



FREE eBook

LEARNING

stm32

Free unaffiliated eBook created from
Stack Overflow contributors.

#stm32

Table of Contents

About.....	1
Chapter 1: Getting started with stm32.....	2
Remarks.....	2
What is STM32?.....	2
Product series.....	2
Development boards.....	2
Versions.....	3
Examples.....	3
First time setup with blink LED example using SW4STM32 and HAL library.....	3
IDE installation.....	3
Creating a project.....	3
Blink LED application.....	6
Chapter 2: Integrated development environments (IDEs).....	10
Introduction.....	10
Remarks.....	10
Examples.....	13
SW4STM32: System Workbench for STM32.....	13
Introduction.....	13
Installation.....	14
IAR-EWARM.....	14
Introduction.....	14
Installation.....	15
Atollic - TrueSTUDIO.....	15
Introduction.....	15
Installation.....	15
ColDE.....	15
Introduction.....	15
Installation.....	16
Chapter 3: UART - Universal Asynchronous Receiver/Transmitter (serial communication).....	17
Introduction.....	17

Examples.....	17
Echo application - HAL library.....	17
Transmit large amount of data using DMA and interrupts - HAL library.....	18
Credits	22

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [stm32](#)

It is an unofficial and free stm32 ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official stm32.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with stm32

Remarks

This section provides an overview of what stm32 is, and why a developer might want to use it.

It should also mention any large subjects within stm32, and link out to the related topics. Since the Documentation for stm32 is new, you may need to create initial versions of those related topics.

What is STM32?

STM32 is a 32-bit Flash microcontroller family developed by ST Microelectronics. It is based on the ARM® Cortex®-M processor and offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation.

A detailed description about each series, development tools and part number decoding can be found on [Wikipedia](#).

Product series

	Cortex-M0 / -M0+	Cortex-M3	Cortex-M4	Cortex-M7
High performance:		STM32F2	STM32F4	STM32F7 , STM32H7
Mainstream:	STM32F0	STM32F1	STM32F3	
Ultra-low-power:	STM32L0	STM32L1	STM32L4	

Development boards

	STM32 Nucleo (mbed enabled)	Discovery kits	Evaluation boards
Typical use case:	Flexible prototyping, community	Prototyping, creative demos	Full feature evaluation
Extension possibilities:	+++	++	++
Connectivity:	Arduino™, ST, Morpho	ST	ST

Versions

Version	Release Date
1.0.0	2016-11-01

Examples

First time setup with blink LED example using SW4STM32 and HAL library

(**Note:** There are many IDE, toolchain and library which are ready-to-use with STM32. The following setup requires minimal effort to get it work, but it is only one of the many. Feel free to explore others, it is not the purpose of this example to force anyone to use the tools that will be used here.)

IDE installation

[System Workbench for STM32](#): free IDE on Windows, Linux and OS X. It has been built by [AC6](#) and available for download after registration from the [OpenSTM32 Community's website](#).

The IDE itself is based on Eclipse, but comes with some extras for STM32 development like:

