBE IDE- Approach
- Board Support Package (BSP)
- HAL Delay
- Frist project: Blinky in C

Enumeration types

C in C++ projects in STM32CUBEIDE
- First C++ project

Self study

Microcontroller board Nucleo-STM32F746

Hardware

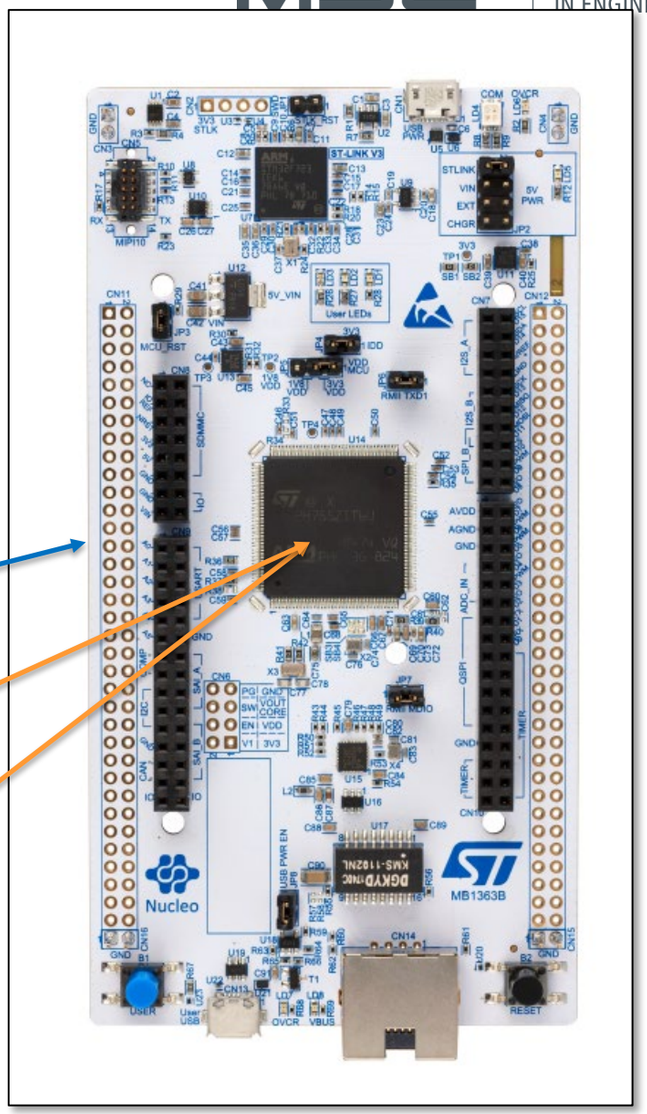
The core is the STM32H745 Nucleo Board

Therefore we need:

Board manual: UM2408: User manual STM32H7 Nucleo-144 boards

Programmers manual: PM0253 Programming manual

Reference manual: STM32H745xI/G



https://www.st.com/en/evaluation-tools/nucleo-h745zi-q.html

Documentation

Nucleo documentation:
https://www.st.com/en/evaluation-tools/nucleo-h745zi-q.html#documentation:
Nucleo Board manual:
https://www.st.com/resource/en/user_manual/um2408-stm32h7-nucleo144-boards-mb1363-stmicroelectronics.pdf

Microcontroller documentation :
https://www.st.com/en/microcontrollers-microprocessors/stm32h745-755.html#documentation

Programming handbook
https://www.st.com/resource/en/programming_manual/pm0253-stm32f7-series-and-stm32h7-series-cortexm7-processor-programming-manual-stmicroelectronics.pdf

Microcontroller handbook
https://www.st.com/resource/en/datasheet/stm32h745zg.pdf

Referenz handbook
https://www.st.com/resource/en/reference_manual/rm0399-stm32h745755-and-stm32h747757-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

Sensorboard: IKS01A3

IKS01A3

The Sensor Board contains 6 sensors.

We will use:

HTS221 and LIS2DW12

The sensors can be accessed over I2C or SPI.

In our module mainly over I2C.

X-NUCLEO-IKS01A3 Hardware description

- The X-NUCLEO-IKS01A3 is a motion MEMS and environmental sensor evaluation board system.
- It is compatible with the Arduino UNO R3 connector layout, and is designed around ST's latest sensors.

Key products on board

LSM6DSO
MEMS 3D accelerometer (±2/±4/±8/±16 g) + 3D gyroscope (±125/±250/±500/±1000/±2000 dps)

LIS2DW12
MEMS 3D accelerometer (±2/±4/±8/±16 g)

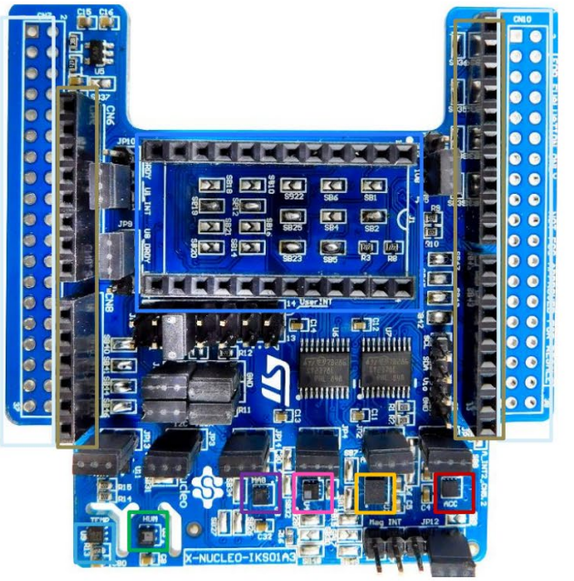
LIS2MDL
MEMS 3D magnetometer (±50 gauss) +

LPS22HH
MEMS pressure sensor, 260-1260 hPa absolute digital output barometer

HTS221
Capacitive digital relative humidity and temperature

STTS751
Digital Temperature sensor

DIL 24-pin
Socket available for additional MEMS adapters and other sensors (UV index)



Latest info available at www.st.com
X-NUCLEO-IKS01A3

** Connector for the STM32 Nucleo Board

Structure of the lessons

Motivation for C++
- Motivation for C++ in the embedded sector
- The difficult topics of C++ in Embedded Systems

