Tool-chain and documents for developing on STM32 MCUs

Install a tool-chain including IDE, programmer, debugger for developing on STM32 ARM Cortex-M. List of necessary documents related to hardware and software. How to read and extract information from documents.

#arm #stm32 #toolchain

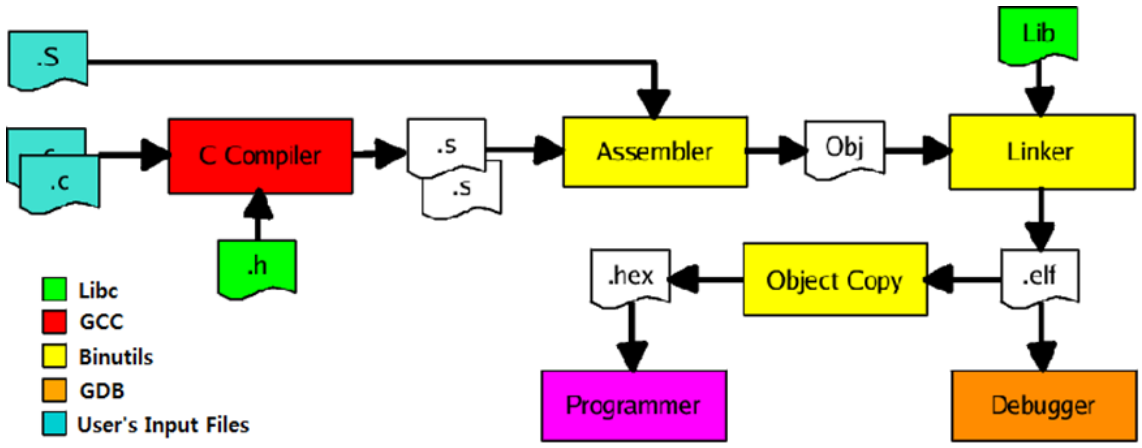
Last update: April 28, 2021

Table of Content

1. Toolchain

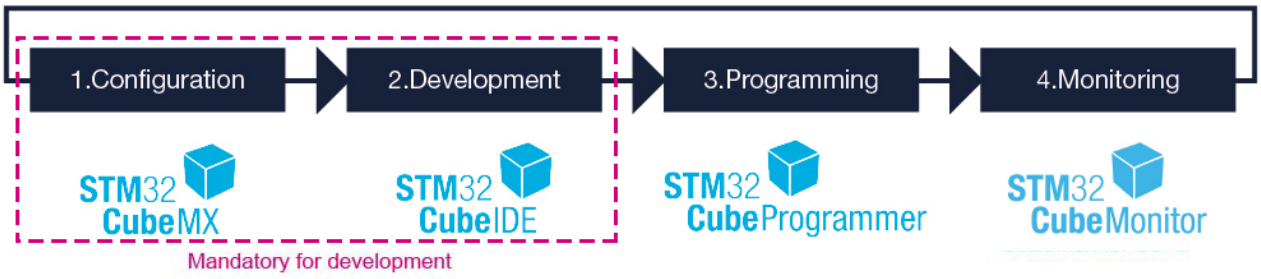
A tool-chain is a set of programs, compilers, and tools that allows developers to:

1. Write code and navigate inside source files of the project
2. Navigate inside the application code, allowing developers to inspect variables, function definitions / declarations, and so on
3. Compile the source code using a cross-platform compiler
4. Upload the executable application on the target board
5. Monitor, inspect and debug the application on the board



Example of a toolchain

STMicroelectronics offers a complete tool-chain for STM32 MCUs as below:



STM32 Toolchain

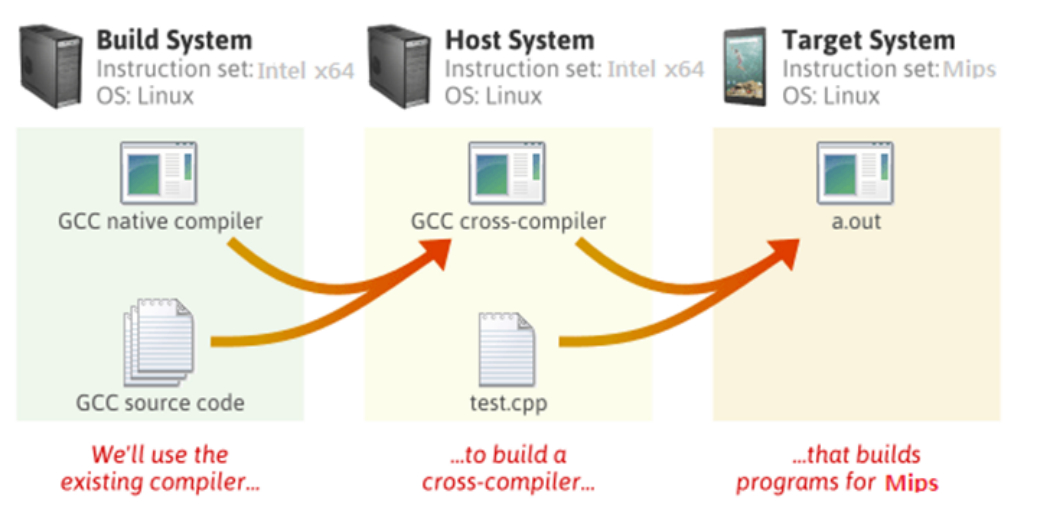
- STM32CubeMX is used for Device Configuration and Code Generation
- STM32CubeIDE is fully integrated IDE which includes:
    - Eclipse IDE - an open source code editor and manager which supports many plugins such as C/C++ Development Platform, GCC Cross Compiler, GDB Hardware Debugger, Make and build scripts

  - GNU ARM Cross-compiler with ST patch for STM32 MCUs - a compiler that converts code to executable and linkable file (.elf) or binary file (.bin, .hex)
  - GDB for inspecting, debugging the running application
- STM32CubeProg is a programmer that downloads the *.elf* or *.bin* file to the target MCU's flash. It has drivers to communicate with the MCUs:
  - ST-LINK GDB or ST-LINK openOCD debugger to probe the target MCUs
- STM32 helps to fine-tune and diagnose STM32 applications at run-time by reading and visualizing their variables in real-time

Cross-compiler

A *compiler* is just a *language translator* from a given programming language (e.g. C/C++) to a low-level machine language, also known as *assembly*.