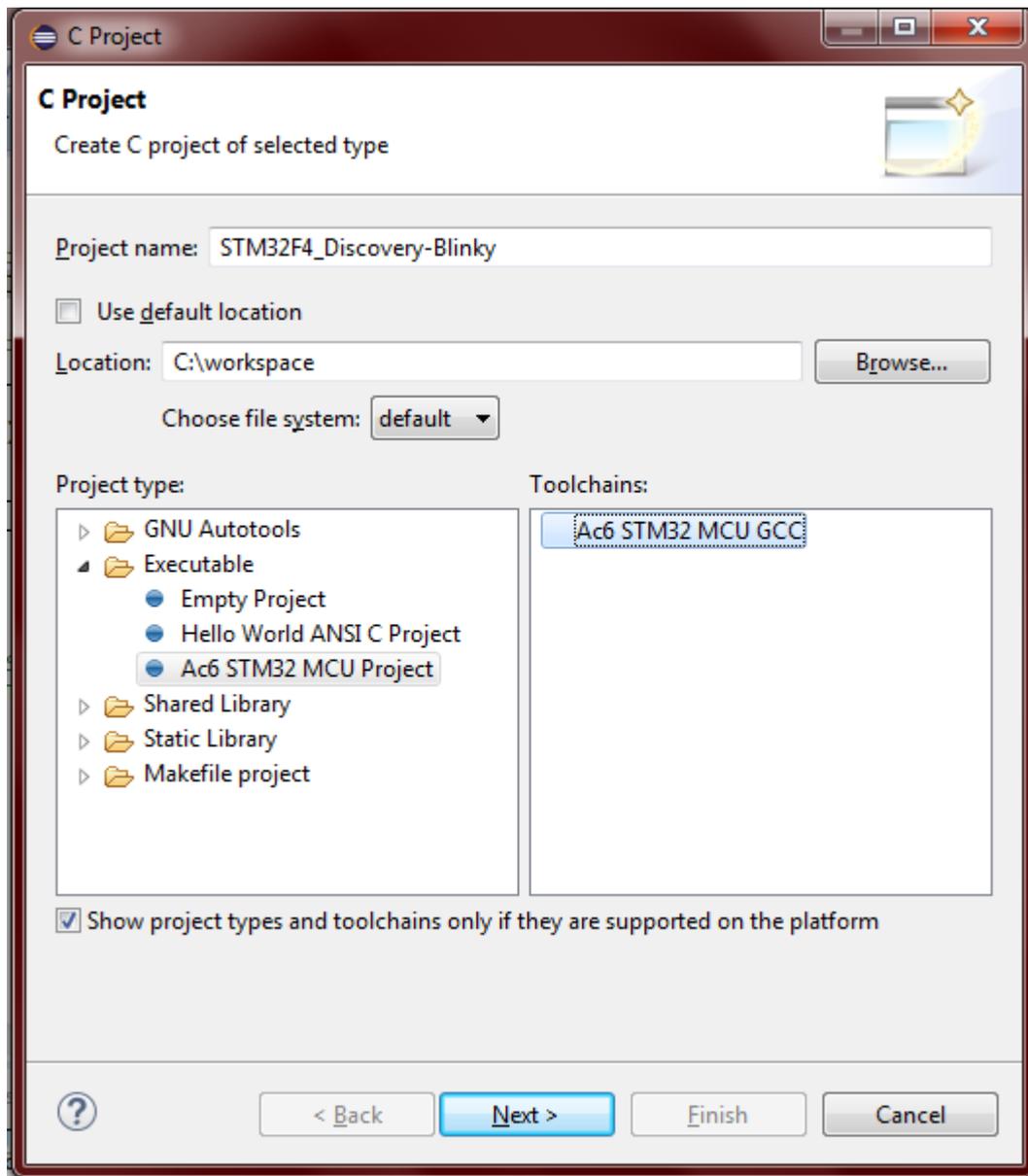
- Ac6 STM32 MCU GCC toolchain
- [OpenOCD](#) and GDB (arm-none-eabi-gdb) with automatically generated debug configurations depending on the target board
- Built-in options to program or erase chip

To start with STM32 before creating your own board, it is recommended to experiment with a [Discovery](#), a [Nucleo](#) or an [Eval board](#), which come with an on-board SWD (Serial Wire Debug) programmer/debugger called ST-Link.

