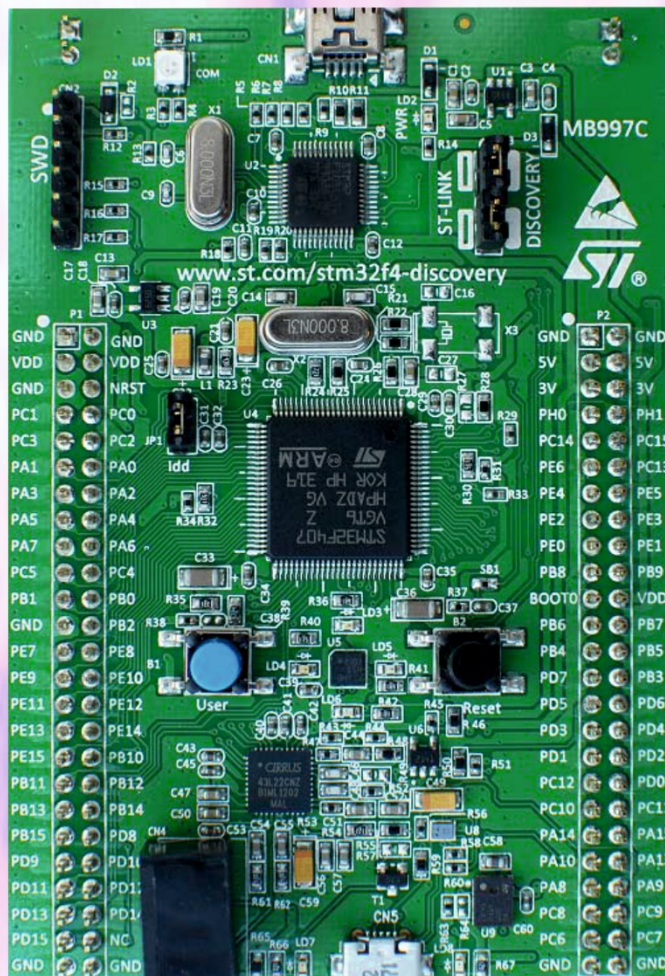


# ARM Microcontrollers Programming for Embedded Systems

Sever Spânulescu



STM32F4-Discovery CMSIS and HAL-API  
in IAR-EWARM or Keil-MDK

# **ARM Microcontrollers Programming for Embedded Systems**

STM32F4-Discovery CMSIS and HAL-API in IAR-EWARM or Keil-MDK

**Sever Spânulescu**

## **Copyright**

Copyright © 2019 SEVER SPÂNULESCU

All rights reserved.

This book or any portion thereof may not be reproduced or used in any manner whatsoever without the express written permission of the publisher except for the use of brief quotations in a book review or scholarly journal.

First Printing: 2019

## **Table of contents**

i. Introduction and motivation of the domain .....	9
ii. Examples of design topics.....	9
iii. Method, device and environment .....	10
<b>1. Installing the environment and writing directly to memory</b>	<b>11</b>
1.1 Presentation of a typical ARM microcontroller and a development board for it .....	12
1.2 Installing and launching the IAR development environment for ARM microcontrollers.....	14
1.3 IAR setting for the selected microcontroller and creation of a blank program for viewing memory and registers .....	15
1.4 Direct action on memory registers for controlling LED-coupled terminals .....	15
1.5 Commands on the registers in the memory space via the pointers .....	20
1.6 Inclusion of the constant and macro libraries declared by the manufacturer and the use of CMSIS	23
1.7 Use of CMSIS in STM32Cube_FW_F4_Vx.x.x.....	26
<b>2. Programming with API Functions, Standard Peripheral Library (SPL) .....</b>	<b>29</b>
2.1 General API programming features.....	29
2.2 Blink LED API LED, using Standard Peripheral Drivers.....	29
<b>3. Using the CubeMX initialization code generator .....</b>	<b>Error!</b>
Bookmark not defined.	
3.1 Overview of the CubeMX initialization code generator .....	<b>Error! Bookmark not defined.</b>
3.2 Install the CubeMX environment .....	<b>Error! Bookmark not defined.</b>
3.3 Create a new project using CubeMX .....	<b>Error! Bookmark not defined.</b>
3.4 Control of GPIO digital pins .....	<b>Error! Bookmark not defined.</b>
<b>4. Digital inputs on STM32F407VG microcontroller pins</b>	
Error! Bookmark not defined.	
4.1 Identify the User button input and track its action on the memory.....	<b>Error! Bookmark not defined.</b>

4.2 The CMSIS programming variant, through registers.....**Error! Bookmark not defined.**

4.3 Button readout and LED control via HAL-API.....**Error! Bookmark not defined.**

## 5. External interrupts ..... **Error! Bookmark not defined.**

5.1 External interrupts. Direct programming via registers.....**Error! Bookmark not defined.**

5.2 External interrupts using API functions.....**Error! Bookmark not defined.**

## 6. USART serial communication interfaces (Universal Synchronous/Asynchronous Receiver/Transmitter)..... **Error! Bookmark not defined.**

6.1 Generalities about RS232 communications .....**Error! Bookmark not defined.**

6.2 Data transmission.....**Error! Bookmark not defined.**

6.3 Data reception.....**Error! Bookmark not defined.**

6.4 Baud Rate Generator.....**Error! Bookmark not defined.**

6.5 Serial transmission program, registers programming.....**Error! Bookmark not defined.**

6.6 Program for serial transmission, reception and display, using registers **Error! Bookmark not defined.**

6.7 Serial transmission, interrupts reception and display, using registers ... **Error! Bookmark not defined.**

## 7. Serial communications via USART using API functions **Error! Bookmark not defined.**

7.1 Serial transmission of a byte .....**Error! Bookmark not defined.**

7.2 Program for serial transmission, reception and display, using API..... **Error! Bookmark not defined.**

7.3 Serial transmission, interrupts reception and display, using API..... **Error! Bookmark not defined.**

## 8. Analog inputs via Analogue-to-Digital Converter, registers programming..... **Error! Bookmark not defined.**

8.1 Integrated ADC overview .....**Error! Bookmark not defined.**

8.2 Analog-numeric conversion program, registry variant .....**Error! Bookmark not defined.**

- 8.3 Serial transmission of conversion result .....**Error! Bookmark not defined.**
- 8.4 Use of ADC with interrupts, registry variant .....**Error! Bookmark not defined.**

## 9. Analog inputs via Analog-to-Digital Converter, API variant Error! Bookmark not defined.

- 9.1 Analog-Numerical conversion using API functions without ADC interrupts..**Error! Bookmark not defined.**
- 9.2 Analog-Numerical conversion using API functions with ADC interrupts. **Error! Bookmark not defined.**

## 10. Timers in basic mode ..... **Error! Bookmark not defined.**

- 10.1 STM32F4 microcontroller timers overview.....**Error! Bookmark not defined.**
- 10.2 Use timer in basic mode, registry variant .....**Error! Bookmark not defined.**
- 10.3 Basic timer, HAL\_API variant .....**Error! Bookmark not defined.**
- 10.4 Use timer comparison outputs, registry variant .....**Error! Bookmark not defined.**
- 10.5 Use of TIM comparison outputs, HAL\_API variant.....**Error! Bookmark not defined.**

## 11. Timer as an external source counter, using interrupts Error! Bookmark not defined.

- 11.1 Use as a counter with external ETR source, registry variant.....**Error! Bookmark not defined.**
- 11.2 Use as a counter with external ETR source, HAL\_API variant ...**Error! Bookmark not defined.**
- 11.3 Use of Overrun and Comparison Interrupts, registry variants..**Error! Bookmark not defined.**
- 11.4 Using update interruptions, API variant.....**Error! Bookmark not defined.**
- 11.5 Using update interrupts to automate a sequence .....**Error! Bookmark not defined.**

## 12. Timers in PWM mode, registers ....**Error! Bookmark not defined.**

- 12.1 Generation of PWM signals and features for STM32F4 microcontrollers .....**Error! Bookmark not defined.**
- 12.2 Generating PWM signals, registers variant .....**Error! Bookmark not defined.**
- 12.3 Modifying the filling factor by ADC, registry variant .....**Error! Bookmark not defined.**

## 13. Timers in PWM mode, API version ....Error! Bookmark not defined.

- 13.1 Using timer in PWM mode with SPL-API function programming..... **Error! Bookmark not defined.**
- 13.2 Modifying the fill factor via ADC, API programming .....**Error! Bookmark not defined.**
- 13.3 Changing the fill factor from a terminal with API programming..... **Error! Bookmark not defined.**
- 13.4 Modifying the fill factor from an ultrasonic sensor, API Programming .. **Error! Bookmark not defined.**

## 14. SPI interface. Coupling with the LIS3DH accelerometer Error! Bookmark not defined.

- 14.1 Modes of SPI data transfer at STM32F4 microcontrollers ...**Error! Bookmark not defined.**
- 14.2 Operation in Master mode.....**Error! Bookmark not defined.**
- 14.3 LIS3DH accelerometer .....**Error! Bookmark not defined.**
- 14.4 Practical working .....**Error! Bookmark not defined.**
- 14.5 Displaying the 3D position in Labview.....**Error! Bookmark not defined.**

## 15. Interfața Inter-Integrated Circuit - I2C.Error! Bookmark not defined.

- 15.1 Overview of the I2C protocol .....**Error! Bookmark not defined.**
- 15.2 Operation of the I2C bus .....**Error! Bookmark not defined.**
- 15.3 Coupling two STM32F4 Discovery boards through I2C.....**Error! Bookmark not defined.**
- 15.4 I2C interface using HAL-API functions .....**Error! Bookmark not defined.**
- 15.5 I2C interface using registry programming.....**Error! Bookmark not defined.**
- 15.6 Master-Multiple slave operating.....**Error! Bookmark not defined.**

## 16. Computer Array Network Interface - CAN.....Error! Bookmark not defined.

- 16.1 CAN transmission .....**Error! Bookmark not defined.**
- 16.2 Reception in the CAN network.....**Error! Bookmark not defined.**

16.3 Building a CAN network using HAL-API functions, version STM32Cube\_FW\_F4\_V1.2x...**Error! Bookmark not defined.**

16.4 Building a CAN network using registry programming .....**Error! Bookmark not defined.**

## 17. Interfacing STM32F4-Discovery for Internet Communications ..... **Error! Bookmark not defined.**

17.1 The STM32F4-Discovery Program .....**Error! Bookmark not defined.**

17.2 Presentation of WiFi module for Internet connection.....**Error! Bookmark not defined.**

17.3 Preparation for programming the ESP8266 module.....**Error! Bookmark not defined.**

17.4 Server program for ESP8266 .....**Error! Bookmark not defined.**

17.5 Creating an Android application for Internet ordering of the STM32F4-Discovery Plate**Error! Bookmark not defined.**

## 18. Annexes. Extracts from the STM32F4xx microcontroller documentation ..... **Error! Bookmark not defined.**

18.1 Memory Map and Interrupt Table for STM32f4xx Microcontrollers ..... **Error! Bookmark not defined.**

18.1.1 STM32F4xx register boundary addresses.....**Error! Bookmark not defined.**

18.1.2 Table 62. Vector table for STM32F405xx/07xx and STM32F415xx/17xx.....**Error! Bookmark not defined.**

18.2 Registers associated with buses .....**Error! Bookmark not defined.**

18.2.1 RCC AHB1 peripheral clock enable register (RCC\_AHB1ENR)..... **Error! Bookmark not defined.**

18.2.2 RCC APB2 peripheral clock enable register(RCC\_APB2ENR) ..... **Error! Bookmark not defined.**

18.3 Registers associated with some input/output ports .....**Error! Bookmark not defined.**

18.3.1 GPIO port mode register (GPIOx\_MODER) (x = A..I/J/K) ...**Error! Bookmark not defined.**

18.3.2 GPIO port output data register (GPIOx\_ODR) (x = A..I/J/K) ..... **Error! Bookmark not defined.**

18.3.3 GPIO port output type register (GPIOx\_OTYPER) (x = A..I/J/K) ..... **Error! Bookmark not defined.**

18.3.4 GPIO port input data register (GPIOx\_IDR) (x = A..I/J/K) ...**Error! Bookmark not defined.**

18.3.5	Interrupt mask register (EXTI_IMR).....	<b>Error! Bookmark not defined.</b>
18.3.6	Rising trigger selection register (EXTI_RTSTR).....	<b>Error! Bookmark not defined.</b>
18.3.7	Pending register (EXTI_PR) .....	<b>Error! Bookmark not defined.</b>
18.3.8	GPIO alternate function low register (GPIOx_AFRL) (x = A..I/J/K)...	<b>Error! Bookmark not defined.</b>
18.4	Registers associated with USART .....	<b>Error! Bookmark not defined.</b>
18.4.1	USART Status register (USART_SR) .....	<b>Error! Bookmark not defined.</b>
18.4.2	USART Control register 1 (USART_CR1).....	<b>Error! Bookmark not defined.</b>
18.4.3	Baud rate register (USART_BRR) .....	<b>Error! Bookmark not defined.</b>
18.5	Registers associated with analog-numeric converters.....	<b>Error! Bookmark not defined.</b>
18.5.1	ADC control register 2 (ADC_CR2) .....	<b>Error! Bookmark not defined.</b>
18.5.2	ADC sample time register 1 (ADC_SMPR1) .....	<b>Error! Bookmark not defined.</b>
18.5.3	ADC regular sequence register 3 (ADC_SQR3) .....	<b>Error! Bookmark not defined.</b>
18.5.4	ADC status register (ADC_SR) .....	<b>Error! Bookmark not defined.</b>
18.6	Registers associated with timers.....	<b>Error! Bookmark not defined.</b>
18.6.1	TIMx control register 1 (TIMx_CR1) .....	<b>Error! Bookmark not defined.</b>
18.6.2	TIMx prescaler (TIMx_PSC).....	<b>Error! Bookmark not defined.</b>
18.6.3	TIMx capture/compare mode register 1 (TIMx_CCMR1) ..	<b>Error! Bookmark not defined.</b>
18.6.4	TIMx capture/compare mode register 2 (TIMx_CCMR2) ..	<b>Error! Bookmark not defined.</b>
18.6.5	TIMx capture/compare enable register (TIMx_CCER).....	<b>Error! Bookmark not defined.</b>
18.6.6	TIMx slave mode control register (TIMx_SMCR) .....	<b>Error! Bookmark not defined.</b>
18.6.7	TIMx event generation register (TIMx_EGR) .....	<b>Error! Bookmark not defined.</b>
18.6.8	TIMx DMA/Interrupt enable register (TIMx_DIER) .....	<b>Error! Bookmark not defined.</b>
18.6.9	TIMx status register (TIMx_SR).....	<b>Error! Bookmark not defined.</b>
18.7	Registers associated with I2C interface.....	<b>Error! Bookmark not defined.</b>
18.7.1	I <sup>2</sup> C Control register 1 (I2C_CR1) .....	<b>Error! Bookmark not defined.</b>
18.7.2	I <sup>2</sup> C Status register 1 (I2C_SR1) .....	<b>Error! Bookmark not defined.</b>