C++ Standard
- Overview C, C++ modern C++
- Hello World
- Build-Process
- Pre-processor
- Data types

Boards used in this part
- Microcontroller board Nucleo-STM32F746
- Sensorboard: IKS01A3

First project and ST Environment
- STM32CUBE IDE- Approach
- Board Support Package (BSP)
- HAL Delay
- Frist project: Blinky in C

Enumeration types

C in C++ projects in STM32CUBEIDE
- First C++ project

Self study

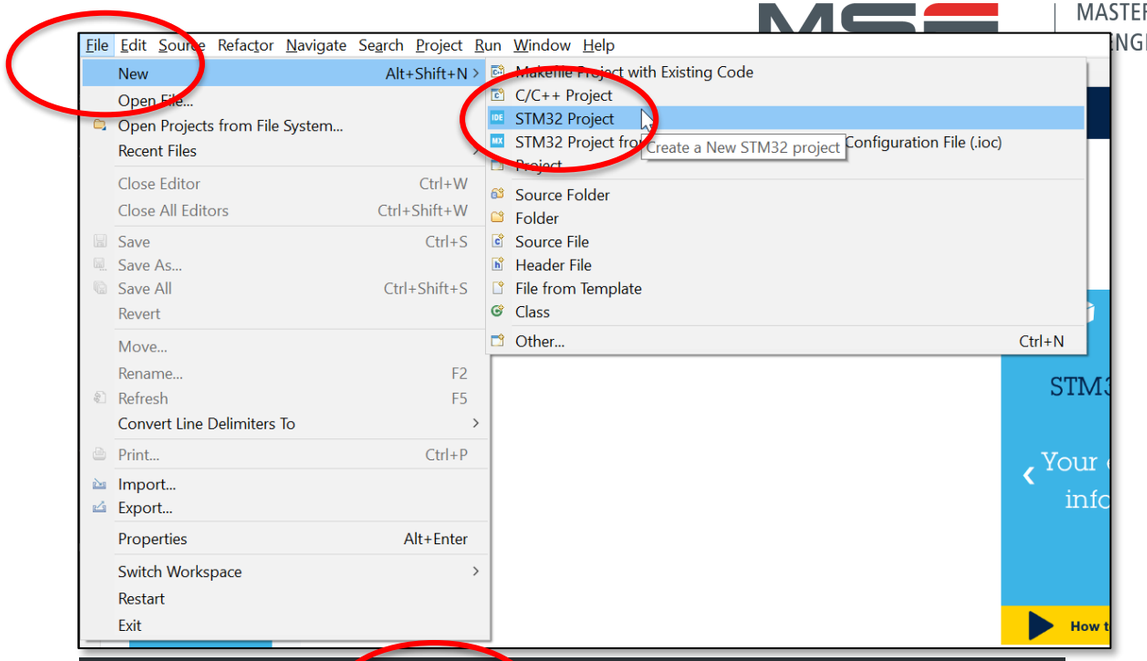
STM32CUBE IDE- Approach

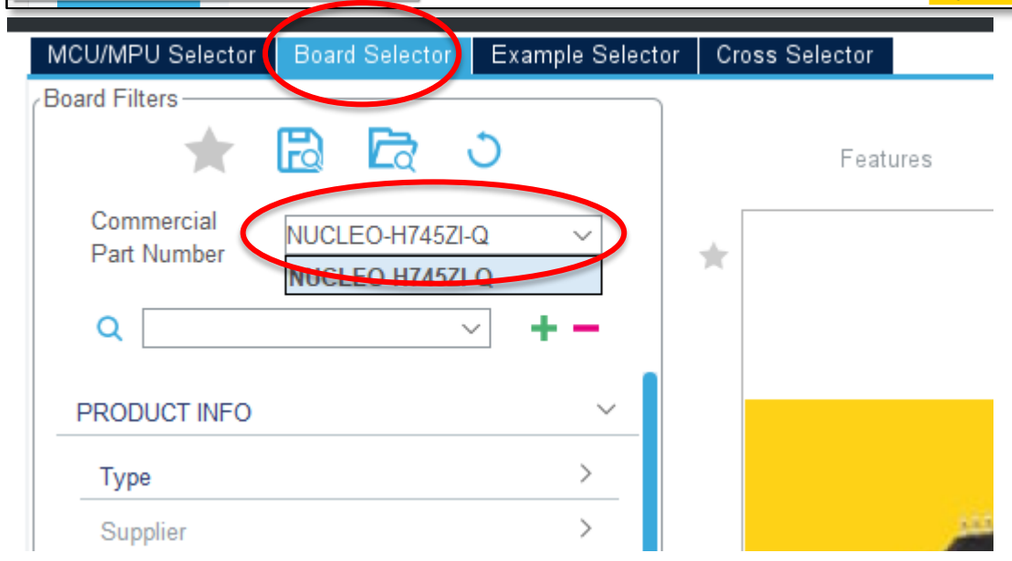
Generate an new STM32 project

Generate a new project by:
File→New→STM32 Project

Choose:
Board Selector→NUCLEO-H745ZI-Q…
Select the board from the list and press next.

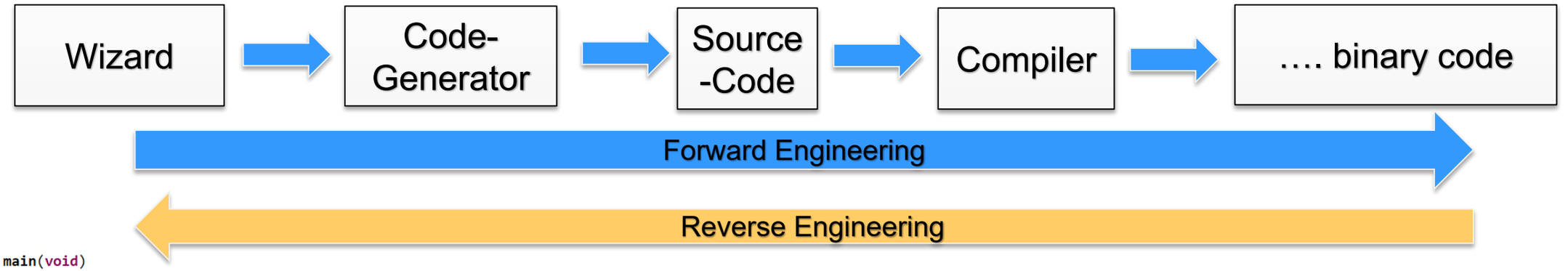
All following steps can be found on moodle in:
«Part1_SetupEmptyProject_ProfDKunz.pdf»





Typical development toolchain problem with wizards and code generation!

