
NTURACING

# MCU Programming

## W1: Introduction

by 羅紀翔

2024-25 FSAE Season

# Agenda

NTURACING

| | | | |
|---|---|---|---|
| Syllabus | 1 | 2 | Introduction |
| C Compilation | 3 | 4 | C Technique 1<br>Preprocessor |
| Memory Layout | 5 | 6 | LAB 1<br>STM32 Blink |

NTURACING

# Syllabus

# Syllabus

NTURACING

## Outline

| Week | Topics | C Techniques | LAB |
|---|---|---|---|
| 1 | Introduction, C Compilation Process, Memory Layout | Preprocessor | STM32 Blink |
| 2 | Program Execution, Exceptions | Macros | Debug |
| 3 | I/O APIs, Peripherals | Built-in Functions | UART I |
| 4 | Using STM32Cube, More about C | GCC Extensions | UART II, Timer |
| 5 | Software Development, Embedded System Design | `container_of` | OOP in C |

# Syllabus

NTURACING

## References

- Textbooks
  - The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors
  - Mastering STM32 - Second Edition
- Websites
  - Code Inside Out: https://www.codeinsideout.com/
  - Memfault Interrupt: https://interrupt.memfault.com/
- Open Courseware (Chinese)
  - Computer Architecture: https://ocw.nthu.edu.tw/ocw/index.php?page=course&cid=305&
  - Operating Systems: https://ocw.nthu.edu.tw/ocw/index.php?page=course&cid=295&

# Syllabus

NTURACING

## Course Material

- Repository: https://github.com/QuantumSpawner/microcontroller_programming
  - Syllabus
  - Slides
  - LAB code
- Google Drive: https://drive.google.com/drive/folders/1oXb_RW5tBzFfpt1J1HNARS3PcakvqrsZ?usp=drive_link
  - Slides
  - Reference Books
  - Recordings

NTURACING

# Introduction to Microcontrollers


References: Ch 1, *The Definite Guide*; Ch 1.1, *Mastering STM32*

# Introduction

NTURACING

## Why Using Microcontrollers

- Simplicity
  - Easy to use
  - Simple communication protocols (c.f. computer USB, Ethernet, PCIe)
  - Self-contained (comes with built-in ROM and RAM)
- Low cost, low power
- High responsiveness
  - Microsecond response time (c.f. millisecond in typical computer)
- Abundant peripherals
  - Large number with many kinds (GPIO, UART, I2C, SPI, CAN, …)
  - Analog IO (ADC, DAC)
- Established echosystems
  - Many devices are designed to be used with microcontrollers

# Introduction

NTURACING

## Limitations of Microcontrollers

- Limited ROM and RAM
  - Typically ROM < 1M, RAM < 500K
- Low computational power
  - Typically x10 slower in single threaded and x100 slower in multithread than computer ref. https://kreier.github.io/benchmark/CoreMark/
- No MMU (memory management unit)
  - Key hardware for computer OS (Windows, Linux)
- Low expansion capabilities
  - SOC design means that nothing is upgradable
- Limited programming language support
  - Typically only in C (C++ heavily limited)

# Introduction

NTURACING

## Embedded Software Development

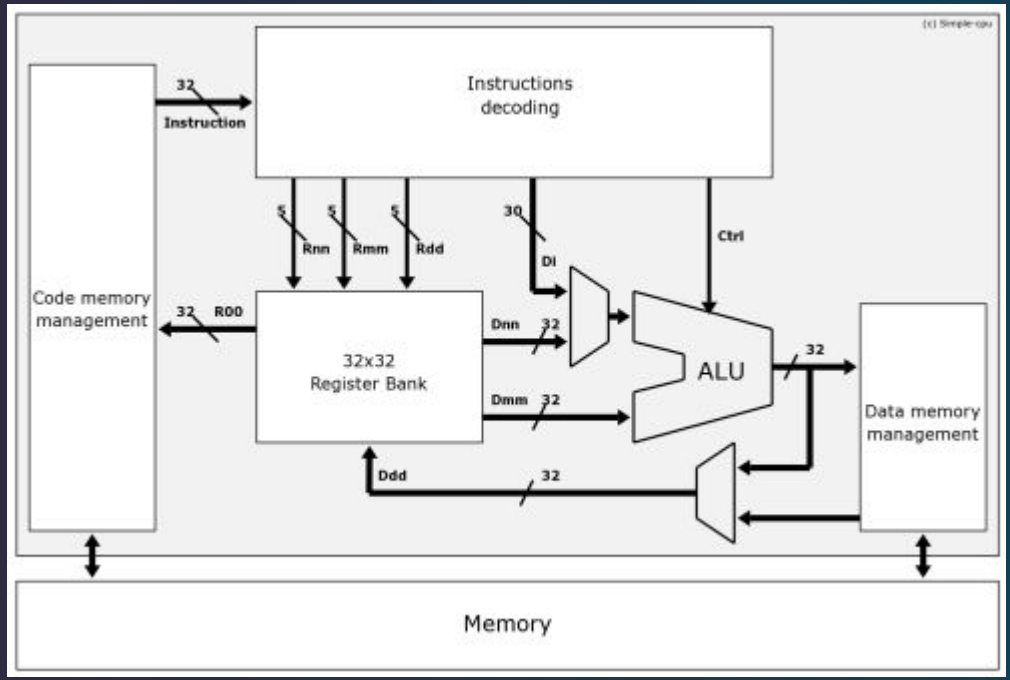
- Knowing the features and limitations of embedded platforms
- NO OS support
  - Bare metal programming -> manage everything yourself
  - Implement low level functions for C standard library (write(), sbrk())
  - Embedded OS
- Different vendor, different hardware, different API
  - Hard to migrate -> Use hardware abstraction layer (HAL)
- Difficult to test
  - Result comes as “side effects”

# Introduction

NTURACING

## Basic Knowledge

- What composes a computer?
  - Turing machine model -> CPU and memory
- What are CPU instructions?
  - Commands that CPU executes
- How a CPU works?
  - Fetch, decode, execute cycle
- What composes a CPU?
  - Control unit
  - Register file (bank)
  - Arithmetic logic unit (ALU)