Creating a project

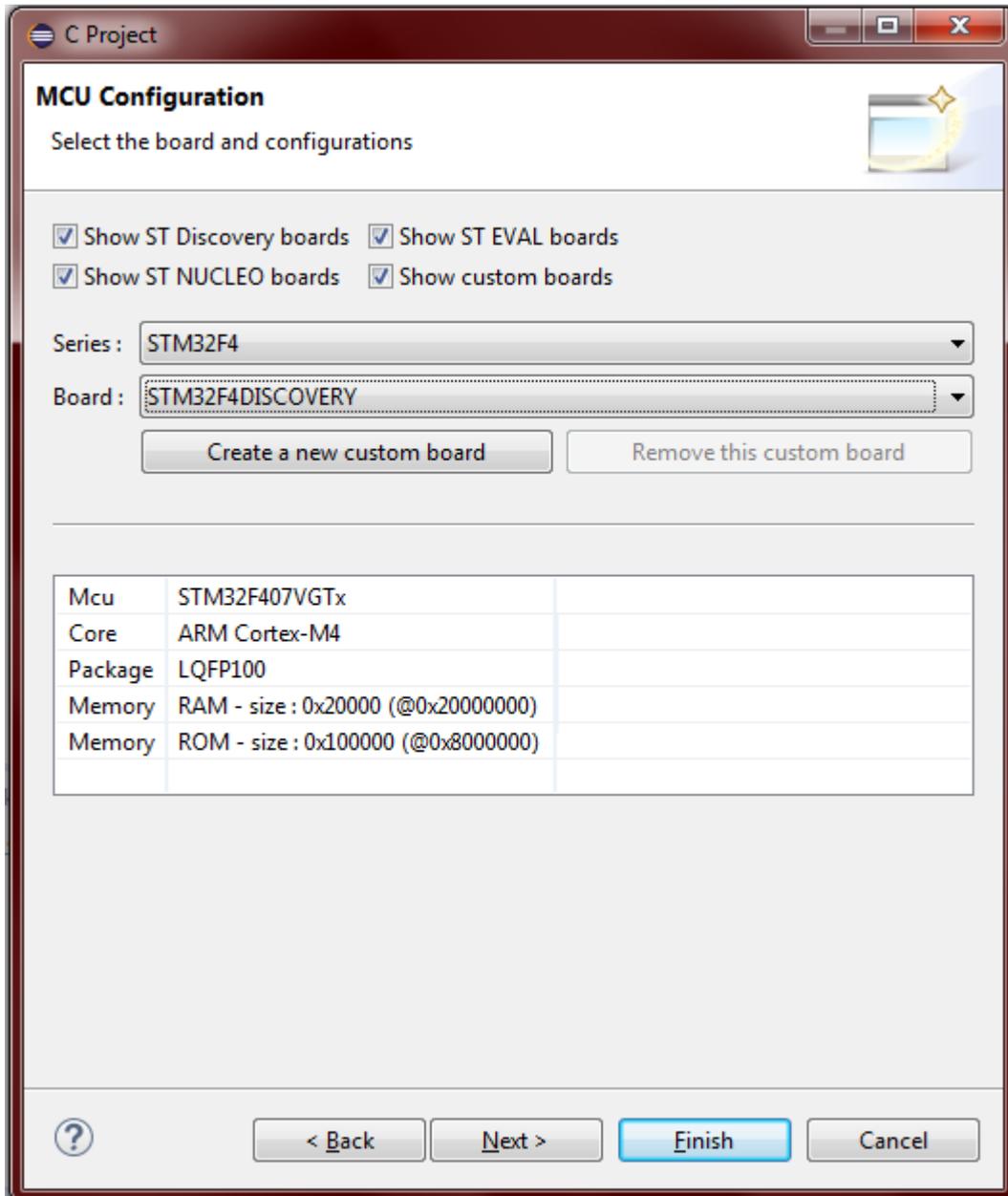
This example will use an [STM32F4 Discovery kit](#), which features an STM32F407VG microcontroller. (Any other board can be used as well.)

1. Open SW4STM32 and create a new C project: **File** → **New** → **C Project**
2. Give it a name like "*STM32F4_Discovery-Blinky*" and from the **Project Type** list choose the **Executable/Ac6 STM32 MCU Project**. By default the only available toolchain is **Ac6 STM32 MCU GCC**. Click Next.