- Configuration
- UML-Diagrams
- XML




```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */
/* USER CODE BEGIN Boot_Mode_Sequence_0 */
  int32_t timeout;
/* USER CODE END Boot_Mode_Sequence_0 */

/* USER CODE BEGIN Boot_Mode_Sequence_1 */
  /* Wait until CPU2 boots and enters in stop mode or timeout*/
  timeout = 0xFFFF;
  while((__HAL_RCC_GET_FLAG(RCC_FLAG_D2CKRDY) != RESET) && (timeout-- > 0));
  if ( timeout < 0 )
  {
  Error_Handler();
  }
/* USER CODE END Boot_Mode_Sequence_1 */
  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */
```

development toolchain are "mostly" forward directed... backward is "mostly" not possible

- Reserved areas for code generated by toolchain
- Reserved areas for code generated by developer

Board Support Package (BSP)

Board Support Package (BSP)

BSP contains board specific, i.e. concrete implementation for the hardware at hand.

HAL is only a library to control the MCU hardware. The translation to the present project belongs one level above implemented.

If one looks at the example projects of STM, then the BSP is copied with.

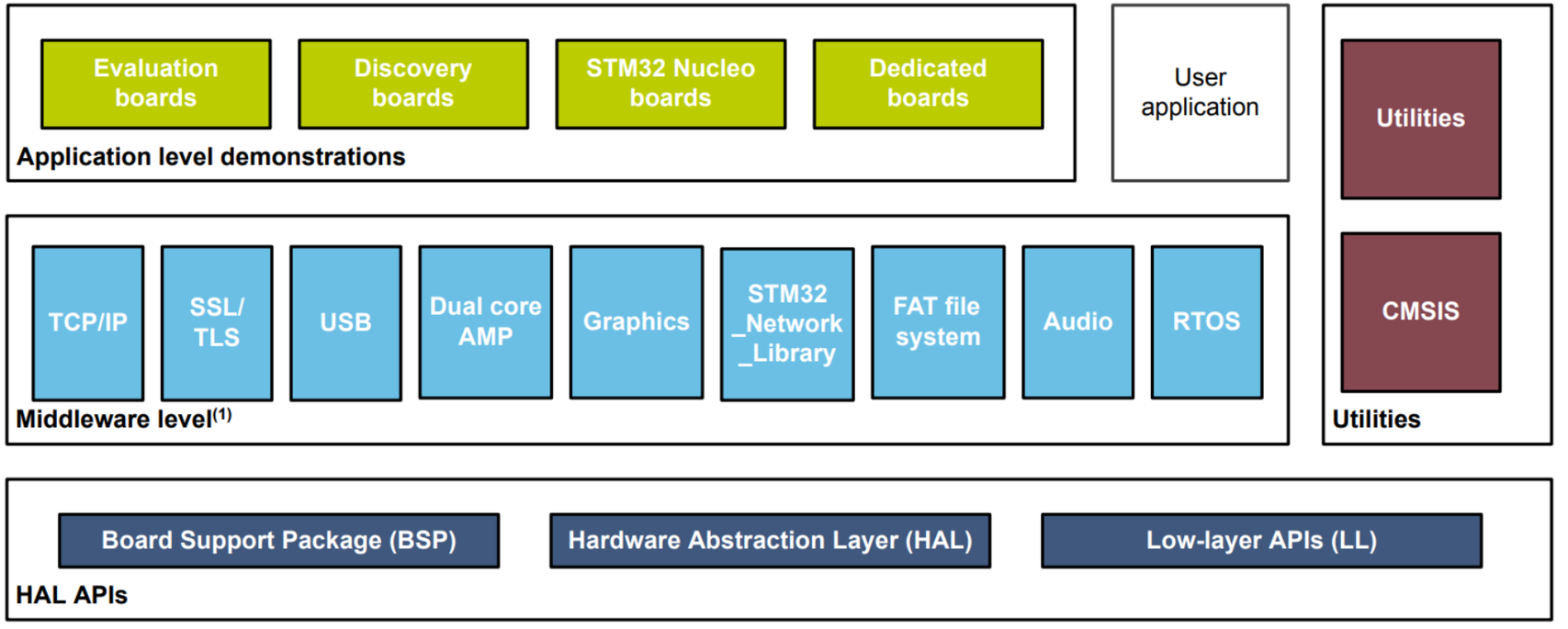
BSP of ST

The BSP here is placed on the same level as HAL, but it uses HAL for implementation.

The BSP is good as an idea generator.

For projects one will write his own BSP.

Figure 1. **STM32CubeH7 firmware components**



(1) The set of middleware components depends on the product Series.

ST: um2204-getting-started-with-stm32cubeh7-for-stm32h7-series--stmicroelectronics.pdf

Datenblätter

STM32F7 e.g. documentation:

https://www.st.com/resource/en/user_manual/um1891-getting-started-with-stm32cubef7-mcu-package-for-stm32f7-series-stmicroelectronics.pdf