A *cross-platform compiler* is a compiler that is able to generate machine code for a hardware machine *different from the one which is being used to develop the applications*. In this case, the GCC ARM Embedded compiler generates machine code for Cortex-M processors while compiling on an x86 machine with a given OS (e.g. Windows or Mac OSX).



Example of Cross-Compiler for MIPS on Linux

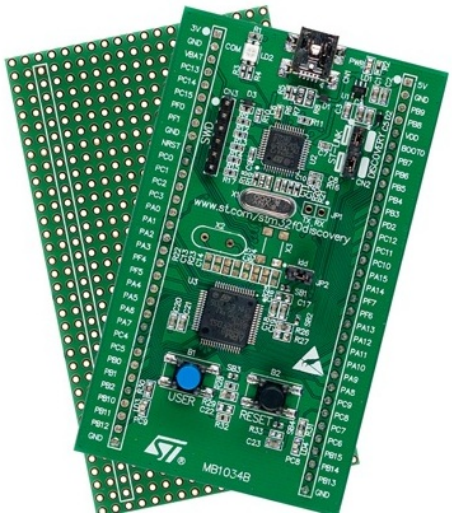
2. Target board

Example Target board

This post uses *STM32F0 Discovery* kit as an example board to practice. There are many Discovery, Evaluation or Nucleo boards sharing the same knowledge to work on them. Please pay attention to check the pin names, the pin assignments on the target processor.

The STM32F0 Discovery board is a low-cost and easy-to-use development kit to quickly evaluate and start development with an STM32 F0 series microcontroller. It offers the following features:

- STM32F051R8T6 *Cortex®-M0* microcontroller featuring 64 KB Flash memory, 8 KB RAM in an LQFP64 package
- On-board *ST-LINK/V2* with selection mode switch to use the kit as a standalone ST-LINK/V2 (with SWD connector for programming and debugging)
- Board power supply: through USB bus or from an external 5V supply voltage
- Four LEDs:
  - LD1 (red) for 3.3 V power on
  - LD2 (red/green) for USB communication
  - LD3 (green) for PC9 output
  - LD4 (blue) for PC8 output
- Two push buttons (user and reset)



STM32F0 Discovery Board

The Cortex-M0 processor implements the ARMv6-M architecture, which is based on the 16-bit Thumb® instruction set and includes Thumb-2 technology. Read about ARM.

STM32F051 has 32-bit low- and medium-density advanced ARM™ MCU with a high-performance ARM Cortex™-M0 32-bit RISC core which has 64 Kbytes Flash, 8 Kbytes RAM, RTC, timers, ADC, DAC, comparators and communication interfaces:

- Core and operating conditions
  - ARM® Cortex™-M0 0.9 DMIPS/MHz up to 48 MHz
  - 1.8/2.0 to 3.6 V supply range
- High-performance connectivity
  - 6 Mbit/s USART
  - 18 Mbit/s SPI with 4- to 16-bit data frame
  - 1 Mbit/s I²C fast-mode plus
  - HDMI CEC
- Enhanced control
  - 1x 16-bit 3-phase PWM motor control timer
  - 5x 16-bit PWM timers
  - 1x 16-bit basic timer
  - 1x 32-bit PWM timer
  - 12 MHz I/O toggling

Integrated debugger

The *ST-LINK/V2* programming and debugging tool is integrated on the STM32F0 Discovery board. Do not need any external ST-LINK debugger. Make sure the *CN2* jumpers are connected to debug the on-board processor.

3. STM32CubeIDE

- Based on Eclipse®/CDT, with support for Eclipse® add-ons, GNU C/C++ for Arm® toolchain and GDB debugger
- Integration of services from STM32CubeMX:
  - STM32 microcontroller, microprocessor, development platform and example project selection
  - Pinout, clock, peripheral, and middleware configuration
  - Project creation and generation of the initialization code
  - Software and middleware completed with enhanced STM32Cube Expansion Packages
- Additional advanced debug features including:

    - CPU core, peripheral register, and memory views
    - Live variable watch view
    - System analysis and real-time tracing (SWV)
    - CPU fault analysis tool
    - RTOS-aware debug support including Azure® RTOS ThreadX and FreeRTOS™ Kernel
- Support for ST-LINK (STMicroelectronics) and J-Link (SEGGER) debug probes

IDEs that support ARM Cortex:

- *ST®, STM32CubeIDE®: free, native support, highly integrate with MX tools*
- *ARM®, Atollic TrueSTUDIO®: was bought by ST, included in STM32CubeIDE*
- *Keil™, MDK-ARM™: only free for STM32F0 and STM32L0 processes*
- *Altium®, TASKING™ VX-toolset: paid license*
- *IAR™, EWARM (IAR Embedded Workbench®): paid license*

3.1. Installation

Download STM32CubeIDE

Need to register a free account to download from ST website. Just need email to verify.

During the installation, please check to install ST-LINK and SEGGER J-Link drivers.

3.2. Create a workspace

When start the program, it will ask to select a directory as workspace - the location to save projects. Consider to make new workspace for different big projects.

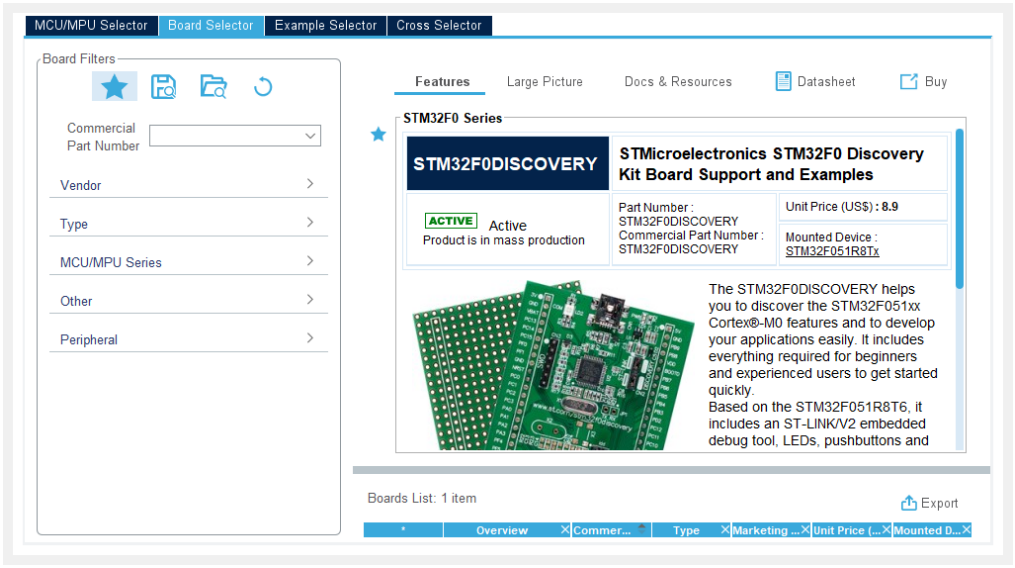
3.3. Create a project

It is recommended to start a new project with STM32CubeIDE as it will automatically configure the project for the selected target processor. STM32CubeIDE includes STM32CubeMX which can generate code from Device Configuration settings. The output C code project is compliant with IAR™, Keil® and STM32CubeIDE (GCC compilers) for Arm®Cortex®-M core.