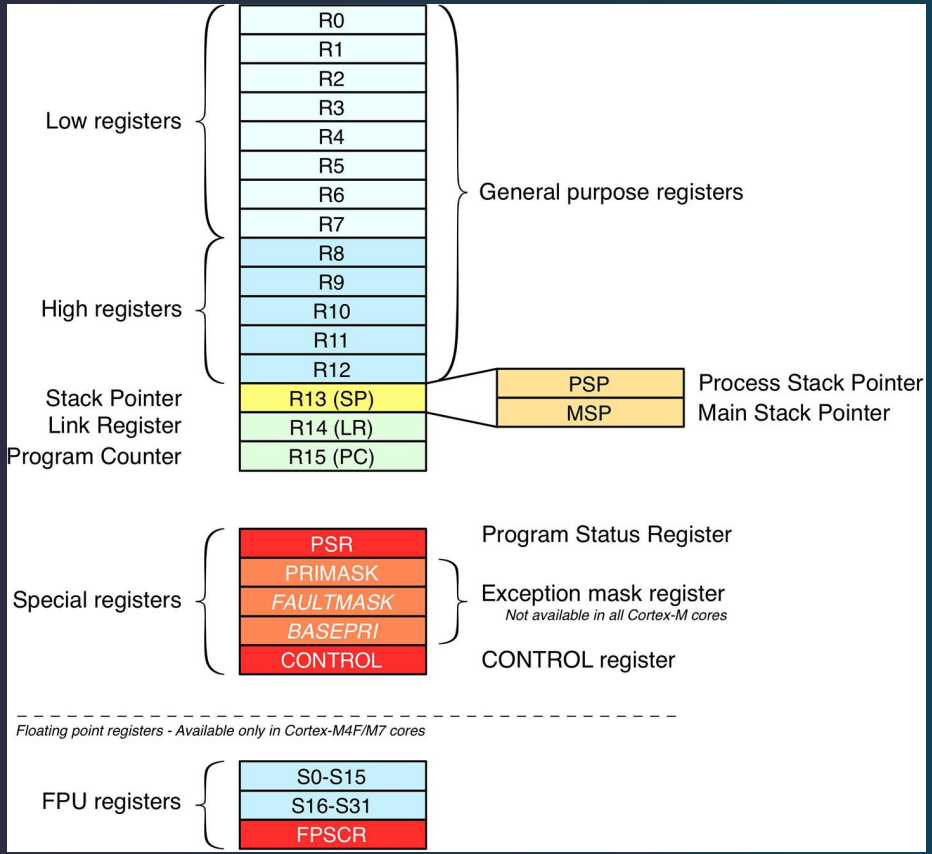


# Introduction

NTURACING

## Registers

- Cortex-M (ARM32)
- Load/store machine: only 2 types of instruction
  - Load/store
  - Operations on registers
- Store intermediate values in operations
- Store function arguments and return value

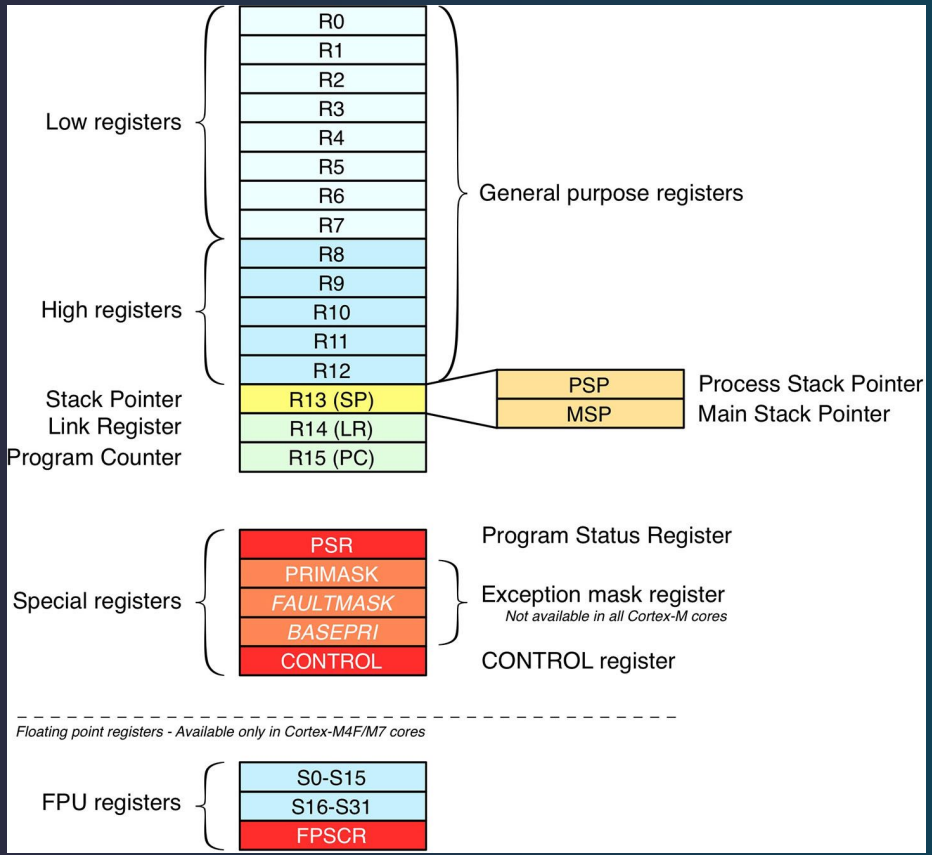


# Introduction

NTURACING

## Special Purposed Registers

- Frame pointer (FP): R7, if used (typically), stores the start of the function frame
- Stack pointer (SP): the top of the stack
  - Shadowed (Banked), usually used by embedded OS to differentiate privileged and unprivileged stacks
- Link register (LR): return address of functions
- Program counter (PC): location of the next instruction