18.7.3	I <sup>2</sup> C Status register 2 (I2C_SR2) .....	<b>Error! Bookmark not defined.</b>
18.8	Registers associated with CAN interface .....	<b>Error! Bookmark not defined.</b>
18.8.1	CAN TX mailbox identifier register (CAN_TlRx) (x=0..2) ....	<b>Error! Bookmark not defined.</b>
18.8.2	CAN transmit status register (CAN_TSR) .....	<b>Error! Bookmark not defined.</b>
18.8.3	CAN mailbox data length control and time stamp register (CAN_TDTxR) (x=0..2) ..	<b>Error! Bookmark not defined.</b>
18.8.4	CAN receive FIFO mailbox identifier register (CAN_RlRx) (x=0..1) ..	<b>Error! Bookmark not defined.</b>
18.8.5	CAN filter master register (CAN_FMR) .....	<b>Error! Bookmark not defined.</b>
18.8.6	CAN filter mode register (CAN_FM1R) .....	<b>Error! Bookmark not defined.</b>
18.8.7	CAN filter scale register (CAN_FS1R) .....	<b>Error! Bookmark not defined.</b>
18.8.8	CAN filter FIFO assignment register (CAN_FFA1R) .....	<b>Error! Bookmark not defined.</b>
18.8.9	CAN filter activation register (CAN_FA1R) .....	<b>Error! Bookmark not defined.</b>
18.8.10	Filter bank i register x (CAN_FiRx) (i=0..27, x=1, 2) .....	<b>Error! Bookmark not defined.</b>
18.8.11	CAN interrupt enable register (CAN_IER) .....	<b>Error! Bookmark not defined.</b>
18.8.12	CAN receive FIFO 0 register (CAN_RF0R) .....	<b>Error! Bookmark not defined.</b>

## **i. Introduction and motivation of the domain**

Digital techniques have experienced the fastest development of all technical fields and have proliferated in the past decades in all areas of human activity, with increasing performances. To a large extent, this evolution was due to the demand for high-tech computing, which attracted immense investments and became more and more accessible. Technological developments brought about by the need of computing have first been taken over by the industry and then become common to many applications of a typically digital nature. Integrating in a single circuit all blocks of a central processing unit of a computer, the **microprocessor** has captured the attention of users interested in command and control areas for industrial, space, military applications, etc. Achieving affordable and small sized Personal Computer systems seemed for a while a solution for applying microprocessor in control systems. Programmable

Logic Controller (PLC) modular systems with increased reliability destined to control and control destinations in the industrial sphere have been developed, using increasingly evolved microprocessors, that are currently used. However, the requirements of resilience in hostile environments have drastically reduced their fall in prices, and hence the scope of applicability, for economic reasons.

As an alternative to mechanical modularization of standard digital components in PLC solutions, microprocessor manufacturers have followed another path: integrating the processor core, memory, and in-out ports on the same chip. A new class of devices, capable of "own intelligence" - **microcontrollers** - has emerged. Incorporating all the blocks needed to run any algorithm, microcontrollers can obtain, by software, the intelligence of that algorithm just as a classical computing system. Instead, reliability is much higher due to the smallest possible number of components, consumption and volume are greatly reduced, and as a consequence the low-cost price allows them to be incorporated into a wider range of applications.

This way, the **Embedded systems** have emerged, that have rapidly expanded their applicability, ranging from industry, consumer goods, medicine, communications, transport, and so forth. As a result, it has been estimated, even 10 years ago, that over 98% of all processors were used by embedded systems, and the trend is that the range of industrial products using microcontrollers is widening more and more.

## ii. Examples of design topics

Suppose we have to design the command and control system for a sophisticated technology line in a highly automated factory. A first alternative that comes to our attention is the use of a classic computerized system, with a central unit, peripheral equipment and sensor systems, actuators and communication paths. On a pragmatic analysis, however, some problems of this solution will appear. Thus, with many highly specialized subsystems, reliability and average running time are low. Any failure in such a subsystem may stop the entire line. Even highly mechanical PLC solutions have many connections, which are low reliability points. In addition, initial investment and maintenance costs are quite high, whether justified or not.

A derived solution would be not to centralize the entire line functionality in a single computer, but to use multiple nodes equipped with local intelligence, which are in correspondence with a central computer that will have the role of supervisor (SCADA system). Each node will have a much simpler task and lower performance requirements, sacrificed for reliability. Here it is clear that a microcontroller is net superior, having a minimal number of components, low consumption and therefore much relaxed heat regime. In addition, the microprocessor can be embedded in the subassembly, so most of the signals will be locally transmitted and the flow to the central computer will be much lower, thus more reliable. If the technological line is not too complex, the central computer can be also made with a microcontroller.

Let us now consider a simpler system, for example an elevator. The operating algorithm is simpler, and the price and space occupied implies some limits, so from the start a classical computing system with the associated peripheral needs is excluded. Here is a lot more advantageous to use a microcontroller or, as the case may be, even a set of microcontrollers, without significant repercussions on price and space.

It can go in the same direction with the incorporation of microcontrollers below: a semaphore (hard to imagine that even a trivial PC would be required), a washing machine (a

PC?), a microwave, a phone, a smartphone, and so on. Minor objects around us, priced well below \$ 1, such as a phone card or an RFID card, already incorporate a microcontroller.

Our problem is that all these microcontrollers do not actually have their own intelligence, but they include a part of the man writing it in the non-volatile memory - **the programmer**.

It should be mentioned here that the programming of an embedded system must respond to increased requirements, making it more difficult than a classical one.

First of all, it requires a deterministic programming - everything has to be performed in a certain succession, anticipated by the algorithm, without waiting queues and races between events, so in a real-time operating system -RTOS (**Real Time Operating System**). Any departure from the algorithm's time limits, as can happen with non-deterministic operating systems (Windows, Linux, etc.), would result in application failure.

However, embedded system programming in RTOS is nevertheless accessible to a large number of professional or even amateur programmers. The purpose of this paper is to broaden their class, with individual and socially predictable beneficial consequences.

### **iii. Method, device and environment**

The success of such an approach is conditioned by some primary factors. From the point of view of the author they are: attractive motivation and cursive method. Assuming that the motivation was somewhat exposed above, the method must also have some didactic characteristics: it must start from an accessible level and follow a learning curve high enough for progress to become rapidly visible, but not so steep as to trigger abandonment.

As in any programming learning context, working on a real device is mandatory. Therefore, an affordable device that requires no significant investment, but should be competitive as a performance with the state of the art, has to be chosen. Of the many currently used microcontroller architectures, ARM's (Advanced Risc Machine) can be considered the most widespread and most powerful, and the widest offer of ARM microcontrollers belongs to STMicroelectronics. Of the ARM microcontrollers dedicated to embedded systems (class M, other than class A for classical operating systems), the latest and most powerful appearances are Cortex M4 and Cortex M7.

In the present paper, the Cortex M4 was chosen, which is the core of one of the most advantageous development boards in the year 2019, namely STM32F4-Discovery, which is considered as already in a mature stage and has an affordable price (under 20 \$).

Working with this development board is representative for the most advanced ARM microcontrollers and allows for a mild approach to any other device that can be used in embedded systems. Also, the general principles learned through the examples presented, apply to a wide range of related applications.

The programming language chosen here is C, unlike other author's previous works in the microcontroller field, where the assembly language was mostly used. I considered that this language is a reasonable compromise between performance, strained by the high effort imposed by the assembler and the simplicity, spoiled by low performance, of other high-level languages. For engineers, the only notable disadvantage of C is that it was natively not designed to integrate Web technologies (which, moreover, appeared much later). From the point of view of physicists, of course Fortran holds the supremacy, being used by many supercomputers, but it is hardly adaptable to embedded systems.

C programming of microcontroller systems can be done using a variety of more or less integrated environments, with or without license costs. It has been chosen here one of the two professional development environments, namely IAR-EWARM. The direct competitor, Keil-

MDK is very similar, and the transition from one to the other is immediate: all the presented programs are the same. Although the license is quite expensive, both offer free fully functional variants, but with a space limit (32KB), which is absolutely sufficient for all the examples presented in the paper, especially in the case of registry programming, or a time limit (30 days).

The paper is intended primarily for students from the faculties of automation and computer science which have some knowledge of C, preparing them for professional applications. Although it can also be attractive for amateurs, it is necessary to provide an adequate motivation to renounce the comfortable Arduino environment, so switching from 8-bit AVR systems to 32/64-bit ARM systems.

## **1. Installing the environment and writing directly to memory**

To accomplish the goal of this first project, we will take the following steps:

1. Install and launch the IAR development environment for ARM microcontrollers;
2. IAR setting for the selected microcontroller and creation of a blank program for viewing memory and registers;
3. Direct action on memory registers for controlling LED-coupled terminals.

### **1.1 Presentation of a typical ARM microcontroller and a development board for it**

In the following, we will use the STM32F4-Discovery development board equipped with the STM32F407VG microcontroller from the ARM Cortex-M4 family of microcontrollers STM32F405xx / 07xx, STM32F415xx / 17xx, STM32F42xxx, STM32F43xxx. The trade name of the variant, starting from the year 2017, is STM32F407G-DISC1 (Figure 1.1).

STM32F407G-DISC1 - STMICROE

https://ro.farnell.com/stmicroelectronics/stm32f407g-disc1/dev-board-foundation-line-mcu/dp/2506840?st=STM32F4%20DISC1

Apps Google YouTube Sever Spanulescu - Y: (29070 unread) - seve MIT App Inventor Torrent Search Engine Other bookmarks

Farnell element14

Toate STM32F4 DISC1

Conectare | Înregistrare  
Contul meu

0 articole  
0,00 lei


Toate produsele Producători Instrumente Resurse și ajutor Centrul IoT Comunitate

LIVRARE RAPIDĂ\* PROGRAM DE SĂRBĂTORI PÂNĂ LA 10% REDUCERE\* LIVE AGENT

Pagina principală > Embedded Computers, Education & Maker Boards > ARM > Embedded Development Kits - ARM > STM32F407G-DISC1

Ati găsit o greșeală? Imprimare pagină

### STM32F407G-DISC1 - Development Board, STM32F407 High Performance MCU's, Various Sensors, Develop Audio Applications



Produsător: STMICROELECTRONICS

Nr. de reper producător: STM32F407G-DISC1

Cod de comandă: 2506840

Fișă tehnică: STM32F407G-DISC1 Datasheet

Consultați toate documentele tehnice

★★★★★ Scrieți O Recenzie

Imaginea are caracter strict ilustrativ. Vă rugăm consultați descrierea produsului.