UM1718 STM32CubeMX User Manual

3.3.1. Select target MCU

When start a new STM32 project, IDE shows up the *Target Selection* screen first. There are options to select the target MCU/MPU by name, board, example, and cross-reference.



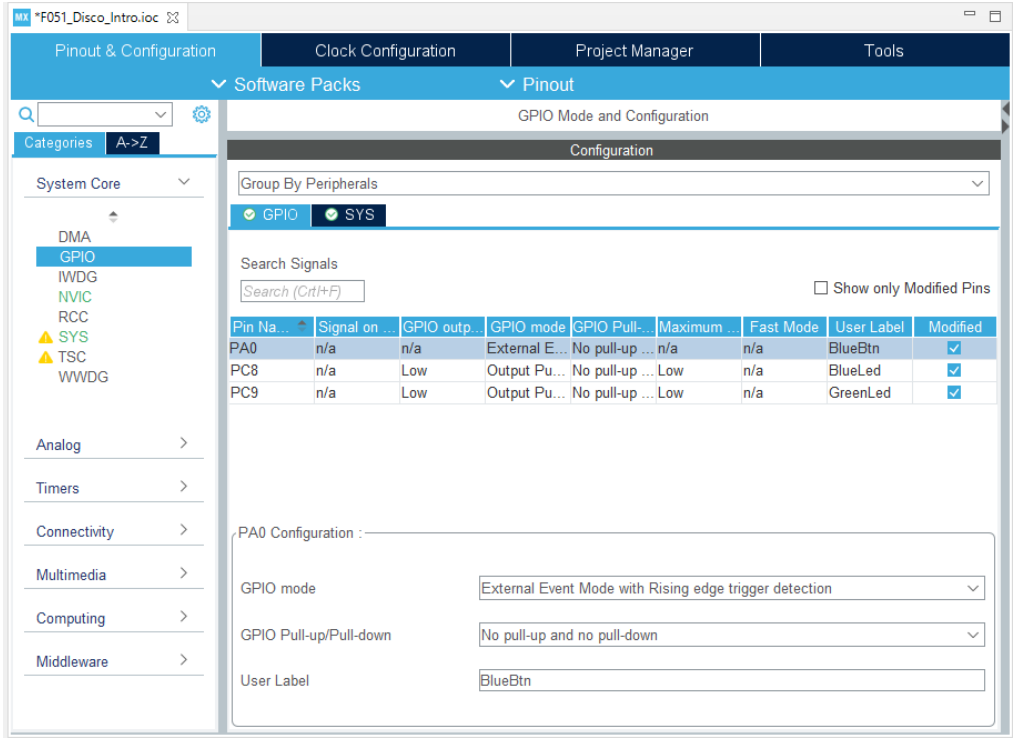
Select STM32F0 Discovery board

IDE will run a screen named *Device Configuration Tool*, in there, it's easy to enable any supported features in graphical mode. If the selected target is a development board, this tool will ask to use a default system config for the target board - usually including ST-Link pins, on-board buttons, LEDs, USB connect.

Device Configuration Tool screen

3.3.2. Pinout Config

Using tab *Pinout & Configuration* to enable supported features on the target processor.



Configure peripheral and pin assignment

Quick pin assignment

- Do a Left click on a pin and select the pin function
- Do a Right click to assign a custom name for that pin.

3.3.3. Clock Config

Next tab is *Clock Configuration* where to adjust the clock frequency, clock paths in the processor. Notice the PLL and System Clock Mux which control `SYSCLK`, Bus clocks, and peripheral clock (Timers, I2C, USART, etc.).

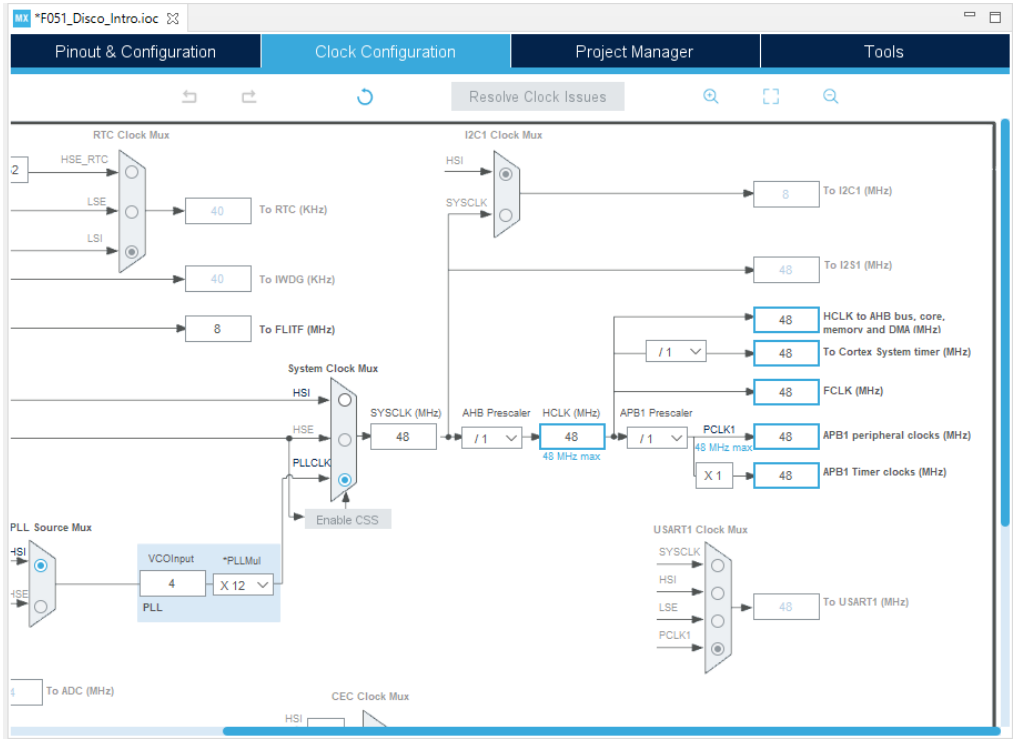
There are some registers to fill in pre-scaler values which will change the clock frequency of the Bus and peripherals.