# Introduction

NTURACING

## Exceptions

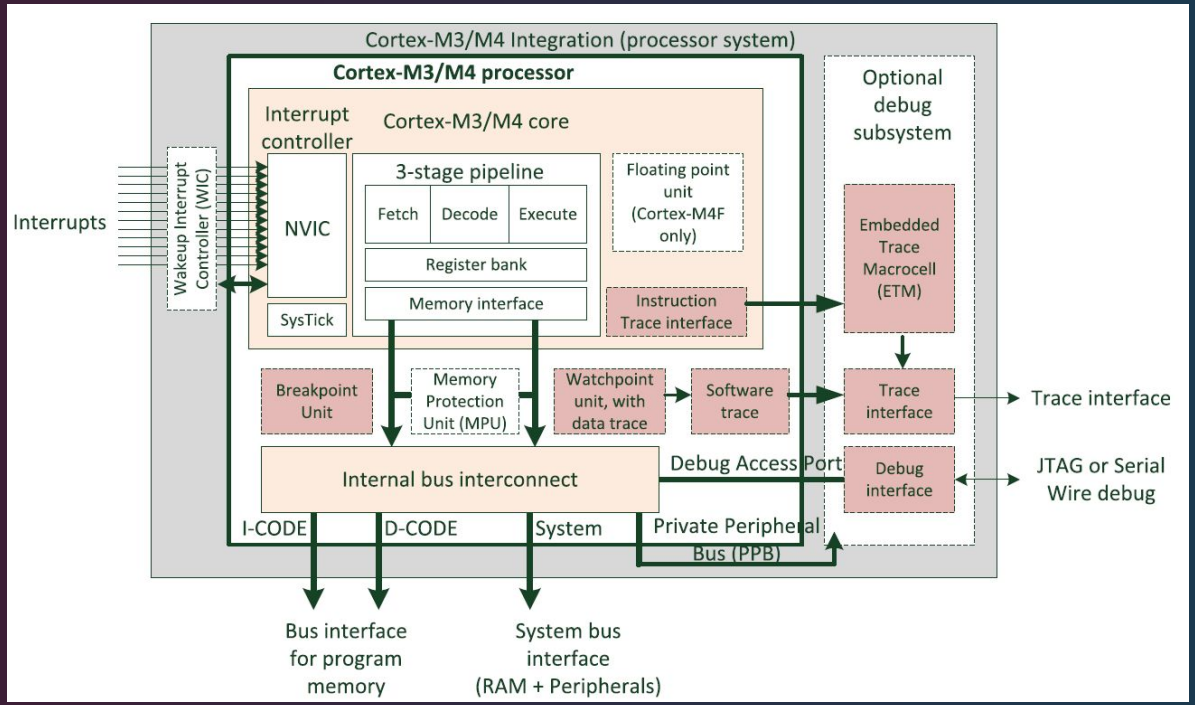
- Hardware: interrupt, software: fault
- Suspends current CPU execution and enters interrupt service routine (ISR), functions to handle exceptions
- Various types to represent different events
- Modern CPUs use isr_vector to store entries to different ISR
    - Basically an array of function pointers that point to where the ISR functions are

| Number | Exception type | Priority[a] | Function |
|---|---|---|---|
| 1 | Reset | -3 | Reset |
| 2 | NMI | -2 | Non-Maskable Interrupt |
| 3 | Hard Fault | -1 | All classes of Fault, when the fault cannot activate because of priority or the Configurable Fault handler has been disabled. |
| 4 | Memory Management[c] | Configurable[b] | MPU mismatch, including access violation and no match. This is used even if the MPU is disabled or not present. |
| 5 | Bus Fault[c] | Configurable | Pre-fetch fault, memory access fault, and other address/memory related. |
| 6 | Usage Fault[c] | Configurable | Usage fault, such as Undefined instruction executed or illegal state transition attempt. |
| 7 | SecureFault[d] | Configurable | SecureFault is available when the CPU runs in *Secure state*. It is triggered by the various security checks that are performed. For example, when jumping from Non-secure code to an address in Secure code that is not marked as a valid entry point. |
| 8-10 | - | - | RESERVED |
| 11 | SVCall | Configurable | System service call with SVC instruction. |
| 12 | Debug Monitor[c] | Configurable | Debug monitor – for software based debug. |
| 13 | - | - | RESERVED |
| 14 | PendSV | Configurable | Pending request for system service. |
| 15 | SysTick | Configurable | System tick timer has fired. |
| 16-[47/239/479][e] | IRQ | Configurable | IRQ Input |