Adăugați pentru comparare

#### Detalii tehnice

Silicon Manufacturer:	STMicroelectronics	Silicon Core Number:	STM32F407VG
No. of Bits:	32bit	Kit Contents:	Dev Board STM32F407VG
Silicon Family Name:	STM32F4	Product Range:	-
		RoHS Phthalates	Yes

**1.044 În stoc**

205 se livrează în 1-2 zile din depozitul nostru din Liege

Produsele pentru clienții neplătitori de TVA sunt expediate numai din depozitul nostru din Marea Britanie. Livrarea comenzii dvs. poate fi întârziată în cazul în care comanda depășește nivelul stocului disponibil în Marea Britanie. Ascundeți acest avertisment

839 se livrează în 1-2 zile din depozitul nostru din UK

consultați perioadele de reduceri

Verificare stoc și termen de livrare

**80,04 lei**

Preț pentru: Bucată

Mai multe: 1 Minim: 1

Cantitate	Preț
1 +	80,04 lei

Solicitați o ofertă pentru cantități mari

Cantitate 1

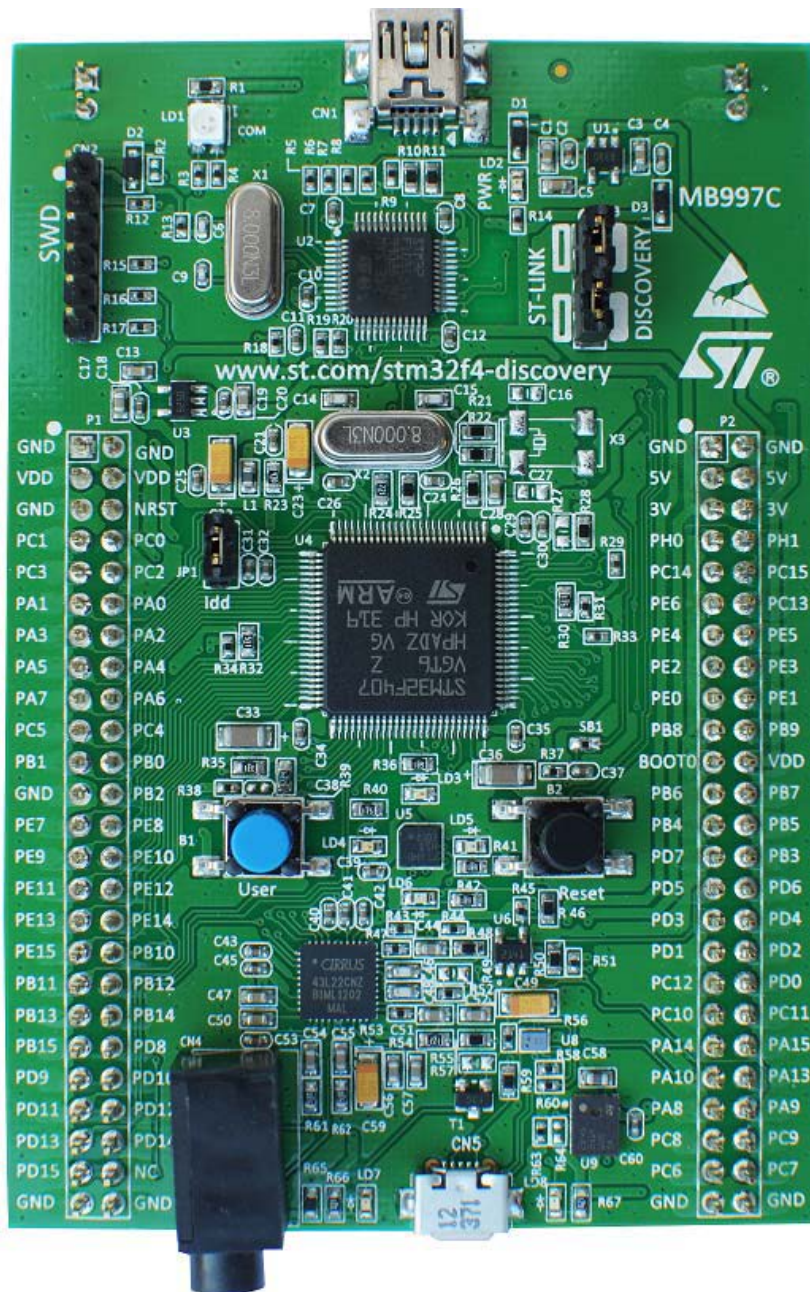
Adăugare la coș

Adăugare nr. piesă /notă

Adăugare la favorite

**Figure 1.1.** Commercial offer of the STM32F4 Discovery module

We present here some of the main features of the board and the microcontroller that equips it, as specified by the manufacturer.



**Figure 1.2.** The appearance of the STM32F4 Discovery Module

## Features

- STM32F407VGT6 microcontroller featuring 1 MB of Flash memory, 192 KB of RAM in an LQFP100 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
- External application power supply: 3V and 5V
- LIS302DL or LIS3DSH, ST MEMS motion sensor, 3-axis digital output accelerometer
- MP45DT02, ST MEMS audio sensor, omnidirectional digital microphone
- CS43L22, audio DAC with integrated class D speaker driver
- Eight LEDs: – LD1 (red/green) for USB communication – LD2 (red) for 3.3V power on – Four user LEDs, LD3 (orange), LD4 (green), LD5 (red) and LD6 (blue) – 2 USB OTG LEDs LD7 (green) VBus and LD8 (red) over-current
- Two pushbuttons (user and reset)

- USB OTG with micro-AB connector
- Extension header for LQFP100 I/Os for quick connection to prototyping board and easy probing

## LEDs

- LD1 COM: LD1 default status is red. LD1 turns to green to indicate that communications are in progress between the PC and the ST-LINK/V2.
- LD2 PWR: red LED indicates that the board is powered.
- User LD3: orange LED is a user LED connected to the I/O PD13 of the STM32F407VGT6.
- User LD4: green LED is a user LED connected to the I/O PD12 of the STM32F407VGT6.
- User LD5: red LED is a user LED connected to the I/O PD14 of the STM32F407VGT6.
- User LD6: blue LED is a user LED connected to the I/O PD15 of the STM32F407VGT6.
- USB LD7: green LED indicates when VBUS is present on CN5 and is connected to PA9 of the STM32F407VGT6.
- USB LD8: red LED indicates an overcurrent from VBUS of CN5 and is connected to the I/O PD5 of the STM32F407VGT6.

## Pushbuttons

- B1 USER: User and Wake-Up button connected to the I/O PA0 of the STM32F407VGT6.
- B2 RESET: Pushbutton connected to NRST is used to RESET the STM32F407VGT6.

## STM32F407VGT6 microcontroller

- ARM Cortex-M4 32-bit MCU with FPU
- USB OTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 15 comm. interfaces.
- \* 168 MHz/210 DMIPS Cortex-M4 with single cycle DSP MAC and floating point unit providing: Boosted execution of control algorithms
- \* ART Accelerator, 32-bit, 7- layer AHB bus matrix with 7 masters and 8 slaves including 2 blocks of SRAM, Multi DMA controllers: 2 general purpose, 1 for USB HS, 1 for Ethernet, One SRAM block dedicated to the core, providing performance equivalent to 0-wait execution from Flash Concurrent execution and data transfers and simplified resource allocation
- \* Ultra-low dynamic power, RTC <1  $\mu$ A typical in VBAT mode, 3.6 V down to 1.7 V VDD,
- \* Voltage regulator with power scaling capability when running at low voltage or on a rechargeable battery
- \* Up to 1 Mbyte of on-chip Flash memory, 192 Kbytes of SRAM, reset circuit, internal RCs, PLL.
- \* Extensive tools and software solutions providing a wide choice within the STM32 ecosystem to develop your applications.

## 1.2 Installing and launching the IAR development environment for ARM microcontrollers

Multiple files and drivers are required to use the development board.

a. The IAR Embedded Workbench IDE program is downloaded from <https://www.iar.com/iar-embedded-workbench/> where Free trial-> Download Software is selected. The projects in this paper have been verified on IAR-EWARM versions 6.xx, 7.xx and 8.xx.

b. Normally, Windows 10 finds the STM32F4-Discovery card reader when plugged into USB. For Windows 7/8 on 32/64 bit, the driver stsw-link009 can be downloaded from:

<http://www.st.com/web/en/catalog/tools/PF260219>

c. For examples using Standard Peripheral Drivers in the first part of the paper, the firmware and development file system, called STSW-STM32068, which contains the STM32F4-Discovery\_FW\_V1.1.0 directory (which must preferably be unzipped in the C: \ disk), may be required can be found at:



<http://www.st.com/web/en/catalog/tools/PF257904#>. As you may be warned, its use is not recommended in developing new projects, but here we use it for didactic purposes to understand the software context at the lowest level.

d. The DM00039084.pdf manual of the STM32F4-Discovery board, which can be downloaded from:

[http://www.st.com/st-web-ui/static/active/cn/resource/technical/document/user\\_manual/DM00039084.pdf](http://www.st.com/st-web-ui/static/active/cn/resource/technical/document/user_manual/DM00039084.pdf)

e. The most important document is the Reference Manual DM00031020.pdf for STM32F4xx microcontrollers that can be downloaded from the address

[http://www.st.com/web/en/resource/technical/document/reference\\_manual/DM00031020.pdf](http://www.st.com/web/en/resource/technical/document/reference_manual/DM00031020.pdf)

First, install the IAR Embedded Workbench IDE program, choosing the free version with a 32 KB memory limit (which has no other restrictions), and in this process, when you are prompted, the drivers are also installed. If they are not installed correctly, run the driver program, and check their status.

It is helpful to unpack now STSW-STM32068, by creating the C:\STM32F4-Discovery\_FW\_V1.1.0 directory, as well as a D:\STM32F4 work directory in which to prepare the board and microcontroller documentation.

### **1.3 IAR setting for the selected microcontroller and creation of a blank program for viewing memory and registers**

To do this, take several steps as follows:

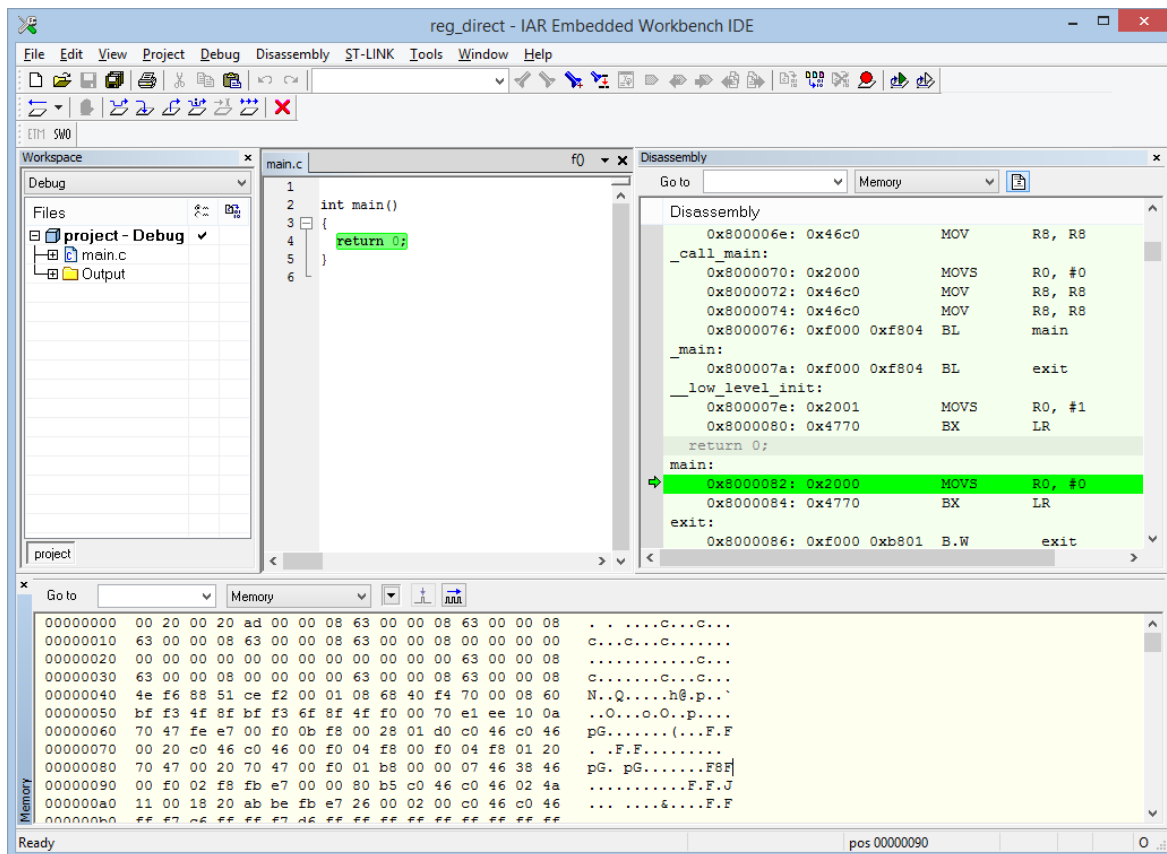
- a. IAR is launched
- b. Project-> Create new project
- c. Choose C-> main.
- d. Choose the work directory and a project name
- e. Project-> Options is given
- f. In General, choose the type of processor (ST-> STM32F407-> STM32F407VG)
- g. In the Debugger, select in Setup-> Driver-> ST-Link and Download Verify and Use Flash loader
- h. Also in Debugger, for ST-LINK select SWD
- i. Compile (F7)
- j. Set a name for the object file
- k. For loading the program into the microcontroller memory click Project-> Download and Debug (CTRL + D) or click on the respective button in the top bar

### **1.4 Direct action on memory registers for controlling LED-coupled terminals**

After loading the program into the flash memory, a Debug window appears in the right where the resulting program in the assembler is displayed. A click on the bar of this window makes it active, so you can step-by-step instructions in assembly language.

We click View-> Memory, so that an additional memory window appears, which can be pulled from the left vertical bar over the message as in Figure 1.3.