3.3.4. Project settings

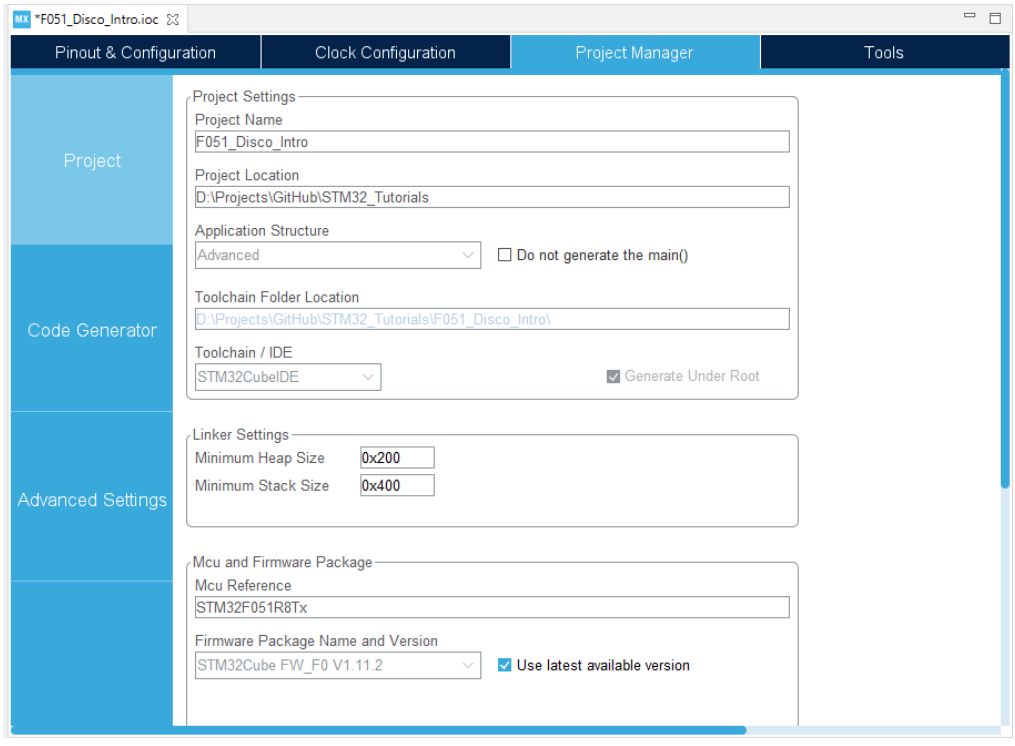
In the tab *Project Manager*, there are settings for specific features:

- Linker settings for Heap

- Firmware version
- Code generator
- Driver function type



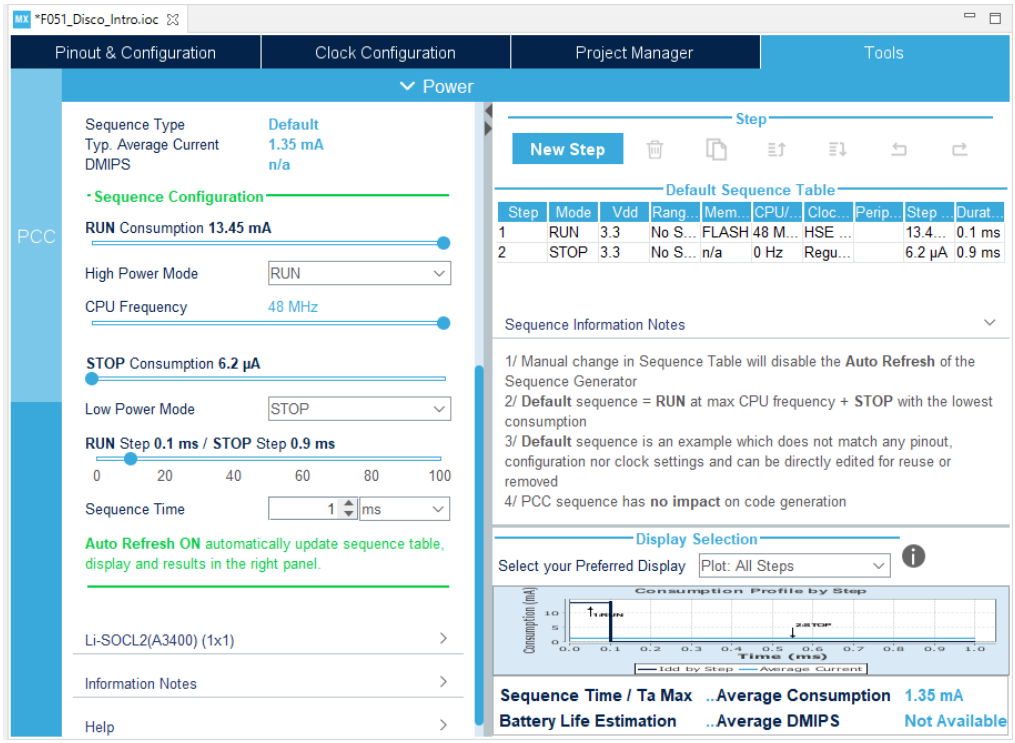
Configure clock paths and frequency



Configure settings for project

3.3.5. Power Tools

There is a tool called Power Consumption Calculator (PCC) to help defining power modes by selecting which peripheral is enabled or disabled, clock frequency, and clock path for each mode.