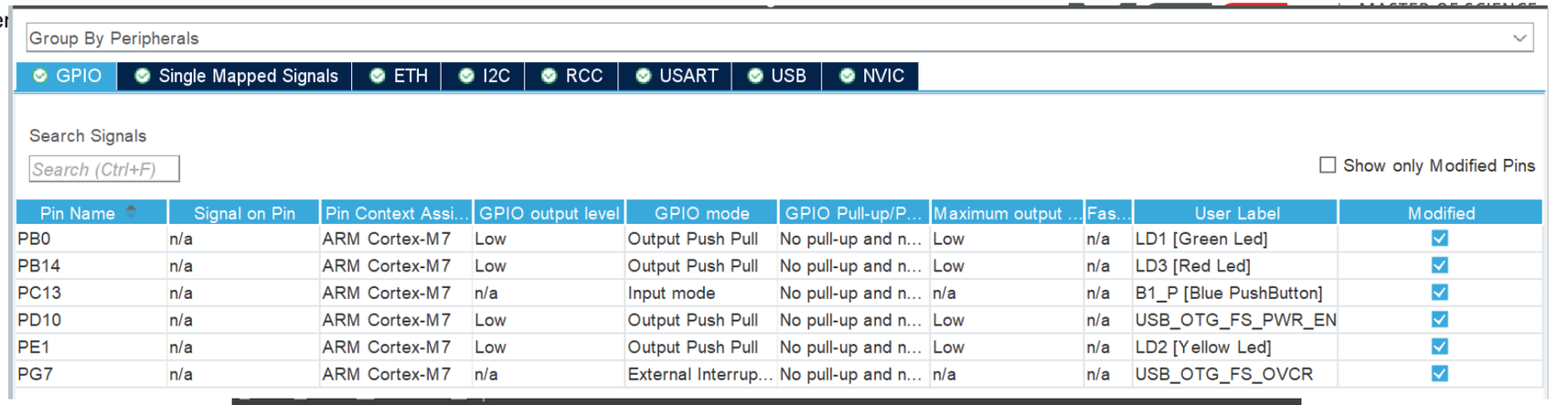
On the way to your BSP

A BSP translates the general port and pin designations to the application specific names.

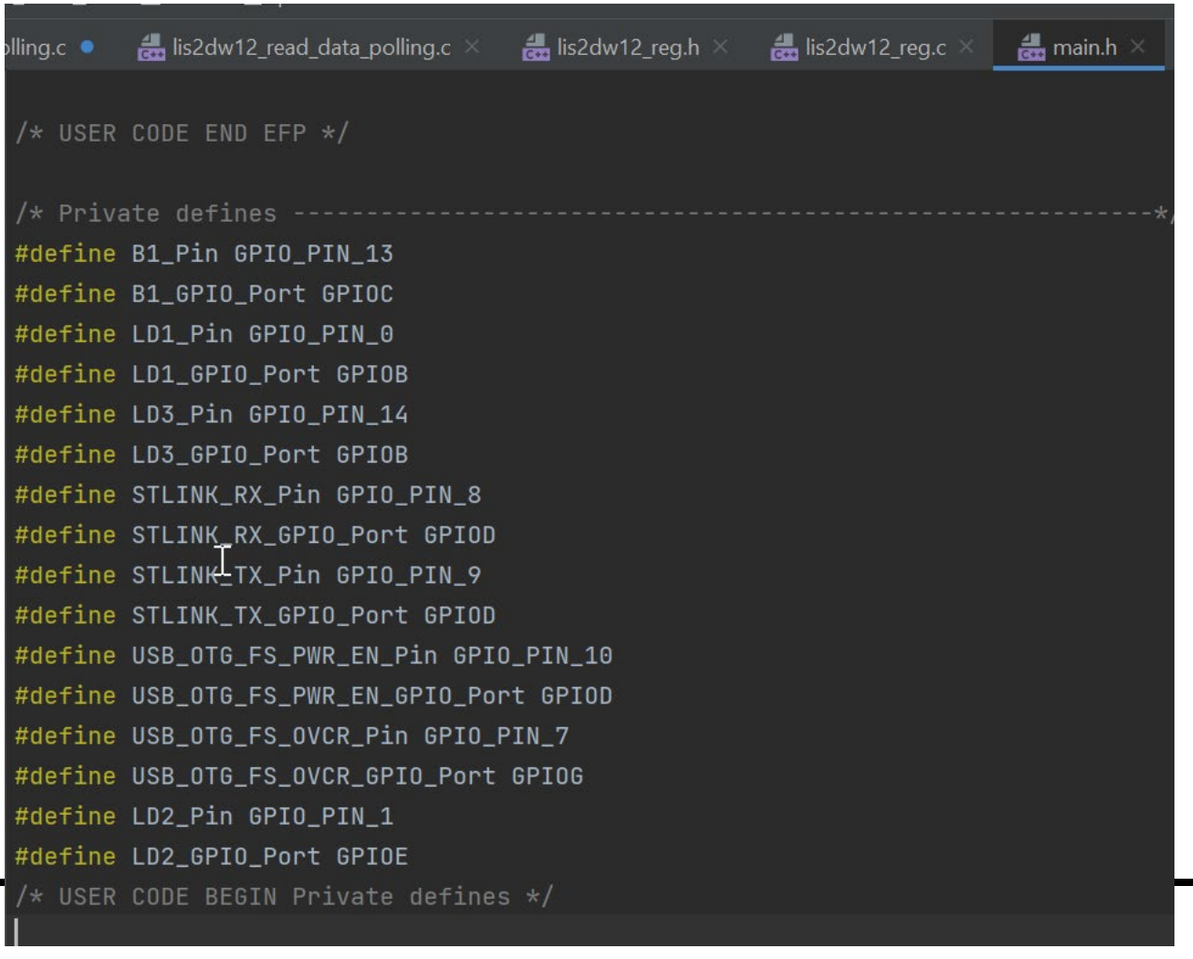
This can be achieved with the configurator of ST.

User Labels generate defines in main.h

Comments = [ ]



Pin Name	Signal on Pin	Pin Context Assi...	GPIO output level	GPIO mode	GPIO Pull-up/P...	Maximum output ...	Fas...	User Label	Modified
PB0	n/a	ARM Cortex-M7	Low	Output Push Pull	No pull-up and n...	Low	n/a	LD1 [Green Led]	☑
PB14	n/a	ARM Cortex-M7	Low	Output Push Pull	No pull-up and n...	Low	n/a	LD3 [Red Led]	☑
PC13	n/a	ARM Cortex-M7	n/a	Input mode	No pull-up and n...	n/a	n/a	B1_P [Blue PushButton]	☑
PD10	n/a	ARM Cortex-M7	Low	Output Push Pull	No pull-up and n...	Low	n/a	USB_OTG_FS_PWR_EN	☑
PE1	n/a	ARM Cortex-M7	Low	Output Push Pull	No pull-up and n...	Low	n/a	LD2 [Yellow Led]	☑
PG7	n/a	ARM Cortex-M7	n/a	External Interrup...	No pull-up and n...	n/a	n/a	USB_OTG_FS_OVCR	☑



HAL Delay

Delay

How to implement a delay classically?

HAL_Delay

What is the time base?

Where is the blocking section?