# Introduction

NTURACING

## Cortex-M Architecture



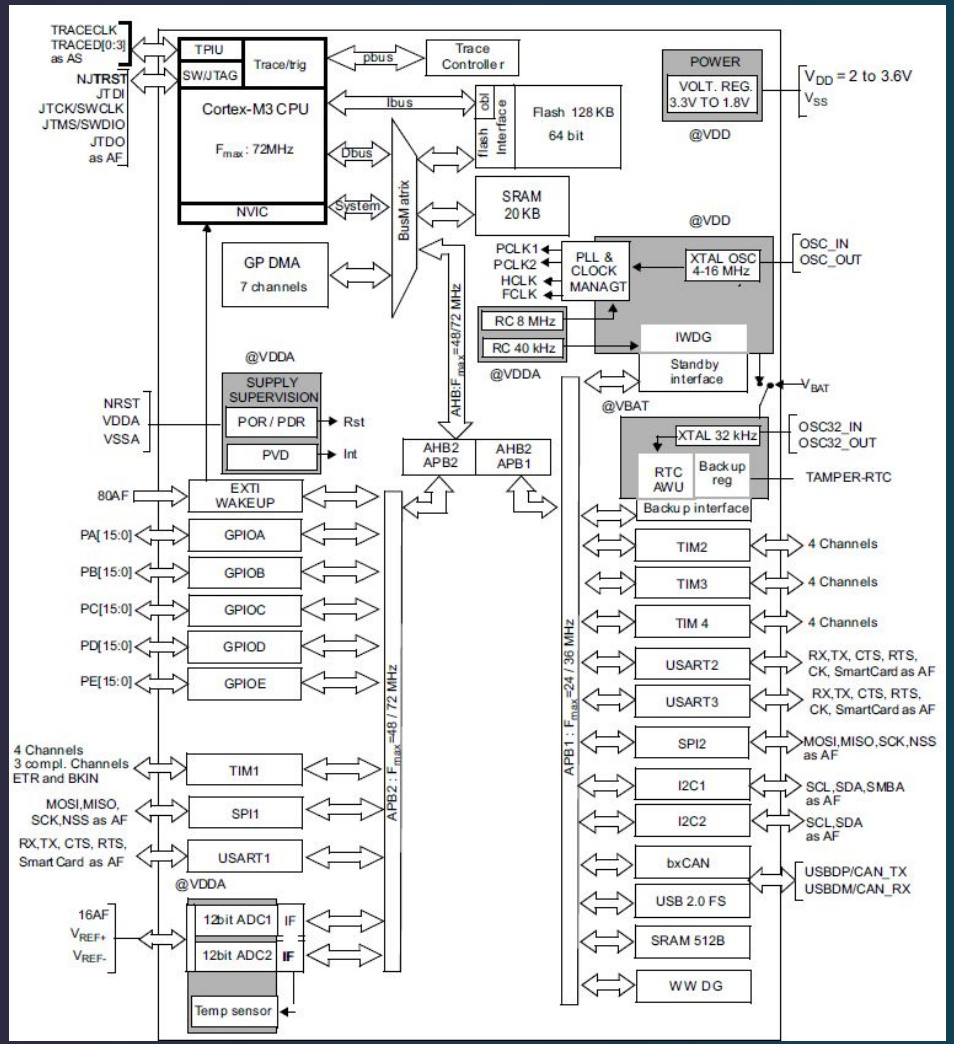
# Introduction

## STM32 Architecture

- STM32F1, form STM32F103 datasheet

## Peripherals

- Memory-Mapped I/O (**MMIO**): CPU controls peripherals via reading and writing data to specific memory address called **registers** of the peripheral
- Those I/O are carried out by buses (AXI, AHB, etc.)



# Introduction

NTURACING

## Peripherals

- Memory-Mapped I/O (**MMIO**): CPU controls peripherals via reading and writing data to specific memory address called **registers** of the peripheral
- Those I/O are carried out by buses (AXI, AHB, etc.)

### 9.2.4 Port output data register (GPIOx_ODR) (x=A..G)

Address offset: 0x0C

Reset value: 0x0000 0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ODR15 | ODR14 | ODR13 | ODR12 | ODR11 | ODR10 | ODR9 | ODR8 | ODR7 | ODR6 | ODR5 | ODR4 | ODR3 | ODR2 | ODR1 | ODR0 |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy:** Port output data (y= 0 .. 15)

These bits can be read and written by software and can be accessed in Word mode only.

NTURACING

# C Compilation Process


References: Ch 2.4, *The Definite Guide*; Compilation Process, Code Inside Out (https://www.codeinsideout.com/blog/c-cpp/compilation/)

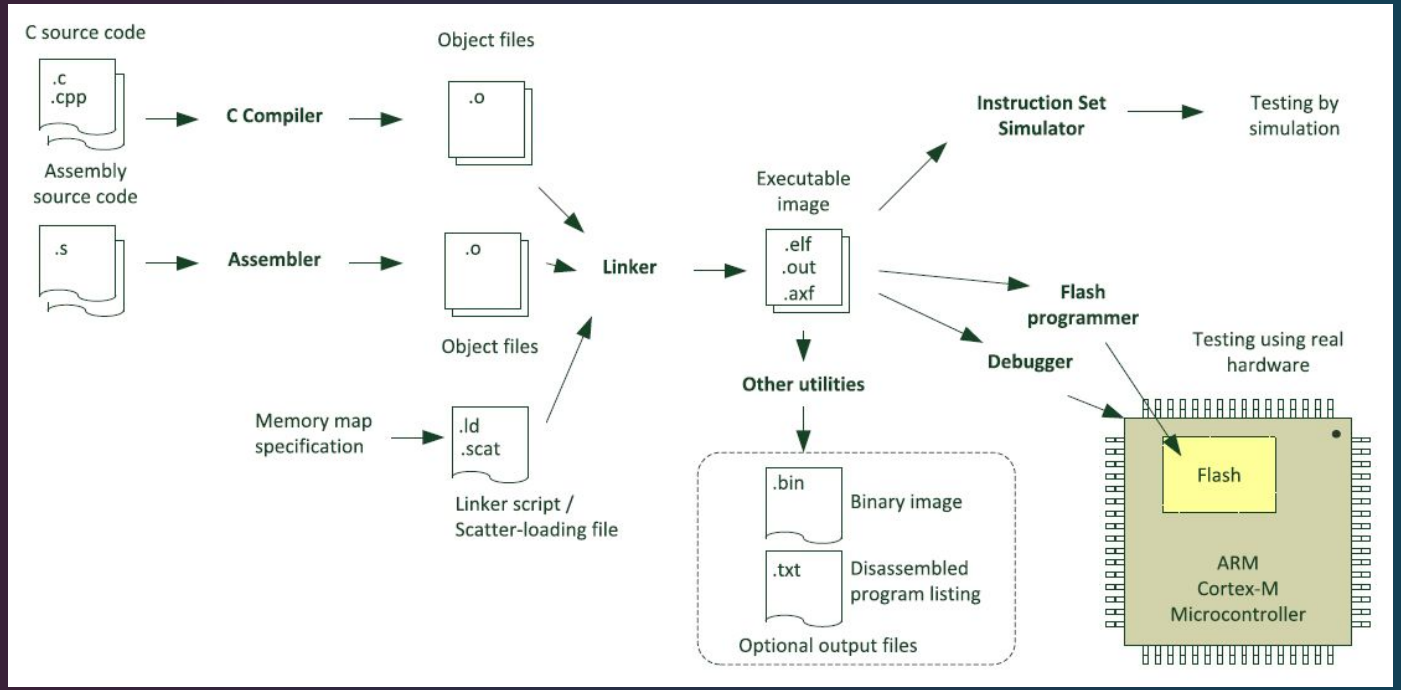
# C Compilation

NTURACING

## C Compilation

1. Preprocessing (.c -> .i)
2. Compiling
3. Assembling
4. Linking

Using "-save-temps " flag in gcc to get all intermediate files from each stage



# C Compilation

NTURACING

## Refer to Code Inside Out

- The following sections will use example from Code Inside Out (https://www.codeinsideout.com/blog/c-cpp/compilation/) to demonstrate C compilation process

# C Compilation

NTURACING

## Preprocessing

- Perform preprocessor directives
  - Things starts with "#", i.e. includes, defines, etc.
- Expand macros
- Remove comments
- Add some special markers to indicate where each line came from so that compiler could produce correct error messages

# C Compilation

NTURACING

## Compiling

- Convert source code into assembly

# C Compilation

NTURACING

## Assembling

- Convert assembly into object files in a formatted binary structure (e.g. ELF in Linux) that contains compiled machine code and a **symbol table**
- Symbols represents names and  addresses associated with functions, variables and other identifiers
- Symbol tables can be inspected by `objdump -t source.o`