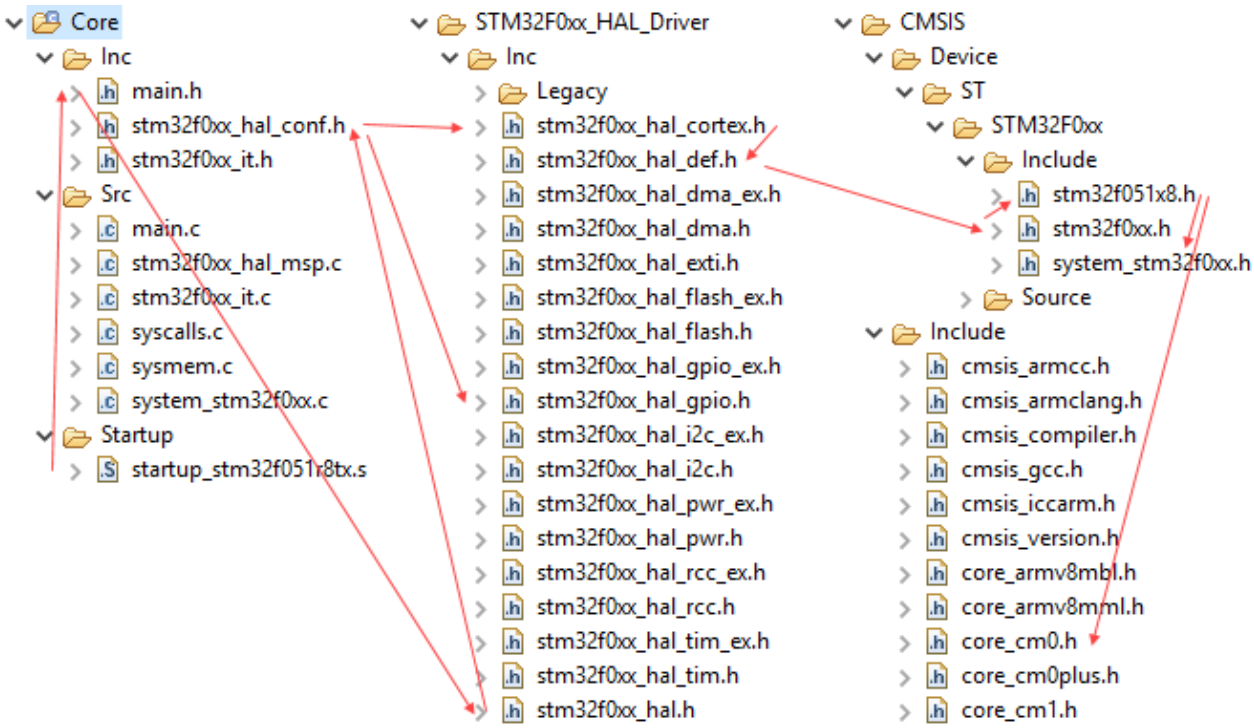
Configure power modes

3.4. Generate Code

After configuring pins, save the settings first and then start generating code. Manually request to generate code by pressing `alt` + `k` or choosing menu **Project** >> **Generate Code**.

When chosen to use a Firmware Library in the project, IDE automatically uses ST Hardware Abstract Layer (HAL) library as the main way of controlling the processor. HAL also makes use of Cortex Microcontroller Software Interface Standard (CMSIS) library to access processor's registers.

Code dependency starts from the `main.h` source file. This file includes HAL files which eventually includes CMSIS files.



Code dependency

A HAL driver includes the following set of files:

File	Description
`stm32f0xx_hal.h/.c`	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on SysTick APIs. This also include `stm32f0xx_hal_def.h`.
`stm32f0xx_hal_def.h`	Common HAL resources such as common define statements, enumerations, structures and macros. This includes CMSIS headers.
`stm32f0xx_hal_ppp.h/.c`	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices, example: `stm32f0xx_hal_adc.c`, `stm32f0xx_hal_irda.c`
`stm32f0xx_hal_ppp_ex.h/.c`	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in different way, for example: `stm32f0xx_hal_adc_ex.c`, `stm32f0xx_hal_flash_ex.c`.

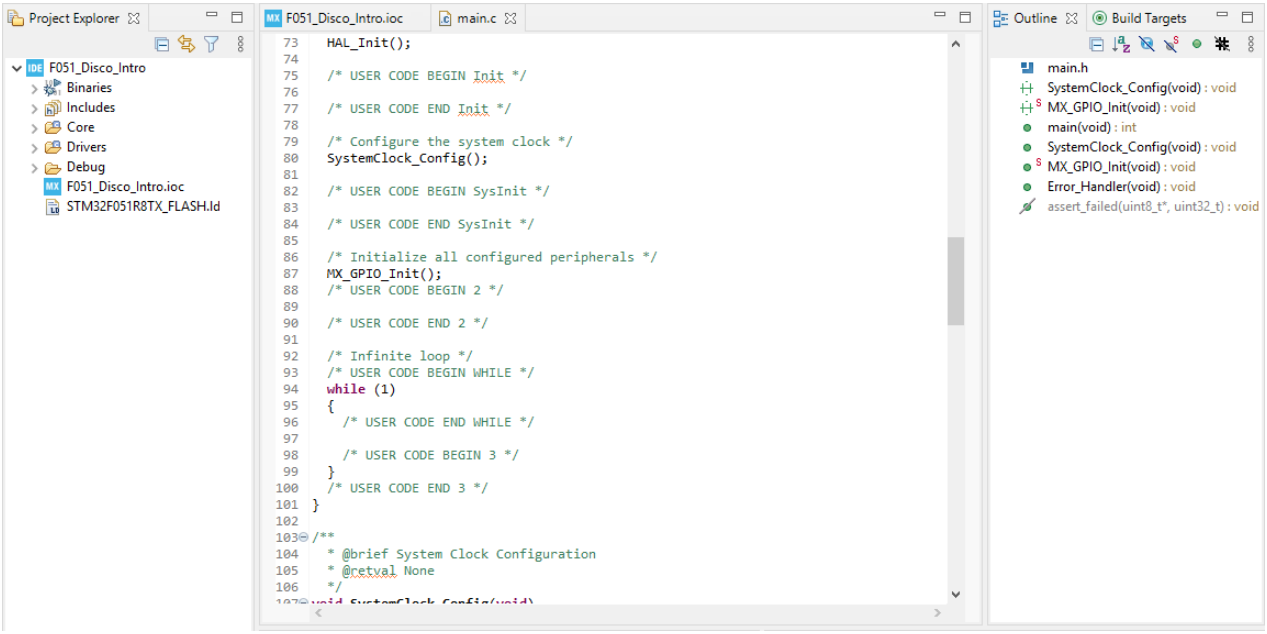
The minimum files required to build an application using the HAL are listed in the table below:

File	Description
`startup_stm32f0xx.s`	Toolchain specific file that contains reset handler and exception vectors. For some toolchains, it allows adapting the stack/heap size to fit the application requirements
`system_stm32f0xx.c`	This file contains `SystemInit()` which is called at startup just after reset and before branching to the main program. It does not configure the system clock at startup (contrary to the standard library).This is to be done using the HAL APIs in the user files. It allows relocating the vector table in internal SRAM.
`stm32f0xx_hal_conf.h`	This file allows the user to customize the HAL drivers for a specific application. It is not mandatory to modify this configuration. The application can use the default configuration without any modification. This call to STM32F0 HAL headers.
`stm32f0xx_hal_msp.c`	This file contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
`stm32f0xx_it.h/.c`	This file contains the exceptions handler and peripherals interrupt service routine, and calls `HAL_IncTick()` at regular time intervals to increment a local variable (declared in `stm32f0xx_hal.c`) used as HAL timebase. By default, this function is called each *1ms* in Systick ISR.The `PPP_IRQHandler()` routine must call `HAL_PPP_IRQHandler()` if an interrupt based process is used within the application.
`main.h/.c`	This file contains the main program routine, mainly: • call to `HAL_Init()` • have `assert_failed()` implementation • set system clock configuration • declare peripheral HAL initialization • user application code.

3.5. Add user code

User code sections are marked with a pair of phrases `/* USER CODE BEGIN x */` and `/* USER CODE END x */`. User code inside those marks are kept remaining during code generation.

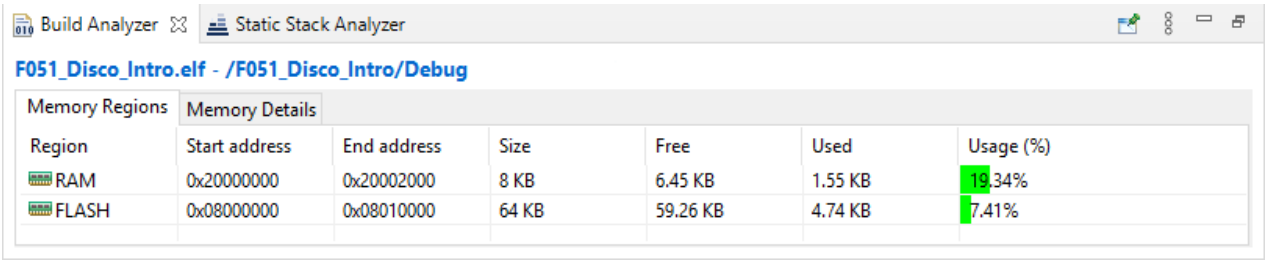
```
/* Private includes */
/* USER CODE BEGIN Includes */
    <code goes here>
/* USER CODE END Includes */

/* Private define */
/* USER CODE BEGIN PD */
    <code goes here>
/* USER CODE END PD */
```



STM32CubeIDE based on Eclipse®/CDT

3.6. Build project

Build the application by pressing `ctrl` + `b`, or in menu **Project** >> **Build All**. There are some reports about the resource usage to check after compilation. The first thing to check is the memory usage, in term of RAM and FLASH free space.