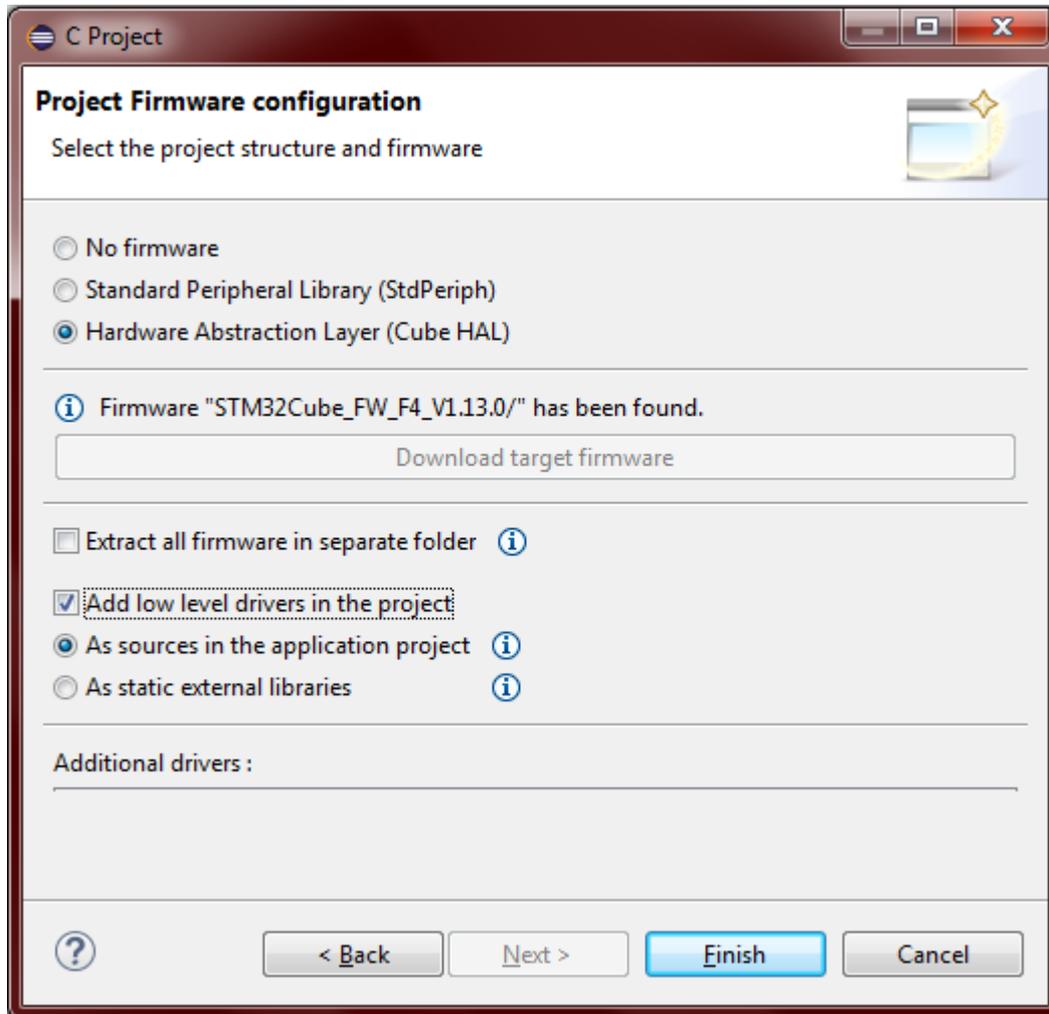


3. Next step is *Debug/Release settings*, can be skipped now by clicking Next.

4. *Board selection*. Existing boards can be selected as in this example the STM32F4 Discovery or new custom boards can be added.

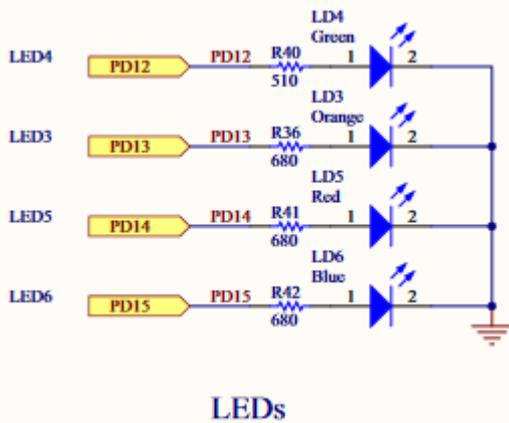


5. Next step is *Project Firmware configuration*. Choose between **No firmware**, **Standard Peripheral Library (SPL)** or **Hardware Abstraction Layer (HAL)**. It is questioned which one is more suitable for development, but this question is out of scope in this example. This example will use the HAL library as it is the currently supported by ST Microelectronics. Additional available software tool for HAL is [STM32CubeMX](#), which is an initialization code generator. Also several example applications are available by the [STM32CubeFx](#) or [STM32CubeLx](#) software packages. Download the target firmware if it's missing and it is recommended select the "**Add low level drivers in the project**" and the "**As sources in the application**" options. Finally, click Finish.