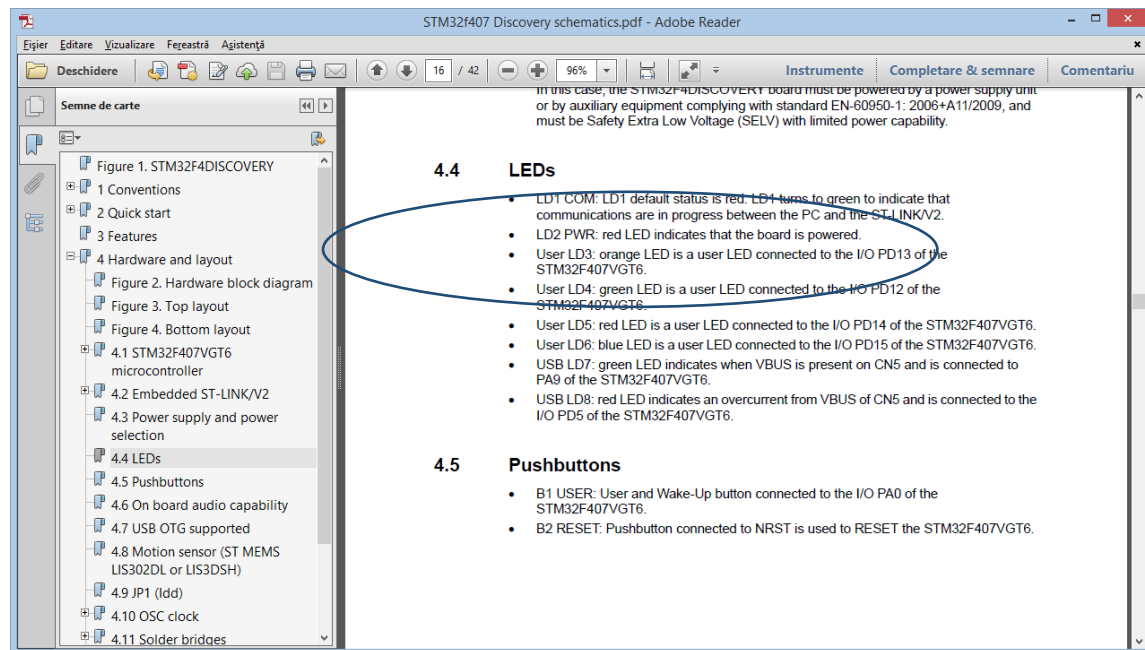
**Figure 1.3.** Minimal program in the IAR - EWARm development environment

A portion of the memory space is seen in the bottom window: the first column on the left is the addresses, and then in groups of 1 to 4 bytes the contents of those locations in hexadecimal and then their equivalent in ASCII.

According to the general operating mode of ARM microcontrollers, two conditions must be met to control a port output:

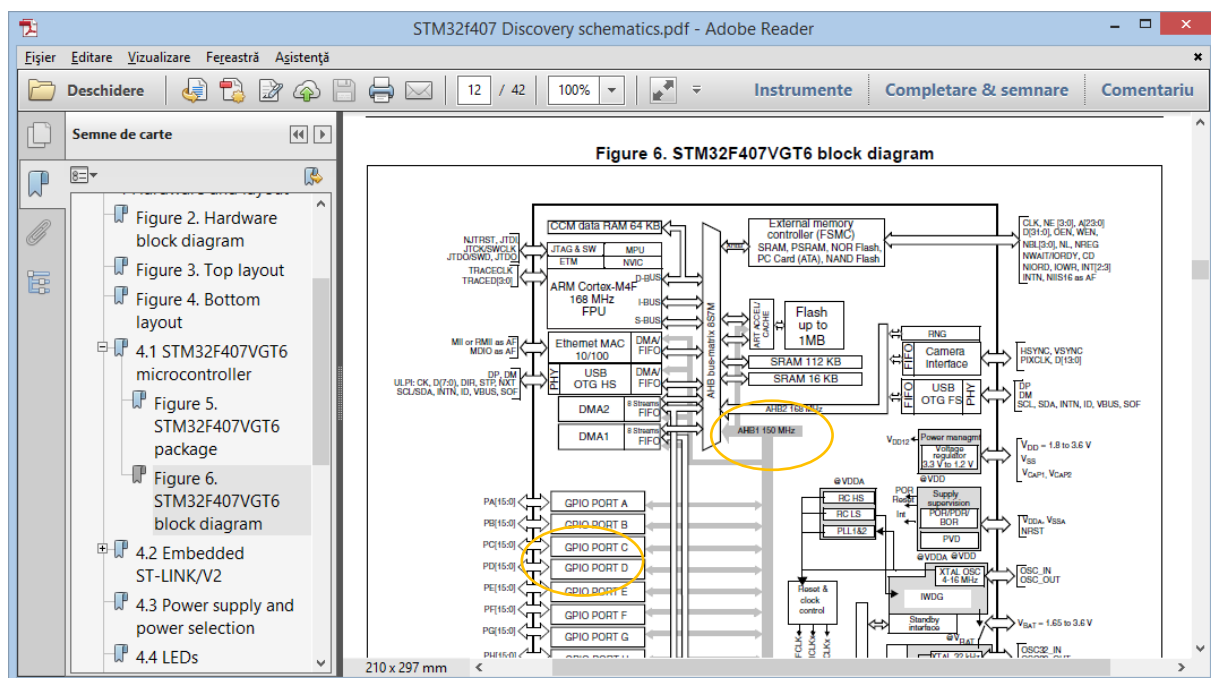
- Generating the clock signal to the respective port;
- Setting the output direction for the pin.

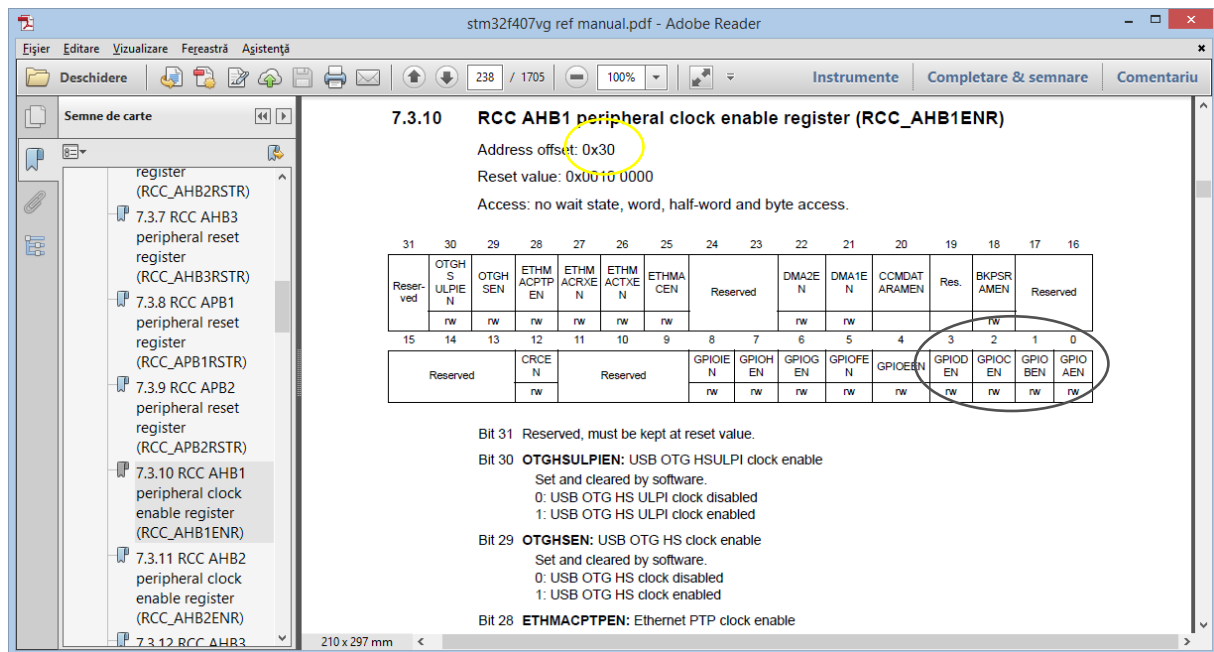
For the target board, the "STM32F407 Discovery Schematics.pdf" documentation shows (page 16) that the user LEDs are on the PD12, PD13, PD14 and PD15 pins (which are part of the GPIOD port), and the user button finds out on PA0 of the GPIOA port, as shown in Figure 1.4.



**Figure 1.4.** Coupling the LEDs to the STM32F4 Discovery module

The block diagram of the microcontroller (page 12) shows that these ports are coupled to the AHB1 bus as shown in Figure 1.5. Therefore, it would be necessary to generate clocks for these ports through the registers that control the clock of this bus.





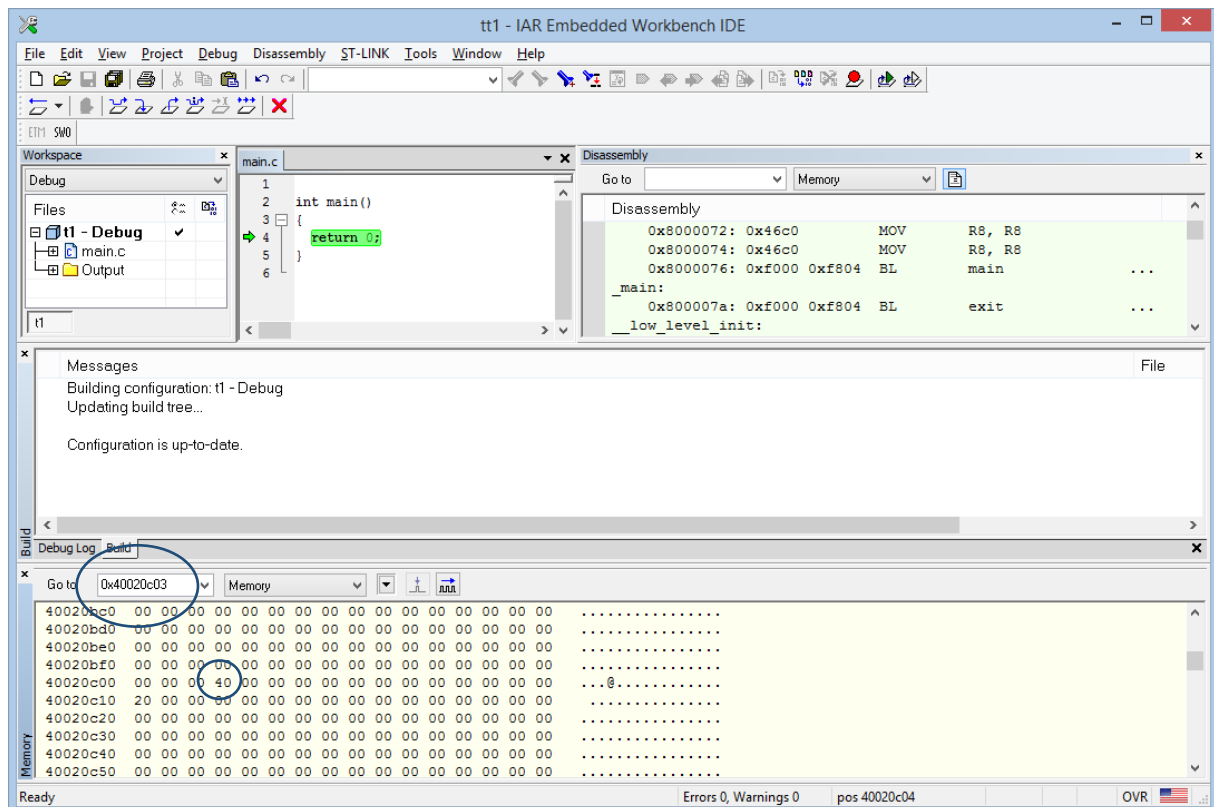
**Figure 1.6.** The bits that enable the clock to the GPIO ports on the STM32F407 microcontroller

Note that the offset address of this register is 0x30 (the second row), and look for its absolute address in the memory. For this we go to "2.3 [Memory map](#)" and on page 65 we find that the RCC base address is 0x4002.3800. Here too we see that the GPIOD base address is 0x4002.0C00.

The first thing we have to do is write a byte with the bit 3 in logic 1 (0b00001000 ie 0x08) at 0x4002.3800 + 0x30, so at 0x4002.3830. For this, in the IAR program's memory window, in the "Go to" window enter 0x40023830 hit Enter, and at this address write 08. This unlocks the clock to GPIOD.