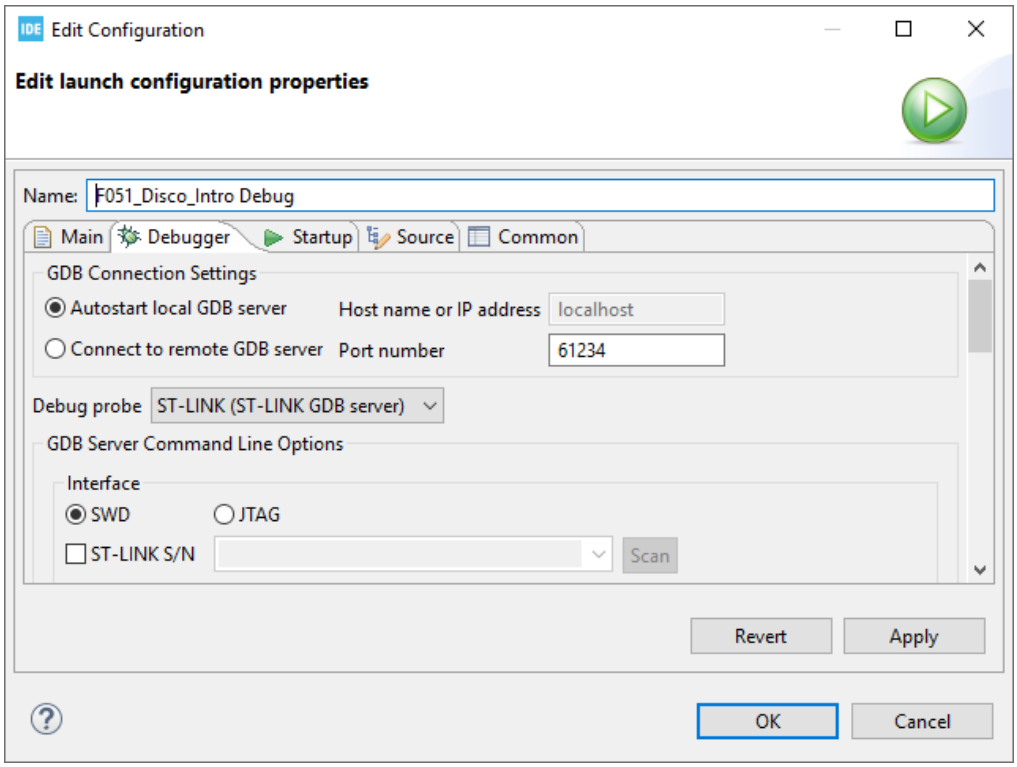
Memory usage in Build report

3.7. Run on board

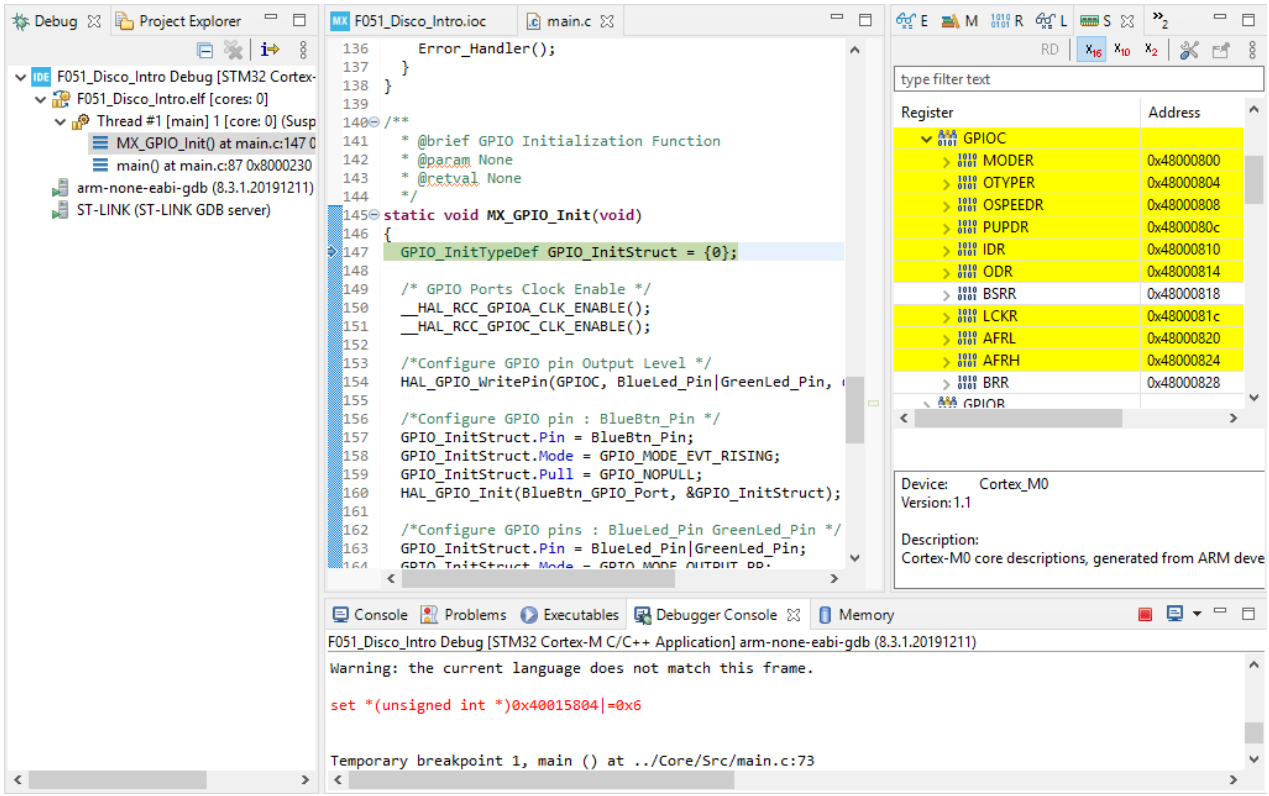
If code is compiled successfully, hit menu **Run** >> **Run** to setup a run configuration and download executable file to board. Leave every settings as default at this time, then check the debugging probe is set to ST-LINK by default.

3.8. Debug on board

The IDE has debugging function and can be invoked by `F11` key. It will set a break point at the `main()` function by default. There are some views to help debugging easier.



Edit run config on target board

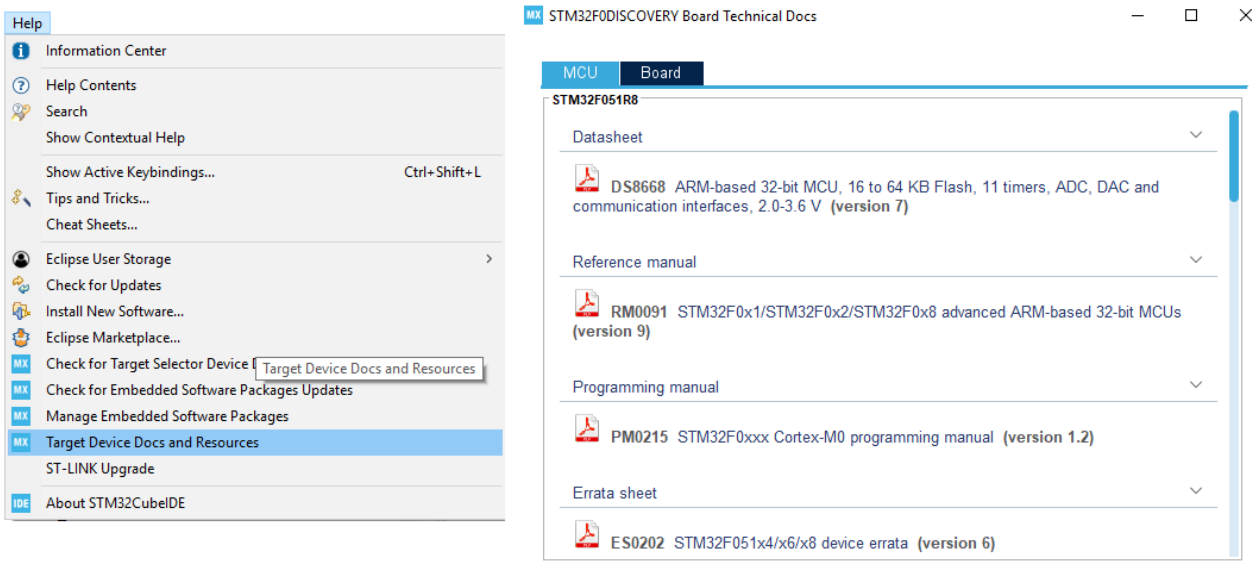


Debugging screen

4. Documents

In embedded programming, documents have a very important role as they are the main reference source for developers to know how the processor works and how to configure it. Those documents mainly come from the processor manufacture.

STM32CubeIDE has a better way to list all related documents of selected processor, and it can download documents too. Find the documents in menu **Help** >> **Target device docs and resources**.



Access to the list of documents

> **The most important documents**
>
> Below sections are the documents that must be read in order to understand about the processor and its peripherals.

4.1. Processor Datasheet

DSxxxx - ARM-based MCU Datasheet

E.g: **DS8668** ARM-based 32-bit MCU, 16 to 64 KB Flash, 11 timers, ADC, DAC and communication interfaces, 2.0-3.6 V (version 7)

This document shows below information:

- Functional features
- Memory Mapping, Boot Mode
- Pinouts and altenative functions
- Core and Peripherals' diagrams, protocols

RMxxxx - Reference Manual

E.g: **RM0091** - Reference manual for STM32F0x1/STM32F0x2/STM32F0x8 advanced ARM-based 32-bit MCUs

This is a very important document as it defines all register structure, bit-fields to control core and peripherals. This document includes:

- System Architecture, Register-Level designs
- Memory Mapping and Boot configuration
- Register name and bit-fields for all accessible registers

PMxxxx" Programming Manual

E.g: **PM0215** - STM32F0xxx Cortex-M0 programming manual

The main content is:

- Processor modes: in application and interruption routine
- Stacks: manage context's data
- Core Registers: contain instruction's data and result, system status
- Memory Model: fixed memory map with address ranges
- Vector Table: start address, stack pointer, and interruption handlers
- Sleep mode: condition about clock, data rate, regiter mode
- Instruction set: assembly mnemonic for instructions

- Core registers: Address and Name of registers used in core and peripherals. This is the based for programming the processor. It also have API function name for accessing the registers using CMSIS.