Blink LED application

As this project has been created with an STM32F4 Discovery, there are already several ready-to-use functions under the **/STM32F4_Discovery-Blinky/Utilities/STM32F4-Discovery/** project folder which can be used to interface the Discovery kit's peripherals (accelerometer, audio, LEDs, push button). In this example the `void BSP_LED_Init(Led_TypeDef Led)` and the `void BSP_LED_Toggle(Led_TypeDef Led)` functions will be used from the `stm32f4_discovery.c` file to blink the green LED, which is LED4. To decide which LED is which use the schematics of the [Discovery kit](#).



The actual pin and port names are already hidden by some `#define` and `enum`, use *Ctrl + Click* to track them.

1. Inside the `main`, call the `HAL_Init()` function which resets all peripherals, initializes the Flash interface and the SysTick. (SysTick will be used to generate delay for the blinking.)
2. The system clock have to be configured. It can be done by using the [STM32CubeMX clock configuration feature](#) or by the reference manual. In this example the system clock is fed by the internal PLL (Phase Locked Loop), which is sourced by an external 8 MHz crystal oscillator (HSE). Prescalers have been set to achieve the maximum available frequency, which is 168 MHz in case of the F4 Discovery.
3. Initialization of the peripherals, in this case a GPIO pin.
4. Inside an endless loop, call the LED toggling and the `HAL_Delay()` function. `HAL_Delay()` uses the `Systick` and generates a delay in milliseconds.

The whole code is the following:

```
#include "stm32f4xx.h"
#include "stm32f4_discovery.h"

void SystemClock_Config(void);

int main(void)
{
    /* Reset of all peripherals, Initializes the Flash interface and the SysTick. */
    HAL_Init();

    /* Configure the system clock */
    SystemClock_Config();

    /* Initialize one of the LED GPIO pin */
    BSP_LED_Init(LED4);

    while(1)
    {
        BSP_LED_Toggle(LED4);
        HAL_Delay(1000);    // in milliseconds
    }
}

/**
```

```

* @brief System Clock Configuration
*       The system Clock is configured as follow :
*       System Clock source           = PLL (HSE)
*       SYSCLK(Hz)                     = 168000000
*       HCLK(Hz)                       = 168000000
*       AHB Prescaler                   = 1
*       APB1 Prescaler                  = 4
*       APB2 Prescaler                  = 2
*       HSE Frequency(Hz)               = HSE_VALUE
*       PLL_M                           = (HSE_VALUE/1000000u)
*       PLL_N                           = 336
*       PLL_P                           = 2
*       PLL_Q                           = 7
*       VDD(V)                         = 3.3
*       Main regulator output voltage   = Scale1 mode
*       Flash Latency(WS)               = 5
* @param None
* @retval None
*/
void SystemClock_Config(void)
{
    RCC_ClkInitTypeDef RCC_ClkInitStruct;
    RCC_OscInitTypeDef RCC_OscInitStruct;

    // Enable Power Control clock
    __PWR_CLK_ENABLE();

    // The voltage scaling allows optimizing the power consumption when the
    // device is clocked below the maximum system frequency, to update the
    // voltage scaling value regarding system frequency refer to product
    // datasheet.
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    // Enable HSE Oscillator and activate PLL with HSE as source
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;

    // This assumes the HSE_VALUE is a multiple of 1MHz. If this is not
    // your case, you have to recompute these PLL constants.
    RCC_OscInitStruct.PLL.PLLM = (HSE_VALUE/1000000u);
    RCC_OscInitStruct.PLL.PLLN = 336;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
    RCC_OscInitStruct.PLL.PLLQ = 7;
    HAL_RCC_OscConfig(&RCC_OscInitStruct);

    // Select PLL as system clock source and configure the HCLK, PCLK1 and PCLK2
    // clocks dividers
    RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK
        | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
    HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5);
}

```

Build with the hammer , and download the application by right clicking on the project folder and selecting the **Target → Program chip...** option.