# C Compilation

NTURACING

## Linking

- Generate the final executable or library
- Links all object files by replacing the references to undefined symbols with the correct address from other object files or libraries
- STM32CubeIDE produces final symbol table file “.map” in Debug/

NTURACING

# C Technique 1: Preprocessor


References: Preprocessor directives, *Visual C++* (https://learn.microsoft.com/en-us/cpp/preprocessor/preprocessor-directives?view=msvc-170)

# Preprocessor

NTURACING

## Preprocessor

- Preprocessor process "**preprocessor directives** " and expand macros, **#** (stringizing operator), and **##** (token pasting operator) (more on these in the next week's C technique)
- Preprocessor directives include:
  a. #define, #undef
  b. #include
  c. #if, #ifdef, #ifndef, #elif, #else and #endif
  d. #error
  e. #pragma (Pragmas Accepted by GCC: https://gcc.gnu.org/onlinedocs/gcc/Pragmas.html)
  f. #line
- Use examples from GeeksForGeeks (https://www.geeksforgeeks.org/cc-preprocessors/)

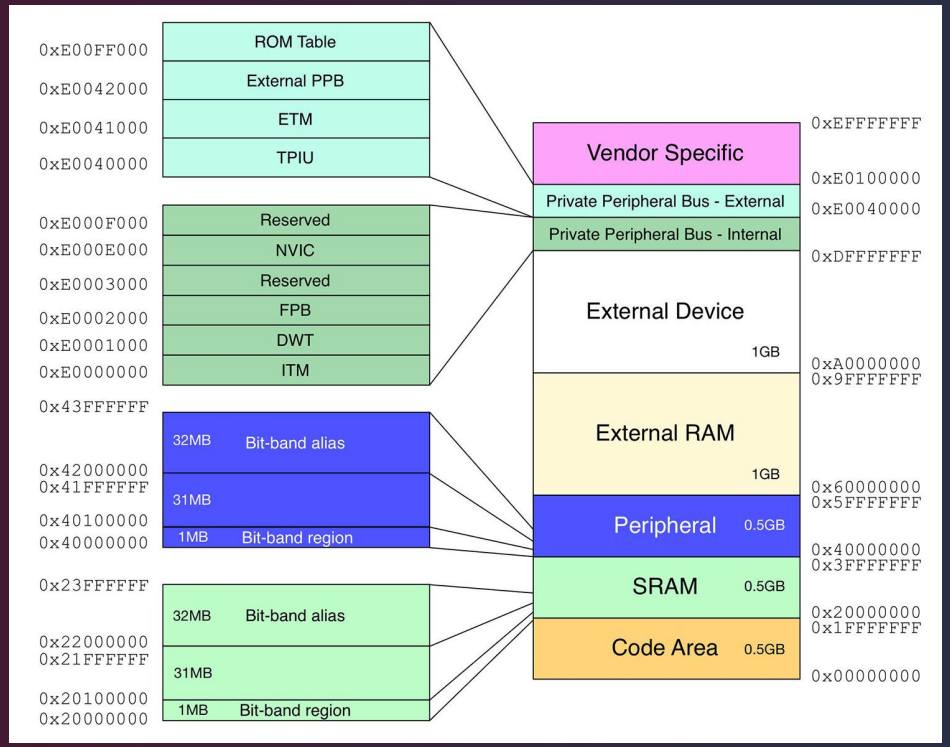
NTURACING

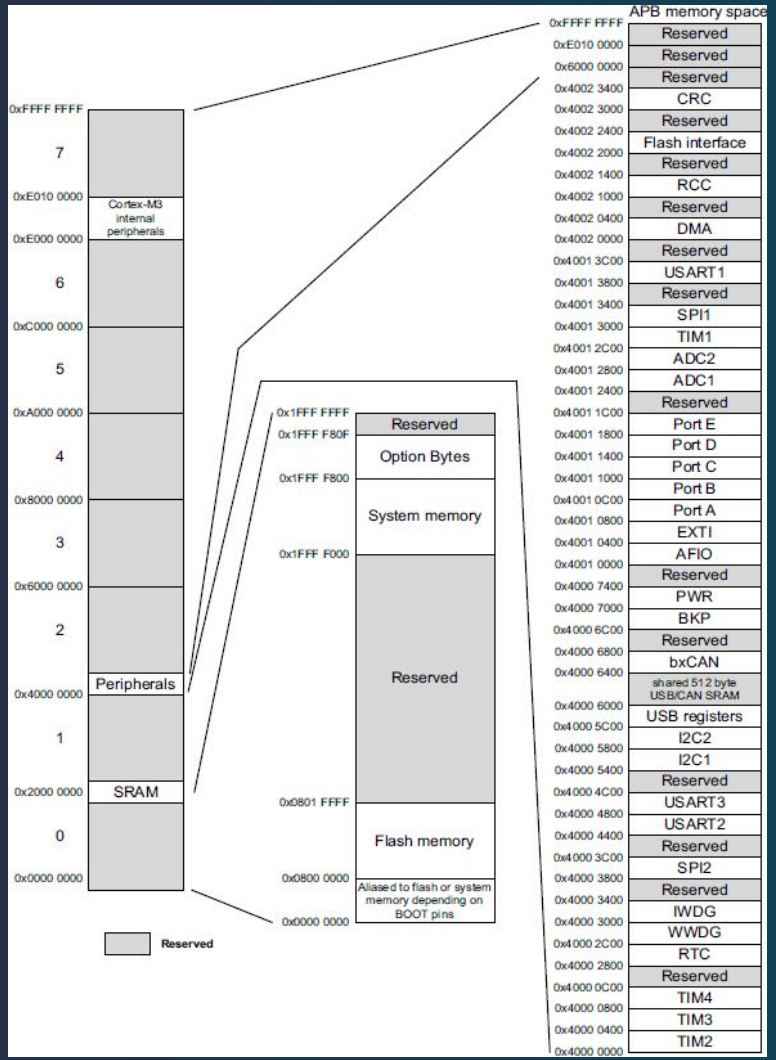
# Memory Layout


References: Ch 6, *The Definite Guide*; Ch 20, *Mastering STM32*

# Memory

## Memory Map (STMF103)



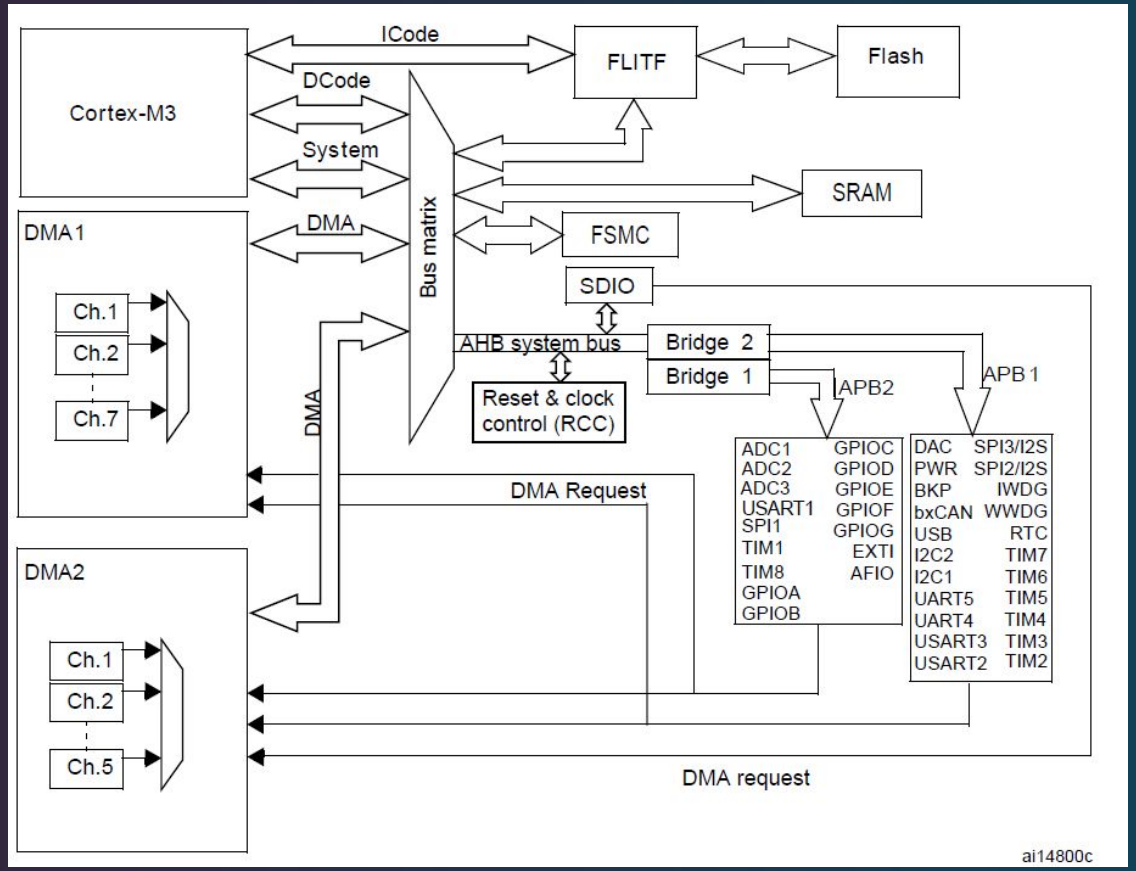



# Memory

NTURACING

## STM32 Bus Architecture

- STM32F1
  - STM32F103 reference manual
- **Modified Harvard** architecture
  - Separate instruction and data bus
  - Unified memory address (hence “modified”)
  - C.f. **Von Neumann** (same bus for I and D)
  - Lower latency

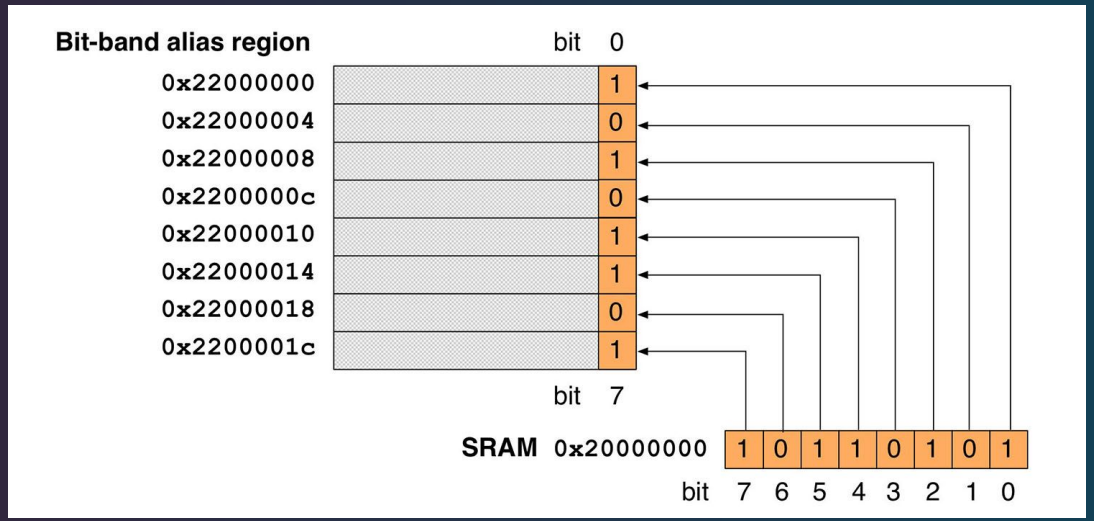


# Memory

NTURACING

## Special Function Memory: Bit-Banding

- Map each bit of a given area of memory to a whole world in the bit-banding aliased region, allowing atomic access to such bit.
- Primarily used for peripherals
- Other special function memory
  - CIM (compute in memory)
  - Does addition/ multiplication