ANxxxx - Application Note

E.g: **AN2548** Using the STM32F0/F1/F3/Gx/Lx Series DMA controller (version 7)

This type of document show:

- How to implement a specific problem in application
- Related configuration and library

4.2. Board schematic

MBxxxx - Mainboard schematic

E.g: **MB1034** RevB.0 - STM32F0-DISCOVERY Schematic

Check the version of hardware on the board, such as MB1034B. Go to ST website and download the corresponding schematic, or find it in User Manual document.

This document includes:

- Input and Output characteristics (Pull-up, Pull-down, Open, Voltage level)
- Connection points (internal wires, connectors)
- Working conditions (Power level, Votage Level torelance)

UMxxxx - User Manual

E.g. **UM1525** - Discovery kit for STM32F0 microcontrollers

Main content includes:

- Hardware components and their locations and markers
- Pinouts and jumpers for connections or configurations
- Solder Bridges for enabling/disabling features or hardware connections
- Mainboard schematic

4.3. Library Manual

UMxxxx - Description of HAL and Low-Layer driver

E.g. **UM1785** - Description of STM32F0 HAL and low-layer drivers

This document describes HAL and LL APIs for programming. Those functions wrap all internal registers and settings, and create a friendly function names and data structure for developer. The HAL drivers include a set of driver modules, each module being linked to a standalone peripheral.

The HAL main features are the following:

- Cross-family portable set of APIs covering the common peripheral features as well as extension APIs in case of specific peripheral features.
- Three API programming models: polling, interrupt and DMA.
- APIs are RTOS compliant:
  - Fully reentrant APIs
  - Systematic usage of timeouts in polling mode
- Support of peripheral multi-instance allowing concurrent API calls for multiple instances of a given peripheral (USART1, USART2, etc.)
- All HAL APIs implement user-callback functions mechanism:
  - Peripheral Init/DeInit HAL APIs can call user-callback functions to perform peripheral system level. Initialization/De-Initialization (clock, GPIOs, interrupt, DMA)
  - Peripherals interrupt events
  - Error events
- Object locking mechanism: safe hardware access to prevent multiple spurious accesses to shared resources
- Timeout used for all blocking processes: the timeout can be a simple counter or a time base

UMxxxx - Developing applications with RTOS

E.g. **UM1722** - Developing applications on STM32Cube with RTOS

This document is a reference to program user application in RTOS. This document has below content:

- FreeRTOS: overview, APIs, memory management, low power managements, and configuration

- CMSIS-RTOS: a higher layer to communicate between CMSIS and FreeRTOS
- Usage to create thread, use Semaphore, Queues, and Timer

CMSIS - Cortex Microcontroller Software Interface Standard

Refer to https://developer.arm.com/tools-and-software/embedded/cmsis

or the package https://developer.arm.com/embedded/cmsis/cmsis-packs/devices/STMicroelectronics/STM32F051R8

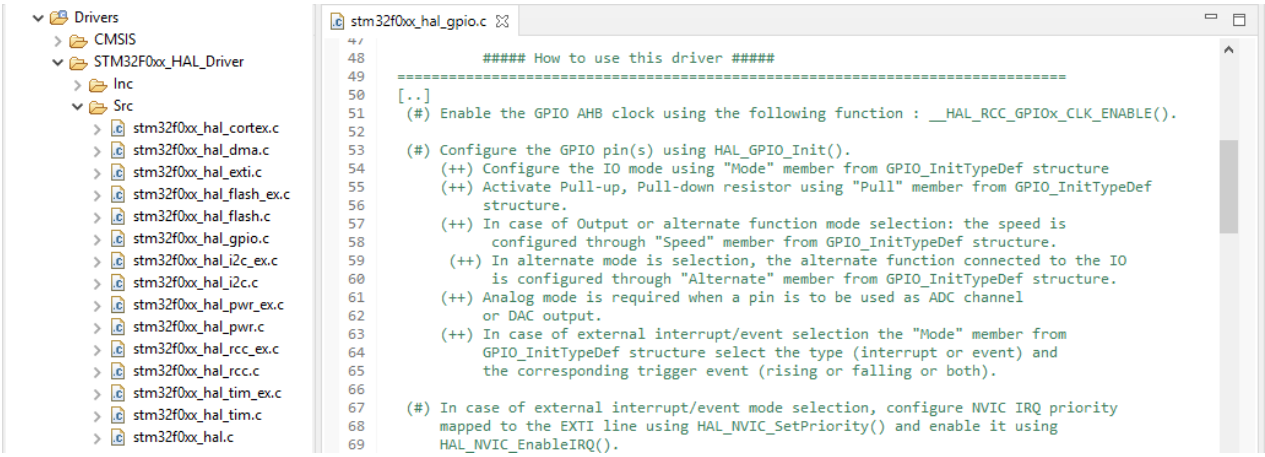
ARM develops the Cortex Microcontroller Software Interface Standard (CMSIS) to allow microcontroller and software vendor to use a consistent software infrastructure to develop software solutions for Cortex-M microcontroller. It is a set of APIs for application or middleware developers to access the features on the Cortex-M processor regardless of the microcontroller devices or toolchain used.

To use the CMSIS-Core (Cortex-M) the following files are added to the embedded application:

- Startup File `startup_<device>.c` with reset handler and exception vectors.
- System Configuration Files `system_<device>.c` and `system_<device>.h` with general device configuration (i.e. for clock and BUS setup).
- Device Header File `<device.h>` gives access to processor core and all peripherals. Register names and bit-fields are defined in the Reference Manual of the process.

4.4. Source code

HAL and CMSIS packages provide open source code with detailed comments. They are very useful when navigating between source files and function declarations within an IDE. In each HAL file, there is a comment section that describes the usage of that HAL driver.



Usage is written in the source code

4.5. Websites

There are plenty open resource to learn on the internet. Here are some websites that have something new to learn every times visit by.

- STM32 Education: official courses from ST
- STM32-base: Templates and settings for starting new STM32 projects