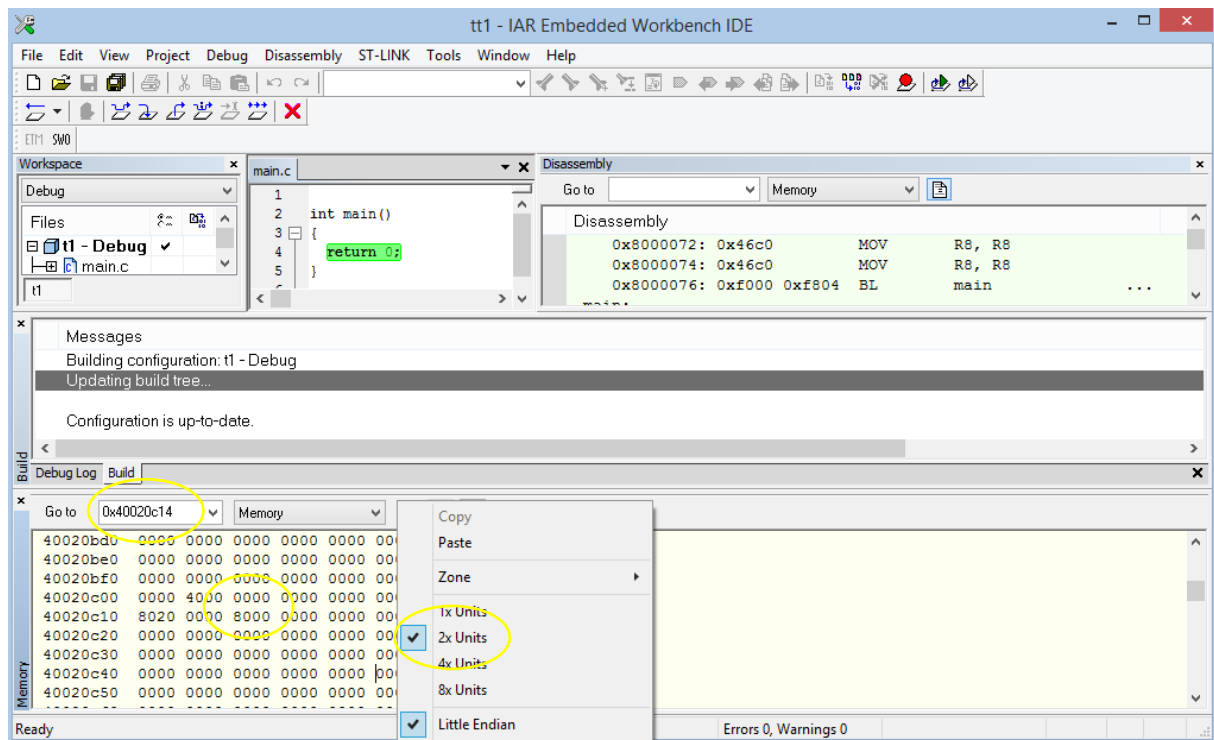
To set a pin as output, the GPIO associated registers are searched for the pins working mode. Under "8. General-purpose I/Os" -> "8.4.1 [GPIO port mode register](#)" on page 278 it is noticed that the offset address is 0x00 and each pin has 2 bits in the 32-bits registers (which occupy 4 bytes in space memory). The 4 pins coupled to LEDs (PD12, PD13, PD14 and PD15) have allocated the bits 24-31, so the highest address of the 4 assigned to this register is 0x4002.0C00 + 0x00 + 0x03 = 0x4002.0C03. From the bits description it follows that to set the output mode on a pin, the combination 01 should be written in those positions. For example, to set the pin 15 (corresponding to the blue LED) as the output, at the address 0x4002.0C03 should be written 0b01000000 = 0x40. This is equivalent to writing from the address 0x4002.0C00 the word consisting of 4 bytes 0x4000.0000.

Enter the address 0x4002.0C03 at Memory-> Go to and write 40, which sets the pin PD15 as the output (Figure 1.7).



**Figure 1.7.** Set the PD15 pin of the GPIOD port to the output mode

Finally, it can now be written to the GPIOD data register, which is described in 8.4.6 (page 281) of the microcontroller documentation. It is noted that a logic 1 is to be written in position 15 of that register, i.e.  $0b1000.0000.0000.0000 = 0x8000$  at  $0x40020C00 + 0x14 = 0x40020C14$ . To write a 16-bit word, select 2 x Units from the drop-down window and in the Go to window type 0x40020C14, as in Figure 1.8. Now writing 8000, the blue LED will light up.



**Figure 1.8.** Writing in 1 of the PD15 pin of the GPIOD port

If we want to be able to light up any of the 4 LEDs, we should write a 01 in each of the positions corresponding to the pins, ie  $0b0101.0101 = 0x55$  at  $0x4002.0C03$ . After that, the LED on PD14 can be lit by writing the word  $0b0100.0000.0000.0000 = 0x4000$  at  $0x40020C014$ , the one on PD13 with  $0b0010.0000.0000.0000 = 0x2000$  and the one on PD12 writing  $0b0001.0000.0000.0000 = 0x1000$ .

## 1.5 Commands on the registers in the memory space via the pointers

As seen in the previous paper, the following operations are required to turn on the blue LED on the STM32F407 Discovery board:

1. Write at  $0x4002.3830$  byte  $0x08$ ;
2. Write the word  $0x4002.0C00$  at the address  $0x5500.0000$  (ie  $0x55$  byte to  $0x4002.0C03$ );
3. Type  $0x4002.0C14$  word  $0x8000$  (ie  $0x80$  byte at  $0x4002.0C15$ );

It is possible to write either a word of 4 or 2 bytes starting with the offset address defined in the documentation for the respective register, or a single byte in the form mentioned in brackets. It is also noted that all the 4 more significant bits of GPIOD were set as outputs, as shown in the previous lab.

In C language, an address is indicated via a pointer, using an asterisk. Reference to the contents of the respective memory location is made by placing the asterisk before the pointer to the address:

**\* pointer**

On the other hand, to show that a numerical expression is a pointer type, it is declared as such with a cast type operator:

**(type \*) expression**

To write  $0x4002.3800$  byte  $0x08$ , we enter the following line in the main program:

```
*(int *)0x40023830 = 0x08;
```

By doing the same with other commands, the main program becomes:

```
int main()  
{  
    *(int *)0x40023830 = 0x08;  
    *(int *)0x40020C00 = 0x55000000;  
    *(int *)0x40020C14 = 0x8000;  
    return 0;  
}
```

Now loading the program into memory, when we click Go (or hit F5), the LED will turn on. It is possible to change the output word, and by flashing again in memory and running, other LEDs may be turned on or off.

To avoid hexadecimal transformations of some words, sometimes difficult and error-generating, you can use the left-hand operator `<<` (see for example [http://en.wikipedia.org/wiki/Bitwise\\_operations\\_in\\_C](http://en.wikipedia.org/wiki/Bitwise_operations_in_C)). Thus, the expression  $(1 \ll 3)$  is equivalent to `0x08` because a logically shifted left to three bits becomes  $0b0000.1000 = 0x08$ . Similarly, the expression  $(0x55 \ll 24)$  is equivalent to `0x5500.0000` because  $0b0101.0101 = 0x55$  shifted 24 bits to the left becomes  $0b0101.0101.0000.0000.0000.0000.0000.0000 = 0x5500.0000$ . You may also write  $0x55 \ll 2*12$  or better  $0x55 \ll (0x0C \ll 1)$ .

Therefore to light up all 4 LEDs the program can be written as:

```
int main()  
{  
    *(int *)0x40023830 = (1<<3);  
    *(int *)0x40020C00 = (0x55<<24);  
    *(int *)0x40020C14 = (0x0F<<12);  
    return 0;  
}
```

Sometimes it is easier to understand the program if you set the bits individually and use the operator "or bit" `|`. Thus, for example, if it is desired to put the bits 14 and 12 of the data register in 1, it can be written:

```
*(int *)0x40020C14 = (1<<14) | (1<<12);
```

Also, if the combination 01 is desired in the mode bits of pins 15 and pins 14, given that each pins has two mode bits, one can write:

```
*(int *)0x40020C00 = (1<<2*15) | (1<<2*14);
```

This operator is recommended to be used in the composite operator `|=` (attention without space between the two characters), which allows you to set **only** the target bits in that register, leaving the other bits unchanged. For example, if we do not want to affect the clock on other AHB1 peripherals, we will use the syntax with `|=`

```
*(int *)0x40023830 |= (1<<3);
```

If we want to turn on and off the LED periodically, it would be necessary to enter the turn on command followed by the turn off command into a “while” loop that runs as long as the condition is met. If condition 1 is used, it is always true and the loop becomes infinite, as in the program:

```
int main ()
{
    * (int *) 0x40023830 | = (1 << 3);
    * (int *) 0x40020C00 | = (0x55 << 24);

    while (1)
    {
        * (int *) 0x40020C14 = (1 << 15);
        * (int *) 0x40020C14 = (0 << 15);
    }
    return 0;
}
```

In this case, the program runs at the speed of the AHB1 bus (tens of MHz) so the blink effect can not be seen. For it to be visible, the while () loop has to be slowed down several million times. This can be done with a “for ()” cycle after each instruction in the loop, as in the program:

```
int main()
{
    *(int *)0x40023830|= (1<<3);
    *(int *)0x40020C00|= (0x55<<24);
    while(1)
    {
        *(int *)0x40020C14 = (1<<15);
        for (int i=0;i<1000000;i++);
        *(int *)0x40020C14 = (0<<15);
        for (int i=0;i<1000000;i++);
    }
    return 0;
}
```

In fact, it is convenient to define the for () cycle as a function of a delay variable, called where necessary, as in the program:

```
void wait(int dt)
{
    for (int i=0;i<dt;i++);
}
int main()
{
    *(int *)0x40023830|= (1<<3);
    *(int *)0x40020C00|= (0x55<<24);
    while(1)
```

```

    {
        *(int *)0x40020C14|= (1<<15);
        wait(1000000);
        *(int *)0x40020C14&= ~(1<<15);
        wait(1000000);
    }
    return 0;
}

```

Note that to bring a bit into 0, you can use the compound operator &= in conjunction with the negation operator ~.

Experienced programmers can see that this program would not be right if optimization were to be done. The variable should be declared "volatile", otherwise the compiler would eliminate the delay if the optimization level (Project-> Options-> C / C ++ Compiler-> Optimizations) would be High, so we must set it to Low. You should also set the clock frequency, AHB1 bus speed, output resistors and output type, but here you can go to the default settings.

Also, the statement return 0 and the warning message that appears at the first compilation can be removed if the main loop is declared void.

The while loop can be made simpler using the composite operator ^= reversing the current state by XOR:

```

while(1)
{
    *(int *)0x40020C14^= (1<<15);
    wait(1000000);
}

```

-----

## 1.6 Inclusion of the constant and macro libraries declared by the manufacturer and the use of CMSIS

Using numeric expressions for pointers can make the program difficult to write and track. This can be avoided by defining them with mnemonic expressions, using #define directives and as mnemonics just the names in the documentation:

```

#define RCC_AHB1ENR *(int *)0x40023830
#define GPIOD_MODER *(int *)0x40020C00
#define GPIOD_ODR *(int *)0x40020C14

void wait(int dt)
{
    for (int i=0;i<dt;i++);
}

```



```

void main()
{
    RCC_AHB1ENR |= (1<<3);
    GPIOD_MODER |= (0x55<<24);
    while(1)
    {
        GPIOD_ODR ^= (1<<15);
        wait(1000000);
    }
}

```

However, it would be necessary to find and write the numerical constants at least at the beginning of the program, which implies the difficult stage of searching in the Microcontroller documentation. To avoid this step, microcontroller manufacturers provide files that contain such definitions, which can be included as a program preprocessor.

**For the STM32F4 family, you can use the "stm32f4xx.h" header, which can be found in the "STM32F4-Discovery\_FW\_Vx.x.x" directory.** Be careful, choose the file that has this name, because there are others with slightly different names.

Copy the path of this file (from Properties) and enter Project-> Options-> C / C ++ Compiler-> Preprocessor-> Additional include directories (Figure 1.9).

Now you can delete the definitions with numeric expressions and replace the directive at the beginning of the program:

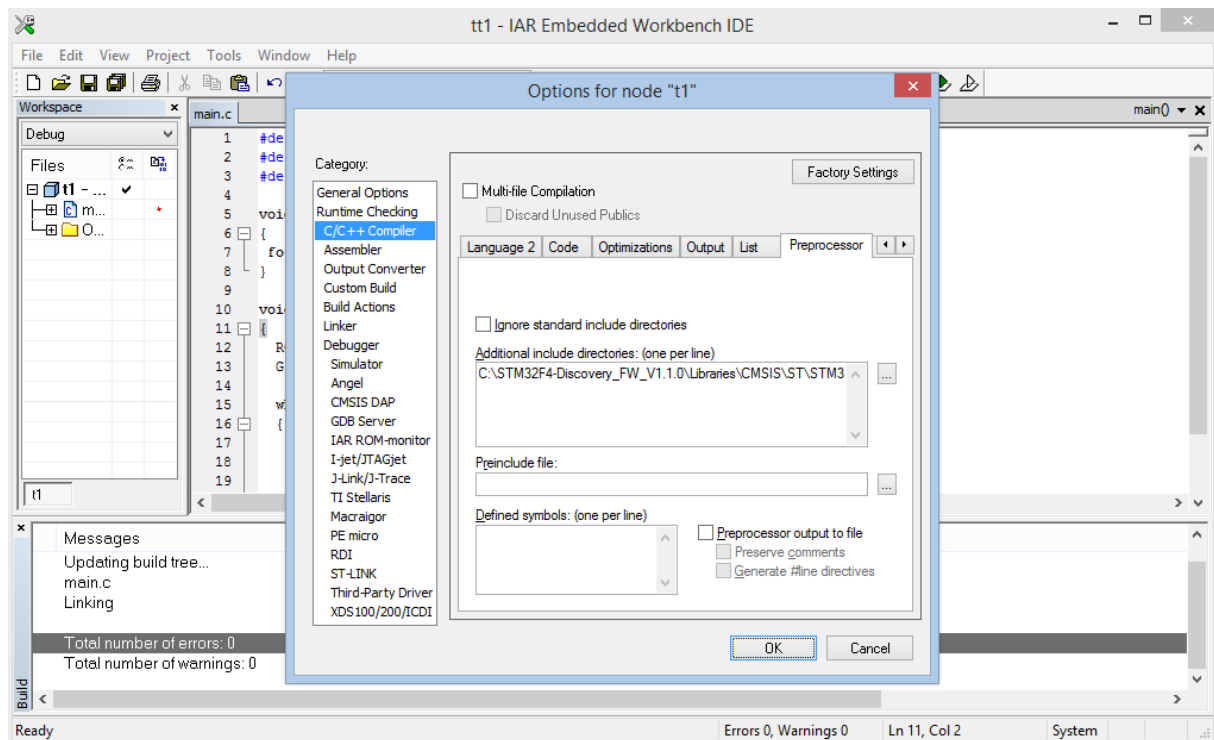
```
#include "stm32f4xx.h"
```

However, if you try to compile the program, an error will occur: the file "core\_cm4.h" is not found. A double click on this error tells us that this file is required in the file "stm32f4xx.h". We search for it in the "STM32F4-Discovery\_FW\_V1.1.0" directory and copy its path from its Properties. Add this path to a new line in Project-> Options-> C / C ++ Compiler-> Preprocessor-> Additional include directories, and rebuild the compilation. For example, these paths may be the following:

```

C:\STM32F4-Discovery_FW_V1.1.0\Libraries\CMSIS\ST\STM32F4xx\Include
C:\STM32F4-Discovery_FW_V1.1.0\Libraries\CMSIS\Include

```



**Figure 1.9.** Inserting paths to files included in the IAR-EWARM project

In a new compilation attempt, we find that the identifiers `RCC_AHB1ENR`, `GPIO_MODER` and `GPIO_ODR` are not defined. By looking at the "stm32f4xx.h" file, for example `RCC`, after several attempts it is found that such blocks are defined by the "typedef struct" directive. The members of such an object can be selected using the "dereference" `->` operator, which represents a pointer to a member of a structure.