What is special about this delay?

hint if an OS would be active?

```
/**
  * @brief This function provides minimum delay (in milliseconds) based
  *        on variable incremented.
  * @note In the default implementation , SysTick timer is the source of time base.
  *       It is used to generate interrupts at regular time intervals where uwTick
  *       is incremented.
  * @note This function is declared as __weak to be overwritten in case of other
  *       implementations in user file.
  * @param Delay  specifies the delay time length, in milliseconds.
  * @retval None
  */
__weak void HAL_Delay(uint32_t Delay)
{
  uint32_t tickstart = HAL_GetTick();
  uint32_t wait = Delay;

  /* Add a freq to guarantee minimum wait */
  if (wait < HAL_MAX_DELAY)
  {
    wait += (uint32_t)(uwTickFreq);
  }

  while ((HAL_GetTick() - tickstart) < wait)
  {
  }
}
```

Manuals

STM32H7 HAL documentation :

https://www.st.com/resource/en/user_manual/um1905-description-of-stm32f7-hal-and-lowlayer-drivers-stmicroelectronics.pdf

(
STM32H7 BSP documentation:

https://www.st.com/resource/en/user_manual/um1891-getting-started-with-stm32cubef7-mcu-package-for-stm32f7-series-stmicroelectronics.pdf
)

ARM Compiler keywords:

https://developer.arm.com/documentation/dui0491/i/Compiler-specific-Features/

Frist project: Blinky in C

01_1_Blinky

Create a new project: C_Blinky.

Use the HAL library to control the LED2 from the NucleoBoard every 500ms.

Note:
Search in HAL functions: Toggle and Delay
Press CTRL SPACE for autocompletion

01_2_BlinkyDelayNotBlocking

Expand project: C_Blinky.

Use the SystemTick timer to implement a software delay that does not block.

Note: HAL Delay

Structure of the lessons

Motivation for C++
- Motivation for C++ in the embedded sector
- The difficult topics of C++ in Embedded Systems

C++ Standard
- Overview C, C++ modern C++
- Hello World
- Build-Process
- Pre-processor
- Data types

Boards used in this part
- Microcontroller board Nucleo-STM32F746
- Sensorboard: IKS01A3

First project and ST Environment
- STM32CUBE IDE- Approach
- Board Support Package (BSP)
- HAL Delay
- Frist project: Blinky in C

Enumeration types

C in C++ projects in STM32CUBEIDE
- First C++ project

Self study

Enumeration types

Enumeration types

Enumeration type `enum` are different in C and C++ than in Java, only from C++11 onwards they are similar.

Java:

```
public class Main {
   enum Level {
      LOW,
      MEDIUM,
      HIGH
   }

   public static void main(String[] args) {
      Level myVar = Level.MEDIUM;
      System.out.println(myVar);
   }
}
```

In Java, an `enum` is more like a class with members

```
MEDIUM
```

C and C++:

```
enum Level { LOW, MEDIUM, HIGH };
enum Level myVar = MEDIUM;
```

In C and C++ the elements of an enum are converted into an integer!

In the code one can use the text, however the numbers are in the background.

```
printf("%s""%d""%s","enum value  \t",myVar,"\n");
```

```
enum value      1
```

Ab C++11

`enum class` or `enum struct`

```
enum class Level { LOW, MEDIUM, HIGH };
```

```
enum Level myVar = Level::MEDIUM;
```

```
printf("%s""%d""%s","myVar  \t",myVar,"\n");
```

```
myVar   1
```

Enumeration type in C und C++

In C and C++ the elements of an enum are converted to an integer.
By default the numbering starts at 0.

0, 1, 2, 3, 4

```
enum state {INIT,START,OPERATING1,OPERATING2,FAULT};
```

You can specify the number.

0, 1, 11, 12

```
enum state {INIT,START,OPERATING1=10,OPERATING2,FAULT};
```

If nothing is given, then the numbering continues with the next higher number.
The order of the numbers does not have to be strictly monotonically increasing.

```
enum state {INIT=-10,START=100,OPERATING1=10,OPERATING2=10,FAULT=-20};
```

But enum is unscoped!
I.e. enum definition is valid for the whole project.
Names may occur only once!

```
enum state {INIT,START,OPERATING1,OPERATING2,FAULT};
enum state2 {INIT,START,OPERATING1=10,OPERATING2,FAULT};
```

```
D:\U_Unterricht\SYSTECH\CPPPR\ab2021\test_tag4\main.c:51:16: error: redeclaration of enumerator 'INIT'
   enum state2 {INIT,START,OPERATING1=10,OPERATING2,FAULT};
```

Enumeration type in C++11 as class or struct

In C++11 with enum class the terms are handled as scope (scope defined).

Consequently, elements may occur more than once in the project.

```
enum class state {INIT=-10,START=100,OPERATING1=10,OPERATING2,FAULT=-20};
enum class state1 {INIT=-10,START=100,OPERATING1=10,OPERATING2=20,FAULT=-20};
```

With the assignment one must indicate the class

```
enum state state1 = state::INIT;
enum state1 state2 = state1::OPERATING1;
enum state1 state3 = state1::OPERATING2;
enum state state4 = state::FAULT;
```

The rest remains the same as in C and C++ classic:

In C and C++ classic the elements of an enum are converted to an integer.

By default the numbering starts at 0.

You can specify the number.

If nothing is given, then the numbering continues with the next higher number.

The order of the numbers does not have to be strictly monotonically increasing.