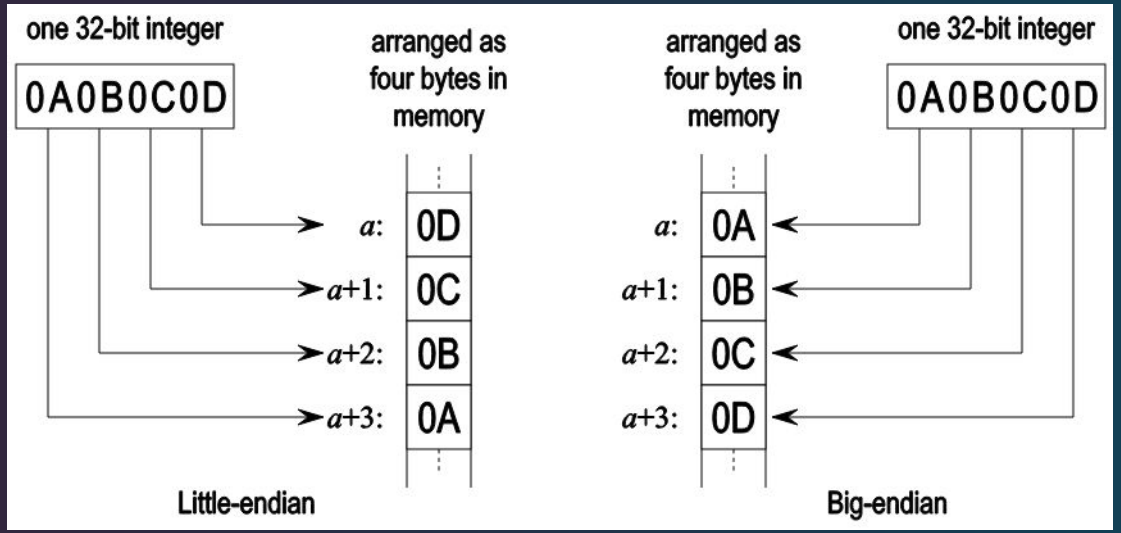


# Memory

NTURACING

## Little vs. Big Endiness

- Smallest byte stored in the **smaller** or **bigger** address
- Processor
  - Intel: little
  - Motorola: big
  - ARM: selectable (typically little)
- Communication protocols
  - Network, I2C, SPI: big
  - UART: implementation dependant
- Be warily when programming!

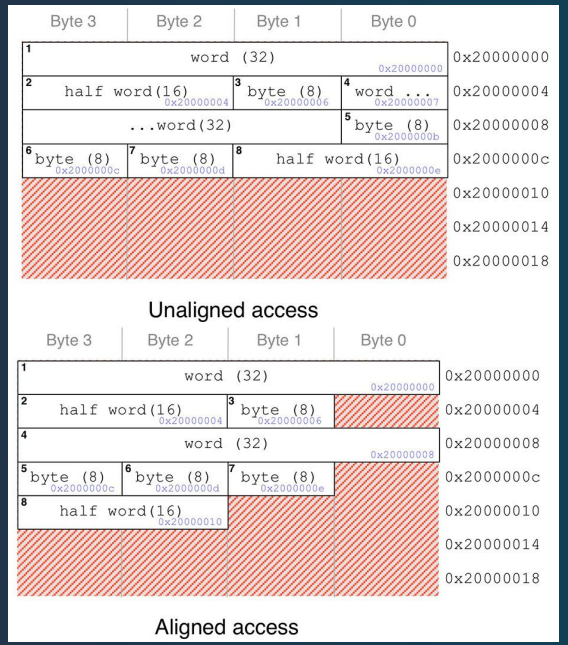


# Memory

NTURACING

## Memory Alignment

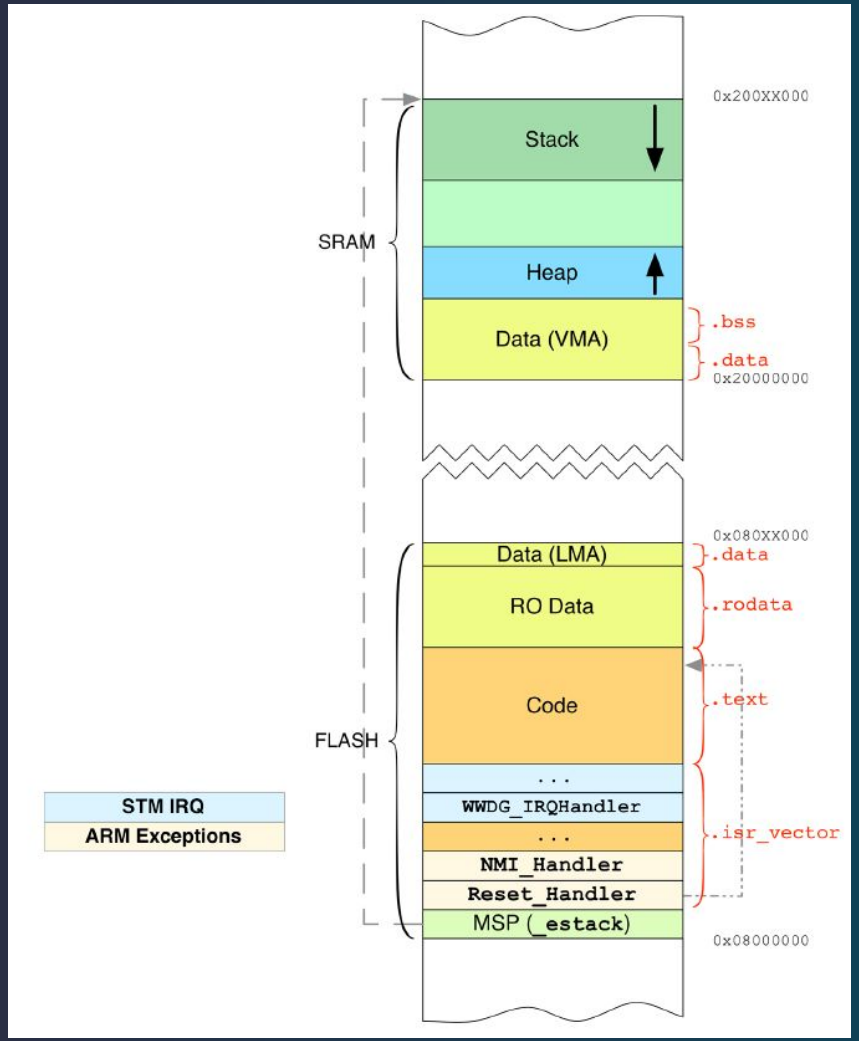
- **Word** = length of address, i.e. a word = 4 bytes in 32-bit machines, half word = 2 bytes, etc.
- **Aligned** : data of 4 bytes must be at address divisible by 4. etc.
- Unaligned memory access are not allowed (Cortex-M0/1) or have performance penalty (Cortex-M3/M4/M7)
- Instructions and stack are also required to be aligned
- Compiler uses **padding** to keep members of structs aligned, structs are also aligned to word boundaries
  - i.e. struct s {bool b; int i;}; sizeof(struct s) == 8!!
- Compiler also enforce other alignment requirements but be warily when doing something low level e.g. allocating memory