In fact, by typing in the main `RCC->` a window will appear with the members of the `RCC` object and `AHB1ENR` is selected (if it does not appear, the program will be saved, will be closed and then reloaded). The same is done with `GPIO`, and the program becomes the following:

```
#include "stm32f4xx.h"

void wait(int dt)
{
    for (int i=0;i<dt;i++);
}
void main()
{
    RCC->AHB1ENR |= (1<<3);
    GPIO->MODER |= (1<<2*15) | (1<<2*14) | (1<<2*13) | (1<<2*12) | ;
    while(1)
    {
        GPIO->ODR ^= (1<<15);
        wait(1000000);
    }
}
```

This registry programming procedure, specifying the bit names as defined by the header files provided by the microcontroller manufacturer, is commonly referred to as the Cortex Microcontroller Software Interface Standard (CMSIS). The advantage is that it is not necessary to identify the bits in the microcontroller's manual, just copying the respective name in the header and even generating them semi-automatically by drop-down and autocomplete.

As you may see, CMSIS programming can be done even when using HAL-API functions, which in principle attempt to mask the hardware aspects. Thus, even projects automatically generated by CubeMX (which will be presented later) include the CMSIS library, from which definitions and macroinstructions can be retrieved.

It is worth mentioning here that CMSIS programming leads to the maximum possible performance, allows detailed tracking of all aspects of the program and is stable in the sense that the manufacturer will not change the definitions for the microcontroller. By contrast, high-level programming (such as SPL-API or HAL-API) is much easier, but performance is lower (low speed and high memory consumption), tracking details is more difficult, and are possible frequent changes of the set by the manufacturer.

Considering these differences between programming variants, in this paper most examples will be presented in both CMSIS and HAL-API, sometimes with their mixing and bits highlighting in the registers involved.

## 1.7 Use of CMSIS in STM32Cube\_FW\_F4\_Vx.x.x

As mentioned above, ST Microelectronics recommends for new projects the use of the integrated CubeMX environment, which can be downloaded from [www.st.com/stm32cubemx](http://www.st.com/stm32cubemx).

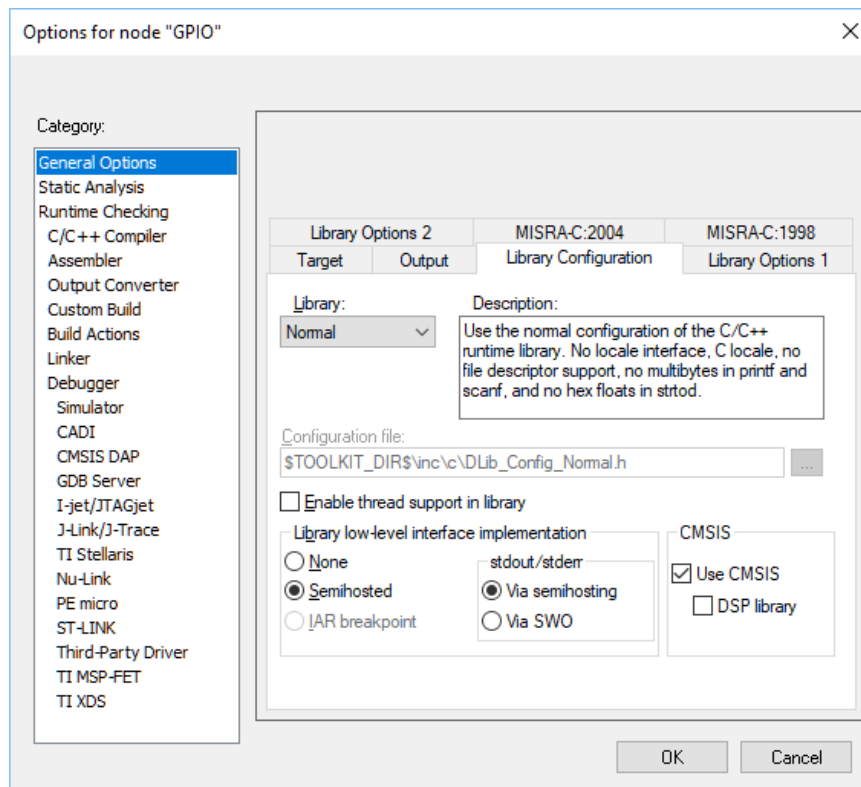
After installing it, a directory called \STM32Cube\Repository appears in the path indicated by the installation, where the current version of the library, STM32Cube\_FW\_F4\_Vx.x.x is located.

Instead of the old version of the STM32-Discovery firmware system, the STM32Cube CMSIS library, currently recommended by the manufacturer, can be used.

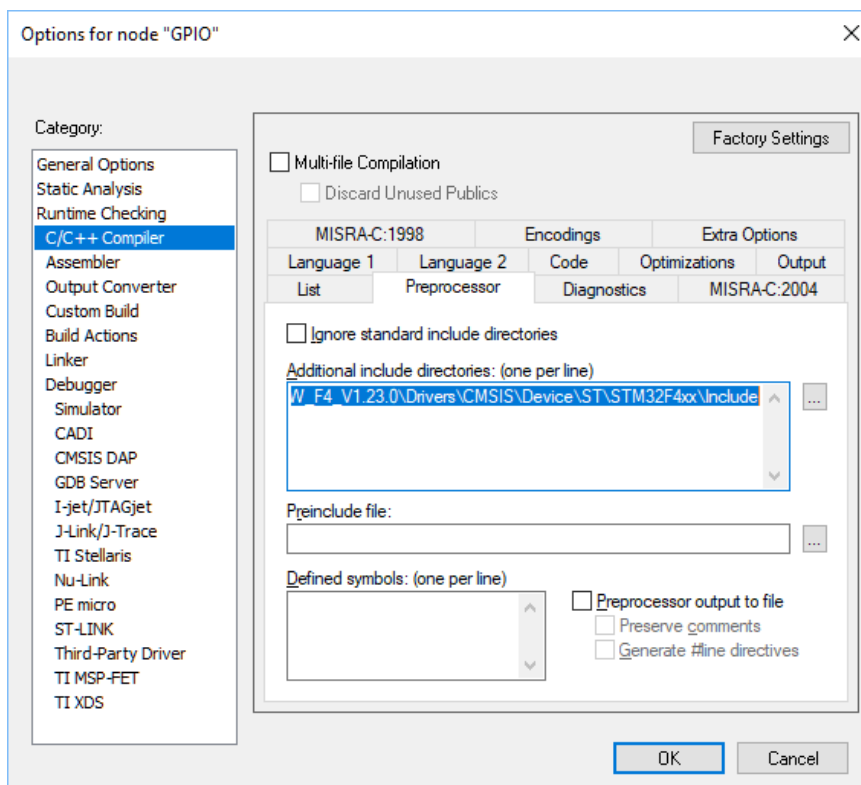
To do so, after writing the program to IAR-EWARM, Project-> Options-> General Options open Library Configuration tab and check "Use CMSIS", as in Figure 1.10.

Also, in the Project-> Options-> C / C ++ compiler-> Preprocessor, insert the path to the directory containing the new STM32Cube headers, for example ... \ STM32Cube \ Repository \ **STM32Cube\_FW\_F4\_V1.23.0** \ Drivers \ CMSIS \ Device \ ST \ STM32F4xx \ Includes, as in figure 1.11.

If the CMSIS is checked, no further settings are needed, instead you need to specify the type of microcontroller in the program by **#include "stm32F407xx.h"**, which will replace the old general directive **#include "stm32Fxx.h"**.



**Figure 1.10.** Enable CMSIS in the IAR-EWARM project



**Figure 1.11.** Inserting paths to CMSIS files in STM32Cube

With these changes, the program for blink led becomes the following:

```

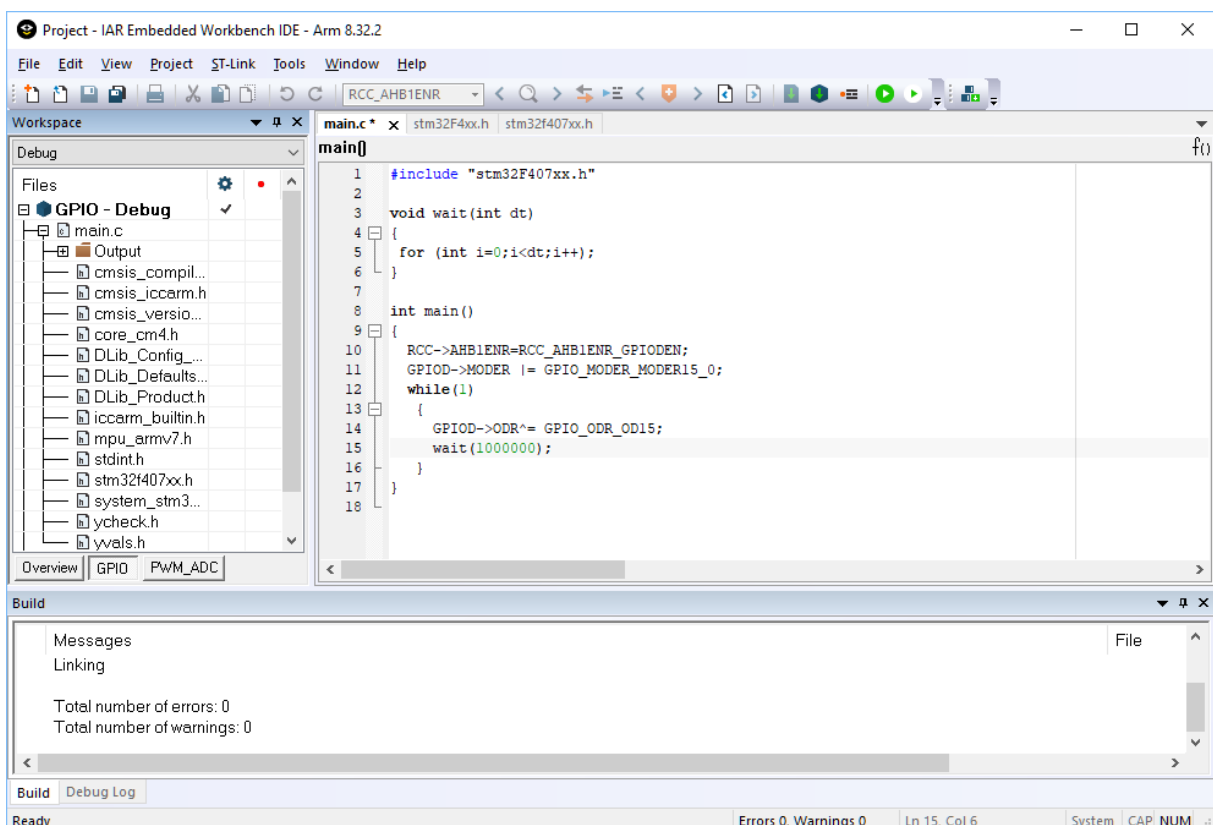
#include "stm32f407xx.h"

void wait(int dt)
{
    for (int i=0;i<dt;i++);
}

void main()
{
    RCC->AHB1ENR |= (1<<3);
    GPIOD->MODER |= (1<<2*15) | (1<<2*14) | (1<<2*13) | (1<<2*12) | ;
    while(1)
    {
        GPIOD->ODR ^= (1<<15);
        wait(1000000);
    }
}

```

It is possible to write the program without looking for the bit positions in the registers in the reference manual, but using only the definition file. By looking at the GPIO portion, all the definitions can be found: the clock validation bits, the mode settings, output data register etc. The program can be written in CMSIS, as in figure 1.12.