Enumeration type in C an C++

Why should you use enum instead of #define?

enum

C, C++

- Without scope
- Automatic numbering
- Is a new data type
- Symbolic debugging
- Compiler can better check the code

C++11

- Scoped
- Automatic numbering
- Is a new class
- Symbolic debugging
- Compiler can better check the code

#define

C, C++

- Without scope
- Numbering by hand
- Is a symbol
- Returns number in debug

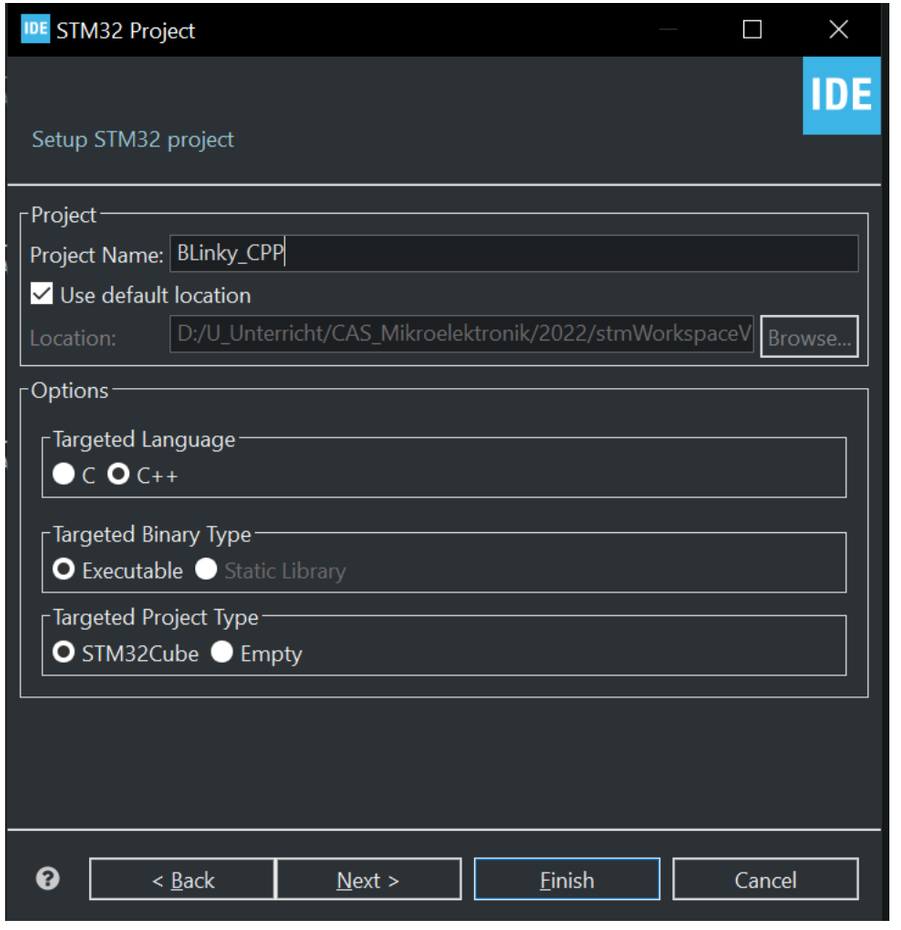
Therefore, it is recommended to use enum most of the time.

Structure of the lessons

Motivation for C++	• Motivation for C++ in the embedded sector • The difficult topics of C++ in Embedded Systems
C++ Standard	• Overview C, C++ modern C++ • Hello World • Build-Process • Pre-processor • Data types
Boards used in this part	• Microcontroller board Nucleo-STM32F746 • Sensorboard: IKS01A3
First project and ST Environment	• STM32CUBE IDE- Approach • Board Support Package (BSP) • HAL Delay • Frist project: Blinky in C
Enumeration types	
C in C++ projects in STM32CUBEIDE	•First C++ project
Self study	

C++ projects in STM32CUBEIDE

Generate C or C++ projects



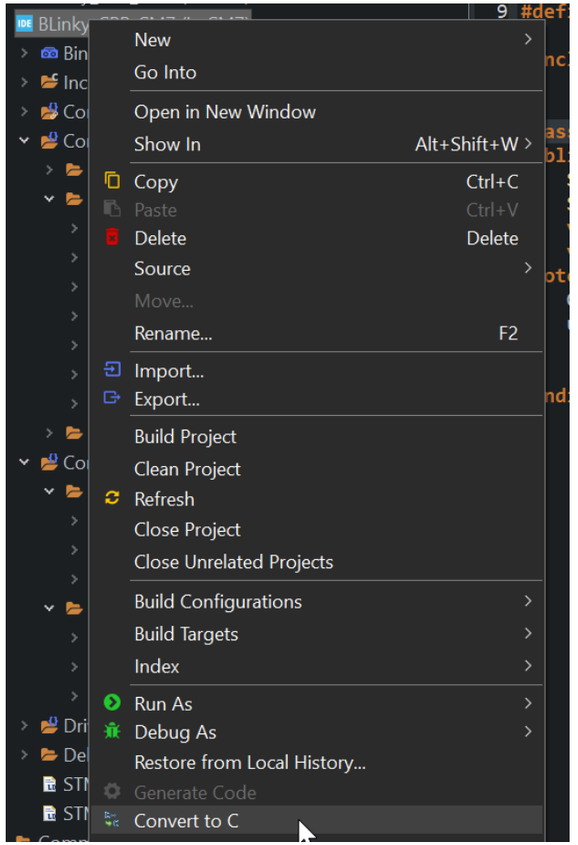
Projects

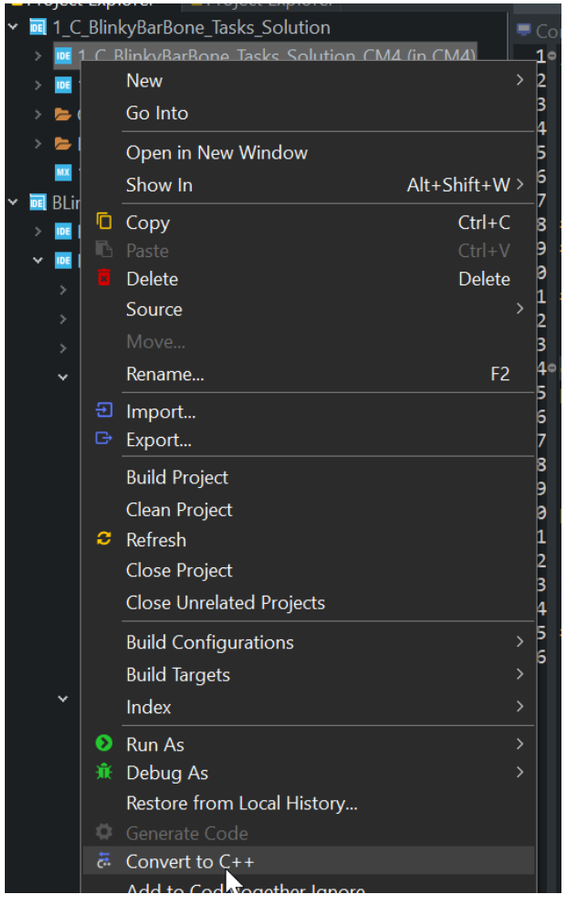
One can create new projects as C++.