Another way to download is with using *debug*. To do so click on the arrow beside the bug icon



in the toolbar and open **Debug Configuration...** menu. Create a new **Ac6 STM32 Debugging** configuration and if the **C/C++ Application** field is empty, fill in the following:

```
Debug\STM32F4_Discovery-Blinky.elf
```

Other debug parameters such as the OpenOCD configuration file and the used Telnet and GDB ports are automatically generated and filled in by the framework. Finally, click the Debug button.

Read **Getting started with stm32 online**: <https://riptutorial.com/stm32/topic/7617/getting-started-with-stm32>

Chapter 2: Integrated development environments (IDEs)

Introduction

The purpose of this topic is to list all integrated development environments (IDE) that can be used to develop software for STM32 microcontrollers. The examples should contain: 1. List of the IDE's main features. 2. List of the operating systems supported by the IDE. 3. Installation process. 4. Additional configuration steps (if there are any).

Remarks

Listed IDEs by ST Microelectronics:

Part Number	General Description	Marketing Status	Supplier	Software Type
ColIDE	CooCox ColIDE, a free and highly-integrated software development environment for ARM Cortex MCUs	Active	CooCox	SW development suites
CosmicIDE	Cosmic ARM/Cortex "M" Cross Development Tools for STM32 Microcontroller	Active	Cosmic	SW development suites
CrossWorks	Rowley Associates CrossWorks, integrated development environment with JTAG Flash download and debug	Active	Rowley	SW development suites
DS-5	ARM Development Studio 5 (DS-5) provides best-in-class tools for the broadest range of ARM processor-based platforms	Active	ARM	SW development suites
EMP-Thunder	Emprog ThunderBench, fully integrated and well-	Active	Emprog	Firmware

Part Number	General Description	Marketing Status	Supplier	Software Type
	crafted development C/C++ tools for ARM Cortex			
Hitop5	Universal user interface, IDE and debugger for all Hitex development tools	Active	Hitex	SW development suites
IAR-EWARM	IAR Integrated development environment and optimizing C/C++ compiler for ARM Cortex-M	Active	IAR	SW development suites
MDK-ARM-STM32	MDK-ARM software development environment for Cortex-M based MCUs	Active	Keil	SW development suites
MULTI	GreenHills integrated development and debug environment for embedded applications using C and C++	Active	GreenHills Software	SW development suites
Men-Nucleus-SF	Nucleus SmartFit for STM32	Active	Mentor Graphics	Firmware
PER-Tracealyzer	Percepio run-time trace analyser for STM32 MCU	Active	Percepio	
PLSUDE-STM32	Debug and emulator platform with optimized Trace and Flash support for STM32 Cortex-M based MCU's by PLS development tools	Active	Pls	SW development suites
RIDE-STM32	Raisonance branded integrated development environment for STM32 MCUs	Active	Raisonance	SW development suites
SOMN-DRT-IDE	SOMNIUM DRT Cortex-M IDE	Active	SOMNIUM	SW development

Part Number	General Description	Marketing Status	Supplier	Software Type
				suites
SW4STM32	System Workbench for STM32: free IDE on Windows, Linux and OS X	Active	AC6	SW development suites
TASKINGVX-STM32	Altium's C/C++ compiler and debugger tools for ARM based MCUs	Active	TASKING	Firmware
TrueSTUDIO	The premier C/C++ development tool for STM32 development, with its unrivalled feature set and unprecedented integration	Active	Atollic	SW development suites
iSYS-winIDEAOpen	iSYSTEM's free unlimited software development platform for all STM32 Cortex-M based devices	Active	iSYSTEM	SW development suites
mikroBasicPRO	MikroElektronika full-featured Basic compiler which makes STM32 development suitable for everyone	Active	Mikroelectronika	SW development suites
mikroCPRO	MikroElektronika full-featured ANSI C compiler for STM32 devices. It features an intuitive IDE, powerful compiler with advanced optimizations	Active	Mikroelectronika	SW development suites
mikroPascalPRO	MikroElektronika full-featured Pascal compiler for STM32 devices. It has an intuitive IDE with docking support, rich with features, advanced text editor, many available tools, libraries and	Active	Mikroelectronika	SW development suites