**Figure 1.12.** The STM32Cube CMSIS variant of the blink program

## 2. Programming with API Functions, Standard Peripheral Library (SPL)

### 2.1 General API programming features

As you may see, the programming style presented above is based on direct writing in the registers. This allows a great flexibility and total control over the system, but has the disadvantage that it is necessary to know (or find) all the hardware details involved, and a vision in terms of logical signals. It is generally preferred by those with hardware knowledge (such knowledge is nevertheless an important advantage for embedded systems, so it is advisable to read as much as possible, in the microcontroller reference manual, for each block used).

For a programmer who prefer a more formal style, it is possible to use API (Application Programming Interface) functions in an Object-Oriented Programming context. This method can have the advantage of using the hardware abstraction layer (HAL), a concept that already is in a mature stage, that allows for greater portability of programs and largely eliminates the need to know the hardware of the system. Using HAL tools, the programmer only formally specifies the task to be executed, and the associated software tools (in this case the preprocessor) automatically generate hardware-dependent instructions.

However, for systems with ARM microcontrollers, given the large number of manufacturers and types, there is no general API standard, each manufacturer and even the class of devices having their own, inexpensive library, which implies a great initial learning effort. Therefore, there remains the advantage of a relative lack of necessity of the hardware knowledge of the microcontroller, as well as of writing the program with expressions closer to the human language.

We will continue to exemplify the programming mode based on the standard STI library of the ST Microelectronics manufacturer for part of the Cortex M4 class of microcontrollers.

Although the standard driver version SPL-API is not recommended for new projects, being replaced by HAL-API, it is much easier to understand in details, and is presented here for didactic considerations.

In the directory resulting from the installation of the firmware and development file system STM32F4-Discovery\_FW\_V1.1.0, there is a subdirectory named STM32F4xx\_StdPeriph\_Driver in the subdirectory of the Libraries, which contains source programs with API functions of the peripherals of this microcontroller.

### 2.2 Blink LED API LED, using Standard Peripheral Drivers

Because the RCC and GPIO blocks are used, the source programs corresponding to them must be added: "stm32f4xx\_rcc.c" and "stm32f4xx\_gpio.c", which are in the ...\\STM32F4-Discovery\_FW\_V1.1.0\\Libraries\\STM32F4xx\_StdPeriph\_Driver\\src directory.

They are added to the project by clicking on the project window and right-clicking Add (Figure 2.1), indicating in turn the two files in the path mentioned above.

Also, add the path to their headers by entering the following lines in Project-> Options-> C / C++ compiler-> Preprocessor (attention, C: \\ STM32F4-Discovery\_FW\_V1.1.0 may vary depending on the installation location and version of the system firmware and development files, STSW-STM32068):

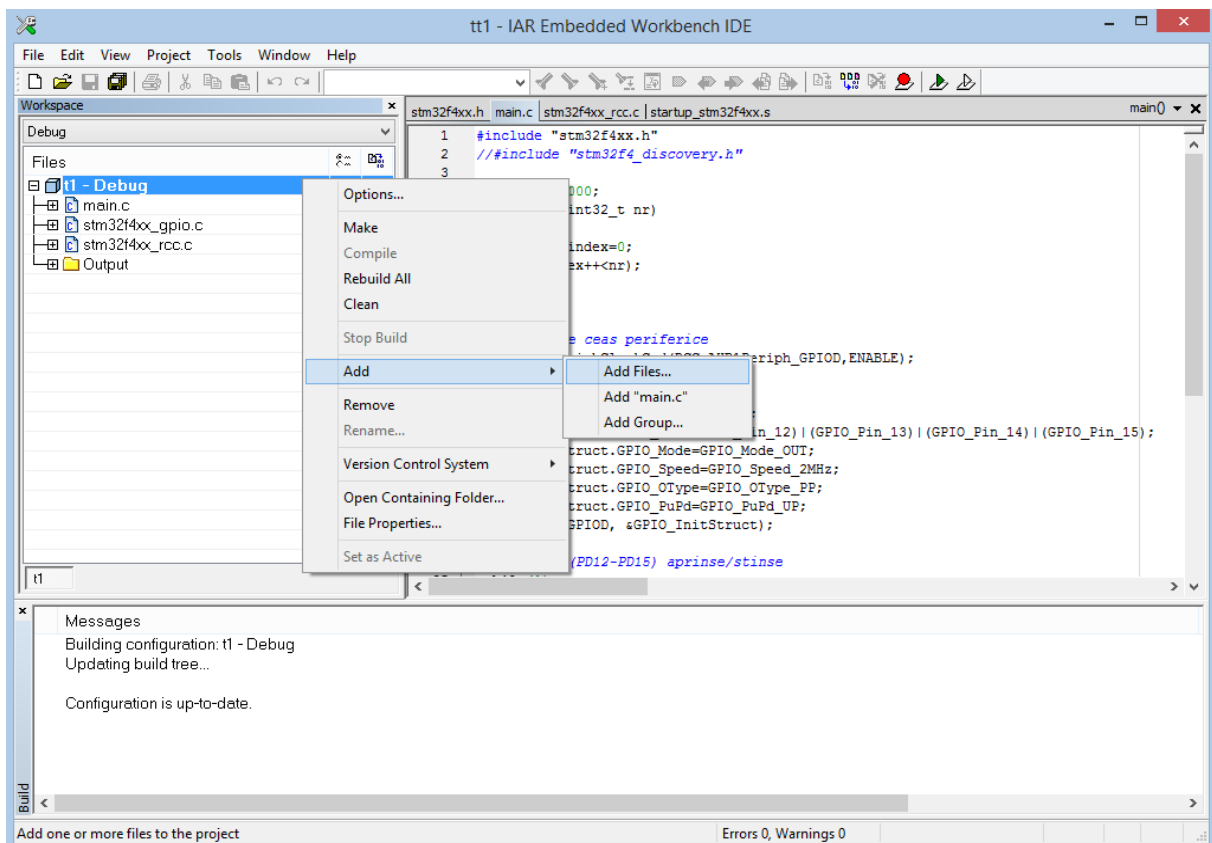
```
C:\\STM32F4-Discovery_FW_V1.1.0\\Libraries\\CMSIS\\ST\\STM32F4xx\\Include  
C:\\STM32F4-Discovery_FW_V1.1.0\\Libraries\\CMSIS\\Include
```

C:\STM32F4-Discovery\_FW\_V1.1.0\Project\FW\_upgrade\inc

C:\STM32F4-Discovery\_FW\_V1.1.0\Libraries\STM32F4xx\_StdPeriph\_Driver\inc

In addition, when using SPL-API functions, in the Preprocessor in the Defined symbols field always must be entered the line:

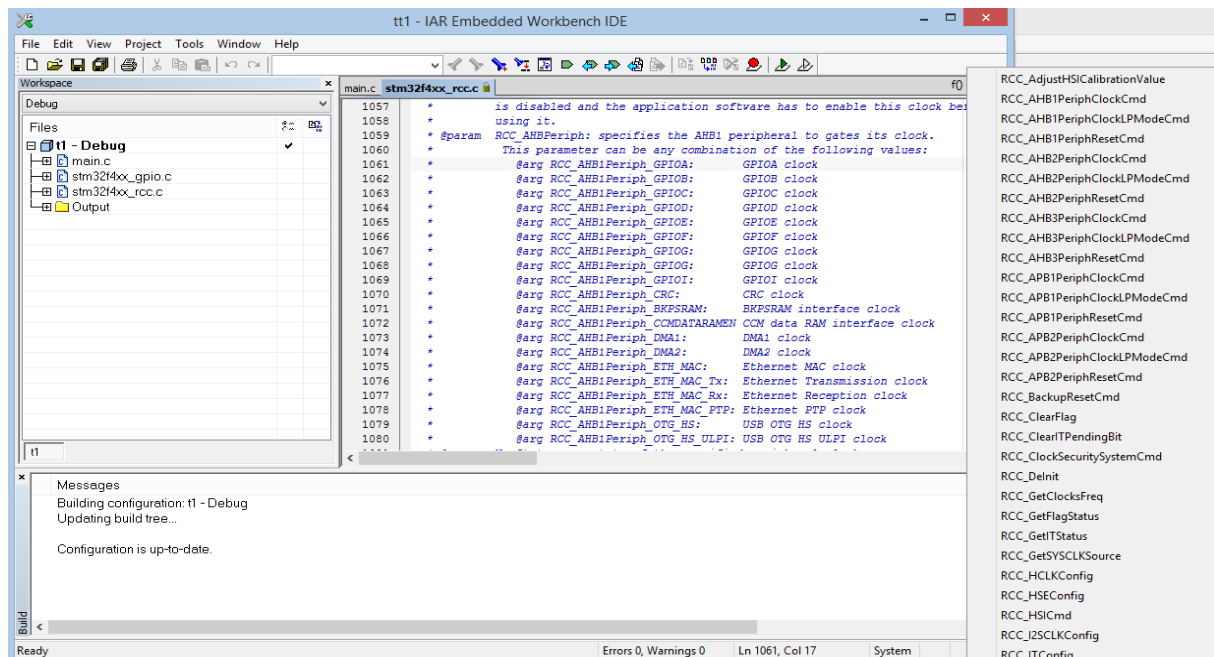
USE\_STDPERIPH\_DRIVER



**Figure 2.1.** Adding files to the IAR-EWARM project

Now you can double-click on the file "stm32f4xx\_rcc.c" and it appears in the main window. Next, click on the "f()" button in the top bar of the main window (Figure 2.2) where the "stm32f4xx\_rcc.c" tab is active, which will display the API functions for this peripheral.

Each of them can be viewed in the program with one click.



**Figure 2.2.** Visualization of API functions in the IAR-EWARM project

You can find the `RCC_AHB1PeriphClockCmd` function, whose description is above its statement: it can unlock the clock of the GPIO port that we are interested in. Also there is the syntax of use, so we can copy and paste/modify the line:

```
RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOD, ENABLE);
```

Also, by double clicking "stm32f4xx\_gpio.c" on the f() button, you can find the GPIO API functions in the "stm32f4xx\_gpio.c" file. They have quite suggestive names, and we will choose those to initialize, set and reset a bit, according to the syntactic explanations above the declaration.

To begin with, select the `GPIO_init` function which leads to its definition in "stm32f4xx\_gpio.c".

The `GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_InitStruct)` is copied in the main.c file (Figure 2.3 and according to the explanations the first parameter will be written as `GPIOD` and the second parameter is a pointer to a `GPIO_InitTypeDef` structure. Since the reference to a pointer is done in C with the `&` symbol, the line in the main that will complete the structures will be:

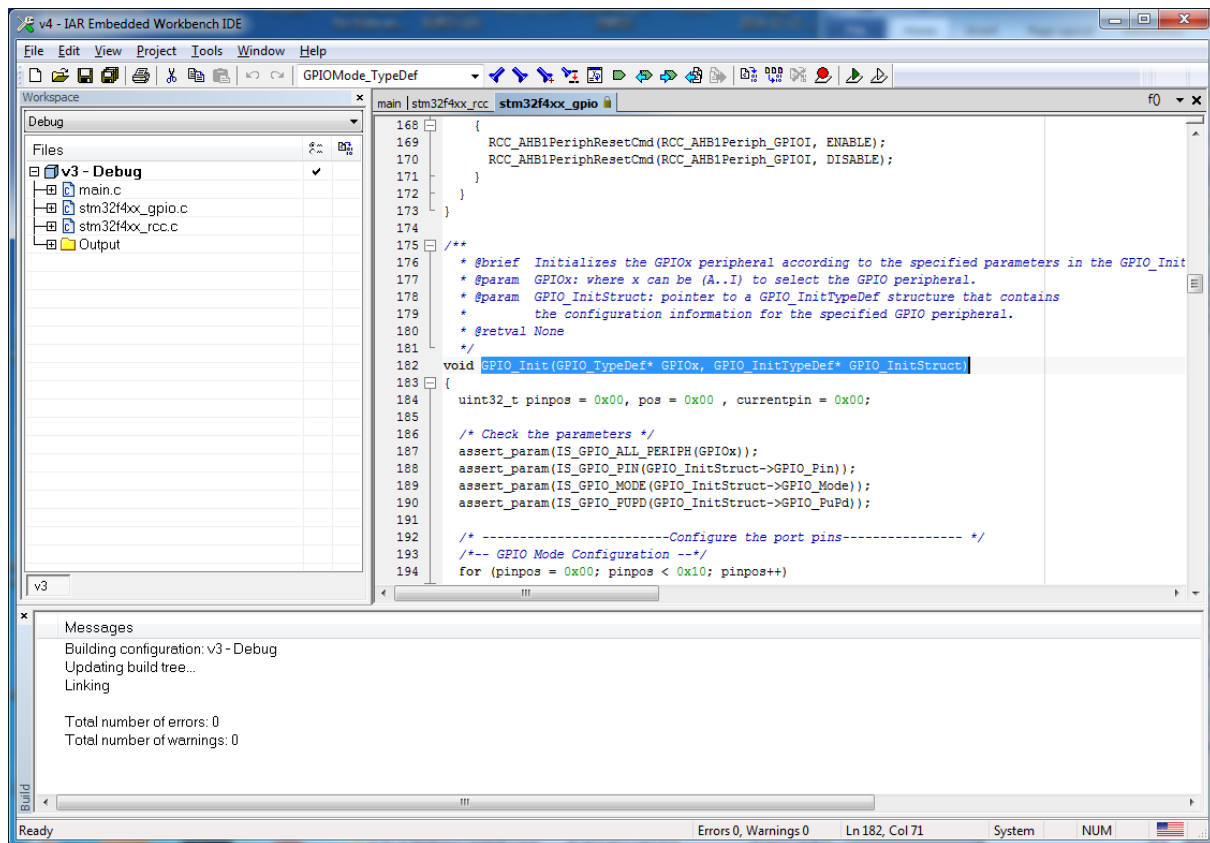
```
GPIO_Init(GPIOD, &GPIO_InitStruct);
```

To build the referenced structure, `GPIO_InitTypeDef`, return to "stm32f4xx\_gpio.c", place the cursor on the `GPIO_InitTypeDef` field and right click to select "Go to definition of `GPIO_InitTypeDef`". This leads us to the associated header file, "stm32f4xx\_gpio.h", where the definition of "typedef struct" is found.