# Memory

## Program in Memory (only applies to MCUs)

- Memory sections
  - .text : instructions
  - .data : initialized data
  - .bss (blocks started by symbols): uninitialized data
  - .rodata : const data and strings
- .isr_vector :
  - Entry point for exceptions
  - After reset, MSP (0x00000000) is loaded to SP, Reset_Handler (0x00000004) is loaded to PC to determine the initial stack and first instruction to execute



# Memory

NTURACING

## Program in Memory (cont'd)

- During boot:
  - **Data initialization** : .data is copied from flash to SRAM
  - **Zero initialization** : .bss is set to all zeros
- Required by C, done by startup code (in assembly)

| Language structure | Binary file section | Memory region at run-time |
|---|---|---|
| Global un-initialized variables | `.common` | Data (SRAM) |
| Global initialized variables | `.data` | Data (SRAM+Flash) |
| Global `static` un-initialized variables | `.bss` | Data (SRAM) |
| Global `static` initialized variables | `.data` | Data (SRAM+Flash) |
| Local variables | <no specific section> | Stack or Heap (SRAM) |
| Local `static` un-initialized variables | `.bss` | Data (SRAM) |
| Local `static` initialized variables | `.data` | Data (SRAM+Flash) |
| Const data types | `.rodata` | Code (Flash) |
| Const strings | `.rodata.1` | Code (Flash) |
| Routines | `.text` | Code (Flash) |

# Memory

NTURACING

## Linker script (.ld)

- Tells the linker about the memory layout of the system
  - Assign each symbol accordingly
- Reffer to *_FLASH.ld in STM32CubeIDE
- Defines symbols for startup code
  - _sdata, _edata: Marks the range of .data section in SRAM

NTURACING

# LAB 1: STM32

# Blink

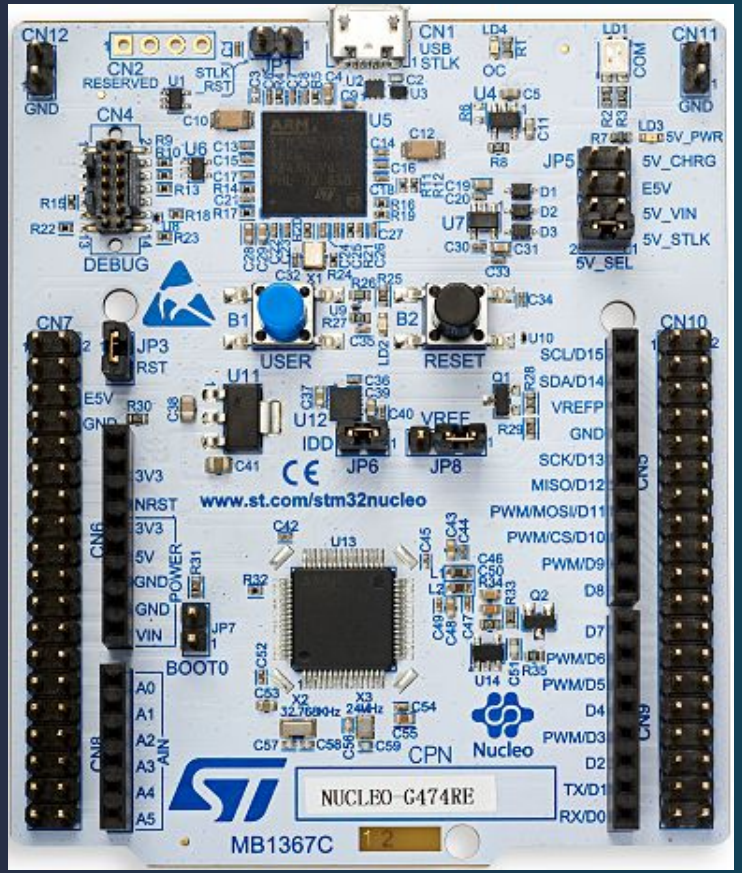

References: Ch 2, *Mastering STM32*

# Blink

NTURACING

## LAB Hardware

- NUCLEO-G474RE
  - STM32G474RE based on Cotrex-M4
  - Built-in debugger
- USB micro to type A cable

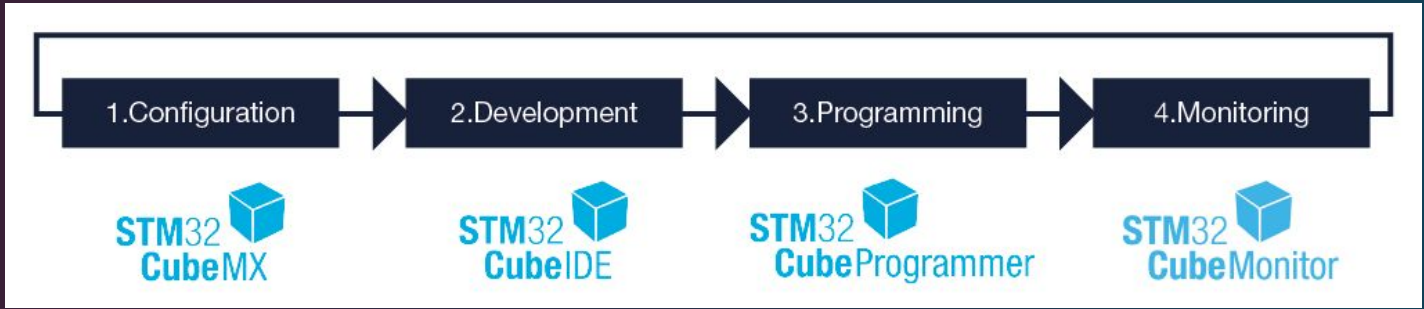


# Blink

NTURACING

## Download STM32CubeIDE

- Link: https://www.st.com/en/development-tools/stm32cubeide.html
  - You have to register to ST account
- ST's toolchain
  - CubeMX
  - CubeIDE
  - CubeProgrammer
  - CubdMonitor



# Blink

NTURACING

## Blink

- Configure via CubeMX
  - Ref. STM32 Guide #2: Registers + HAL (Blink example) by Mitch Davis: https://www.youtube.com/watch?v=Hffw-m9fuxc
  - Save .ioc to generate code
- Add blink to main loop

```
HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
HAL_Delay(500);
```

- Compile and flash
- Blink!