You can convert existing projects to C++.

You can convert existing C++ projects to C.

Sounds very comfortable... but what happens with the sources in the project?





How to use C in C++?

1. C is in many ways the same as C++
2. You can use all commands and mostly functions of C in C++.

So no problem?

But the preprocessor has other mechanisms in C++ because you can overlay methods!

Therefore the preprocessor creates new IDs for the linker.

In C the names (function and variable) are also the ID.

Term: **Name mangeling**

```
#include "rectangle.h"

double rectangle::getArea (void)
{
  return(Area);
}

void rectangle::setArea (double Area)
{
  rectangle::Area = Area;
}

void rectangle::resetAreaToZero (void)
{
  Area = 0;
}

void rectangle::calcArea (double length, double width)
{
  Area= width * length;
}

void rectangle::calcArea (void)
{
Area=width*length;
}
```

How to use C in C++?

Because of the **name mangeling** we must specify to the preprocessor if code is in C or in C++.

This is done with GUARDS:

```
#ifdef __cplusplus
extern "C" {
#endif

//C Code

#ifdef __cplusplus
}
#endif
```

```
#include "rectangle.h"

double rectangle::getArea (void)
{
  return(Area);
}

void rectangle::setArea (double Area)
{
  rectangle::Area = Area;
}

void rectangle::resetAreaToZero (void)
{
  Area = 0;
}

void rectangle::calcArea (double length, double width)
{
  Area= width * length;
}

void rectangle::calcArea (void)
{
Area=width*length;
}
```

Project conversion

The only thing that happens with project conversion is:
GUARDS are added or removed!

Nothing more... no magic

So C-code remains C-code and C++ code remains C++ code

STM32CUBE IDE C++ Project

What the wizard creates is a C project with GUARDS!

There are 2 ways to introduce and use C++:

1. Convert main to C++

2. Create a new main as C++ main extern and call it from main.

But here is only one sustainable solution!

Which one?

Let's try way 1 and way 2!

way 1 main as cpp

way 2: in main call a c++ main here looper

```
 /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
looper();
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

```
/*
 * looper.h
 *
 *  Created on: Mar 14, 2022
 *      Author: dominique.kunz
 */

#pragma once

#ifdef __cplusplus
extern "C" {
#endif

void looper();

#ifdef __cplusplus
}
#endif
```

```
/*
 * looper.cpp
 *
 *  Created on: Mar 14, 2022
 *      Author: dominique.kunz
 */

#include "looper.h"
#include "main.h"