It will then write above the previous line the structure that starts with the line:

```
GPIO_InitTypeDef GPIO_InitStruct;
```





**Figure 2.3.** Copying the prototype of an API function into the IAR-EWARM project

Next, you need to set the five members of the structure as defined in "stm32f4xx\_gpio.h": GPIO\_Pin, GPIO\_Mode, GPIO\_Speed, GPIO\_Otype, and GPIO\_PuPd. They are selected from the menu that appears after writing the GPIO\_InitStruct followed by the point. GPIO\_Pin is selected at the beginning, and "GPIO\_pins\_define" is copied from the "stm32f4xx\_gpio.h" structure and searched for by entering it in the search window (top bar immediately below Help, Figure 2.4). Here are the pin definitions and **copy** GPIO\_Pin\_15, which is connected to the blue LED. The line will then be:

```
GPIO_InitStruct.GPIO_Pin = GPIO_Pin_15;
```

Write on a new line GPIO\_InitStruct followed by point and select GPIO\_Mode. Searching again in the definition of the GPIO\_InitTypeDef structure in "stm32f4xx\_gpio.h", select GPIO\_Mode\_TypeDef and search with Ctrl + F.

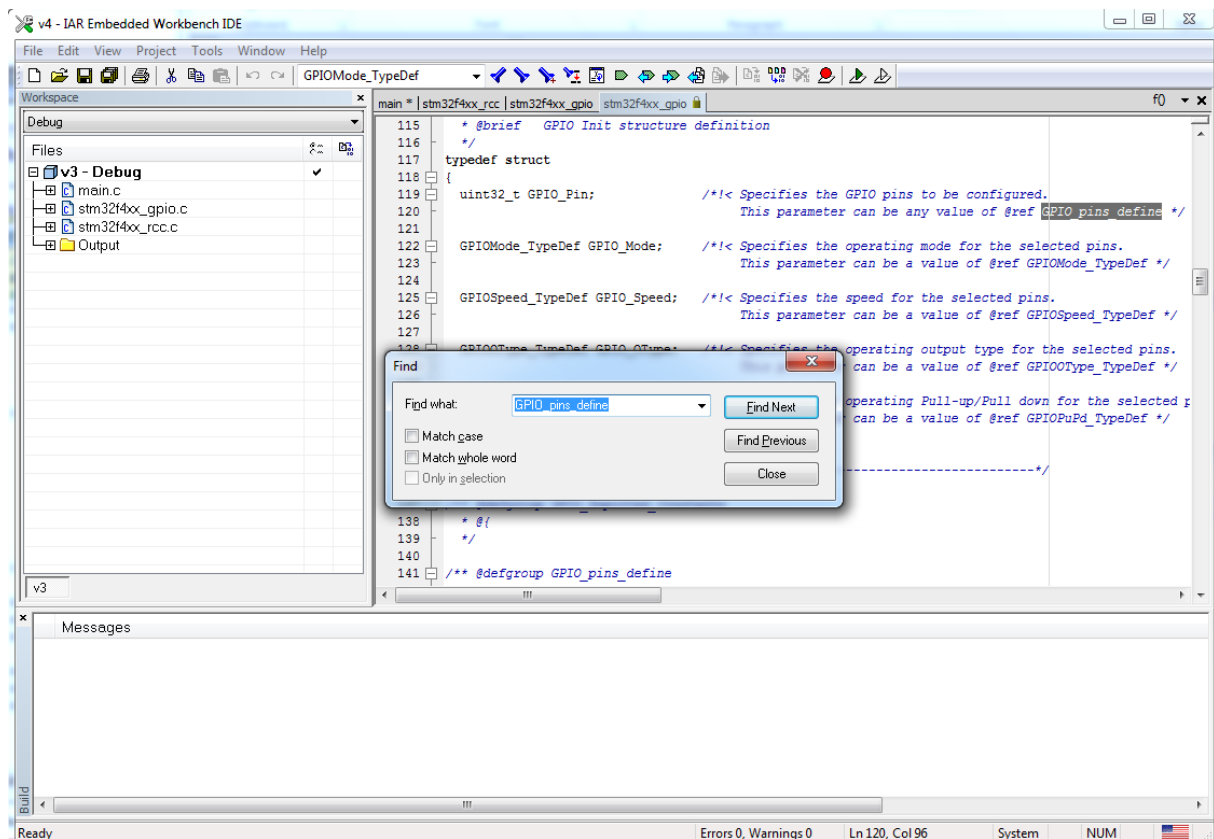
We look for the mode options, and because we need to use the port as output, the new line will look like this:

```
GPIO_InitTypeDef.GPIO_mode = GPIO_Mode_OUT;
```

The same is true for the other three parameters, such as the 2 MHz speed, the output type PP (ie Push-Pull) and the pull resistance UP (which means to Vdd).

The procedure described for viewing the function prototype, structure, and definitions can always be used when programming the API.

We do NOT recommend to directly write the program, because the error of writing is very high and, moreover, it would take much longer. It is more appropriate to copy all of the expressions as indicated, from the selected xx.c and xx.h files in the project window.



**Figure 2.4.** Search for parameter values of an API function

The program made in the previous chapter, that periodically turn on and off a LED can only be written with API functions as follows:

```
#include "stm32f4xx.h"

void wait(uint32_t u)
{
    while(u--);
}

int main()
{
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
    GPIO_InitTypeDef GPIO_InitStructure;
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_15|GPIO_Pin_14;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType=GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd=GPIO_PuPd_UP;
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_100MHz;
    GPIO_Init(GPIOD, &GPIO_InitStructure);

    while(1)
    {
        GPIO_SetBits(GPIOD,GPIO_Pin_15);
        wait(1000000);
    }
}
```

```
GPIO_ResetBits(GPIOD,GPIO_Pin_15);  
wait(10000000);  
}  
}
```

Obviously, in the while loop you could use a toggle API function :

```
GPIO_ToggleBits(GPIOD,GPIO_Pin_15);
```

It can be seen that in this form of the Blink project the period of the LED is about 1 second. If the system clock initialization system were to be thoroughly analyzed, it would be noted that it does not have the maximum speed, which may be an impediment in some applications. The maximum speed would be achieved through a rigorous programming of the clock, but this is a difficult operation at this stage because it requires a careful hardware study of the documentation.

Instead, to achieve a high clock frequency, two files must be added: "startup\_stm32f4xx.s" and "system\_stm32f4xx.c" and the paths to that header.

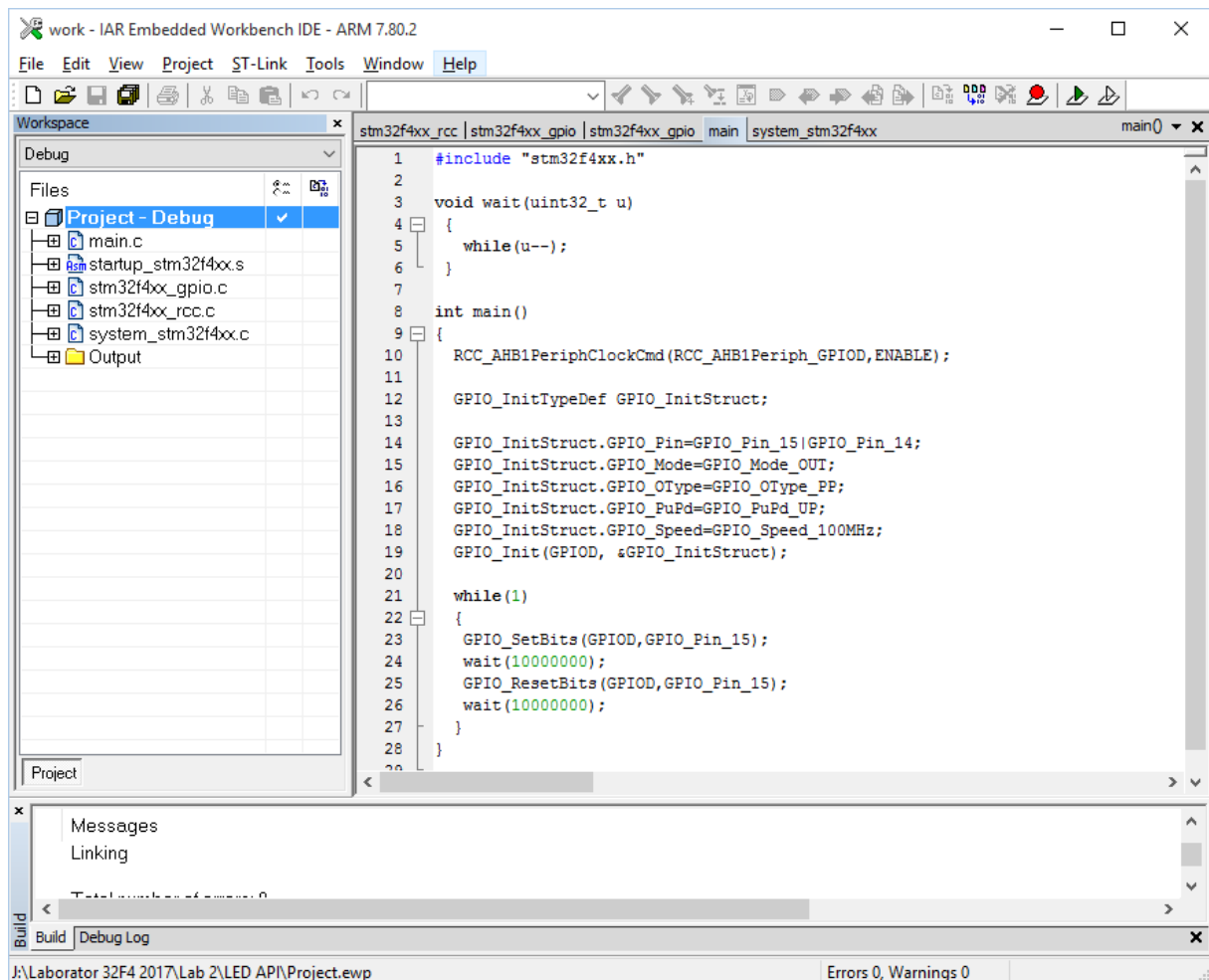
A study of the program system can indicate that there is a "startup\_stm32f4xx.s" file in the \STM32F4-Discovery\_FW\_V1.1.0\Libraries\CMSIS\ST\STM32F4xx\Source\Templates directory. We will add it to the project in this way as shown above and we will retry the compilation. An error will appear, indicating that the "system\_stm32f4xx.c" program, which is located in the C:\STM32F4-Discovery\_FW\_V1.1.0\Libraries\CMSIS\ST\STM32F4xx\Source\Templates directory, is required.

This program should be added to the project, but the compilation again shows an error requesting the path to the file "stm32f4xx.conf.h", so we will enter the path: C:\STM32F4-Discovery\_FW\_V1.1.0\Project\FW\_upgrade\inc.

By adding this path, the preprocessor will contain the following lines:

```
C:\STM32F4-Discovery_FW_V1.1.0\Libraries\CMSIS\ST\STM32F4xx\Include  
C:\STM32F4-Discovery_FW_V1.1.0\Libraries\CMSIS\Include  
C:\STM32F4-Discovery_FW_V1.1.0\Libraries\STM32F4xx_StdPeriph_Driver\inc  
C:\STM32F4-Discovery_FW_V1.1.0\Project\FW_upgrade\inc
```

The windows of the program will now look like in Figure 2.5.



**Figure 2.5.** The main.c file for the PD15 toggle program using API functions

Compiling and running the program finds that the LED period has decreased about 3-4 times, indicating that the system clock now works at a much higher frequency than the previous one, by default.

The syntax may seem rather difficult for registry-oriented programmers, but quite simple for object-oriented. The great advantages of this method will be exemplified in a future work on USART programming.

We mention that this way of working is difficult and time-consuming, because the following steps are needed:

1. Preparing the preprocessor by including paths to all header files involved and specifying the standard library for peripherals;
2. Properly initializing, in the main.c program, of all the peripheral blocks used, including possible interrupt subroutines;
3. The proper writing of the program with API functions.

That is why this variant, like the previous ones, was included here primarily for didactic reasons, for acquaintance with the details of the program, thus gaining control over all its details.

Nowadays, for the development of STM Cortex microcontroller programs, ST Microelectronics recommends using the CubeMX program, an integrated tool for quickly realizing the first two steps listed above. This application is free and generates both the project, all the links and the initialization code of all peripherals, allowing the focus on the programming side itself.