void looper()
{

while (1){

 }
}
```

Good Practice in STM32 C++ projects

- Create a folder for C++ files....
  maybe one folder for headers and one for C++

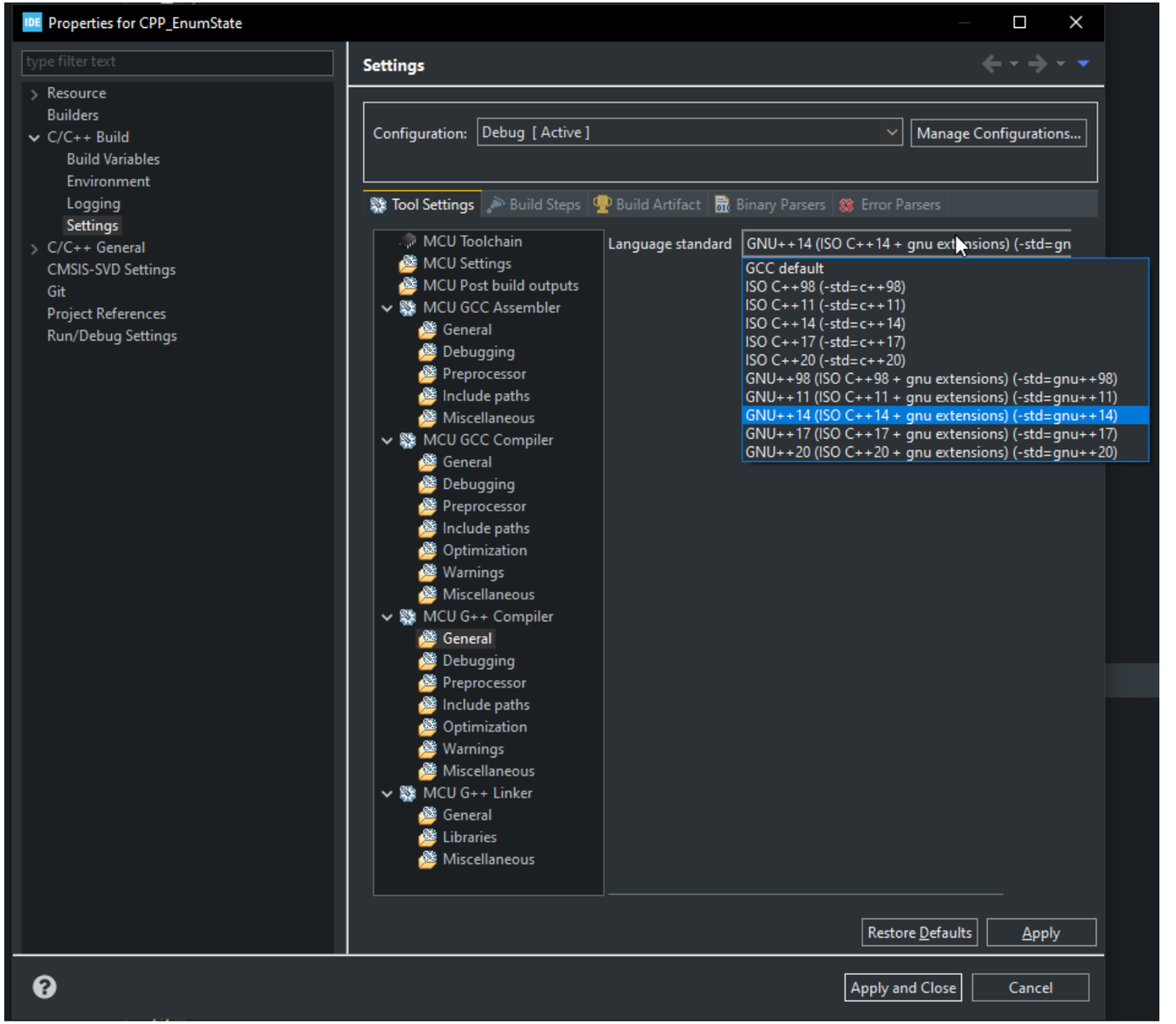
- Create main of C++ in a new file

Check and set the compiler version

- Select Project
- Right Click or Menu→File
  - Properties

- C/C++ Build
  - MCU G++ Compiler
    - General

  Select Compiler Version C++ 20



What we have learned

- We have learned what the motivation could be to use C++ on microcontrollers
- We have got an overview of C and C++ standard.
- We have looked at the basic program build process.
- We have covered the different data types of C, C++, modern C++ and Java.
- We have covered why not to use the standard data types, but the commonly accepted data types and we have seen how to enable and use them.
- We have seen which boards we are going to use in this part of the module.
- We have learned how the STMCUBEIDE wizard works and where the information of the board support package can be found.
- We have learned what the difference of the enumeration type in C and C++ class is.
- We have discussed how to setup the first C++ project so you can apply what we have learned today.

Structure of the lessons

Motivation for C++
- Motivation for C++ in the embedded sector
- The difficult topics of C++ in Embedded Systems

C++ Standard
- Overview C, C++ modern C++
- Hello World
- Build-Process
- Pre-processor
- Data types

Boards used in this part
- Microcontroller board Nucleo-STM32F746
- Sensorboard: IKS01A3

First project and ST Environment
- STM32CUBE IDE- Approach
- Board Support Package (BSP)
- HAL Delay
- Frist project: Blinky in C

Enumeration types

C in C++ projects in STM32CUBEIDE
- First C++ project

Self study

Self study: Tuiton

01_1_Blinky

Create a new project: C_Blinky.

Use the HAL library to control the LED2 from the NucleoBoard every 500ms.

Note:
Search in HAL functions: Toggle and Delay
Press CTRL SPACE for autocompletion

01_2_BlinkyDelayNotBlocking

Expand project: C_Blinky.

Use the SystemTick timer to implement a software delay that does not block.

Note: HAL Delay

01_3_CPP_ Blinky

Create a new project: CPP_Blinky.

Use the HAL library to control the LED2 from the NucleoBoard every 500ms.
Implement this function in the C++ main.

Note:
Search in HAL functions: Toggle and Delay
Press CTRL SPACE for autocompletion

01_4_CPP_EnumState

Create a new project: CPP_EnumState.

Implement two state machines by implementing it in the C++ part with two enum classes.

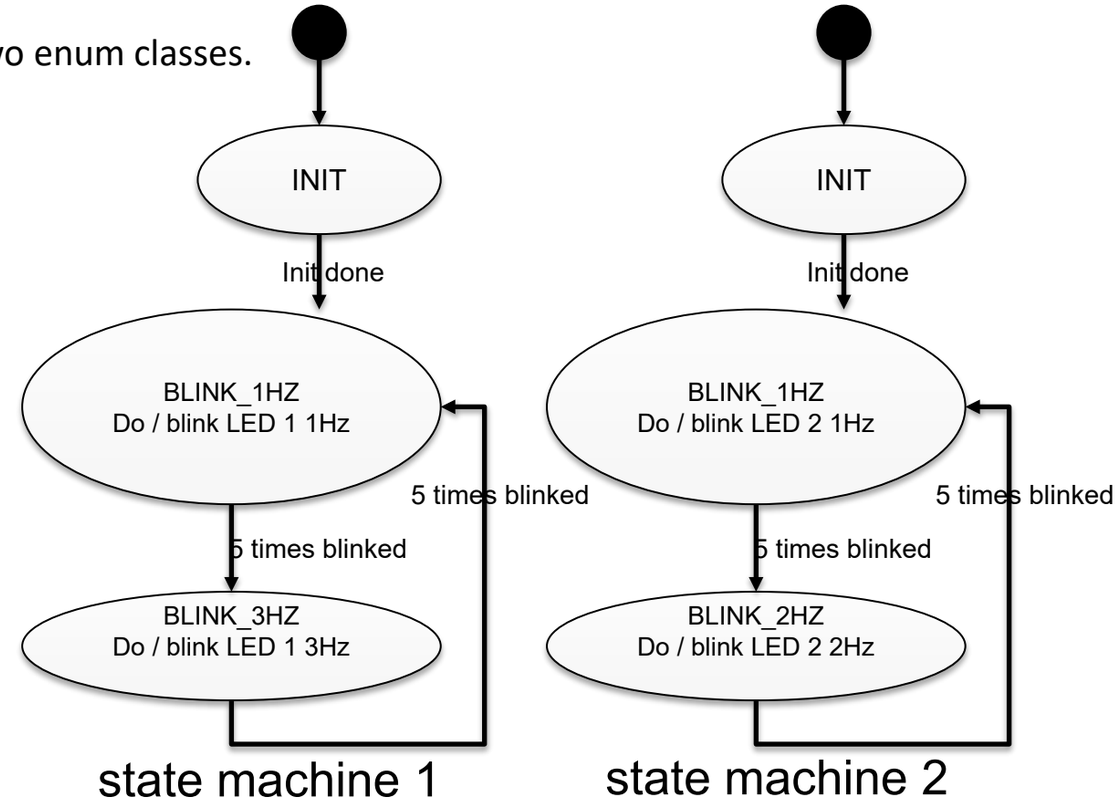
Following states should be implemented:

State machine 1 for LED 1:

- INIT (initializes the hardware)
- BLINK_1HZ (led the LED blink with 50% duty cycle at 1 Hz)
- BLINK_3HZ (led the LED blink with 50% duty cycle at 3 Hz)

State machine 2 for LED 2:

- INIT (initializes the hardware)
- BLINK_1HZ (led the LED blink with 50% duty cycle at 1 Hz)
- BLINK_2HZ (led the LED blink with 50% duty cycle at 2 Hz)



After 5 times blinking the LED the state is switched to BLINK_2HZ/BLINK3HZ and after 5 times back to BLINK_1HZ. Implement the delay non blocking.

Use the HAL library to control the LED1 and LED 2 from the NucleoBoard.

Thank you for your attention and cooperation