Part Number	General Description	Marketing Status	Supplier	Software Type
examples				
winIDEA-STM32	iSYSTEM's complete software development and test solution for the STM32 MCUs	Active	iSYSTEM	Firmware

Examples

SW4STM32: System Workbench for STM32

Introduction

System Workbench for STM32 is a free IDE on Windows, Linux and OS X. Description from [ST Microelectronics](#):

The System Workbench toolchain, called SW4STM32, is a free multi-OS software development environment based on Eclipse, which supports the full range of STM32 microcontrollers and associated boards.

The SW4STM32 toolchain may be obtained from the website www.openstm32.org, which includes forums, blogs, and trainings for technical support. Once registered to this site, users will get installation instructions at the Documentation > System Workbench page to proceed with the download of the free toolchain.

The System Workbench toolchain and its collaborative website have been built by AC6, a service company providing training and consultancy on embedded systems.

This product is supplied by a third party not affiliated to ST. For the latest information on the specification, refer to the third party's website: www.ac6.fr.

Key Features

- Comprehensive support for STM32 microcontrollers, STM32 Nucleo boards, Discovery kits and Evaluation boards, as well as STM32 firmware (Standard Peripheral library or STM32Cube HAL)
- GCC C/C++ compiler
- GDB-based debugger
- Eclipse IDE with team-work management
- Compatible with Eclipse plug-ins
- ST-LINK support
- No code size limit
- Multiple OS support: Windows®, Linux and OS X®

Installation

1. Go to: <http://www.openstm32.org/HomePage> .
2. Register and log in to the site.
3. Navigate to:
<http://www.openstm32.org/Downloading+the+System+Workbench+for+STM32+installer> .
4. Download the latest version for you operating system.
5. Run the downloaded installer.

IAR-EWARM

Introduction

IAR Integrated development environment and optimizing C/C++ compiler for ARM Cortex-M.
Description from [ST Microelectronics](#):

The IAR-EWARM is a software development suite delivered with ready-made device configuration files, flash loaders and 4300 example projects included. IAR Embedded Workbench is compatible with other ARM®EABI compliant compilers and supports the following ARM®cores for STM32:

Key Features

1. Key components:
 - Integrated development environment with project management tools and editor
 - Highly optimizing C and C++ compiler for ARM®
 - Automatic checking of MISRA C rules (MISRA C:2004)
 - ARM® EABI and CMSIS compliance
 - Extensive HW target system support
 - Optional I-jet and JTAGjet™-Trace in-circuit debugging probes
 - Power debugging to visualize power consumption in correlation with source code
 - Run-time libraries including source code
 - Relocating ARM® assembler
 - Linker and librarian tools
 - C-SPY® debugger with ARM® simulator, JTAG support and support for RTOS-aware debugging on hardware
 - RTOS plugins available from IAR Systems and RTOS vendors
 - Over 3100 sample projects for evaluation boards from many different manufacturers
 - User and reference guides in PDF format
 - Context-sensitive on-line help
2. Chip-specific support:
 - 4300 example projects including for STMicroelectronics evaluation boards
 - Support for 4 Gbyte applications in ARM® and Thumb® mode

- Each function can be compiled in ARM® or Thumb® mode
 - VFP Vector Floating Point co-processor code generation
 - Intrinsic NEON™ support
3. Hardware debugging support:
 - STMicroelectronics ST-LINK V2 : Supports STM32 devices
 - STMicroelectronics ST-LINK : Supports STM32 devices
 4. RTOS support: consult IAR's web site <http://www.iar.com>
 5. Supported devices: consult IAR's web site <http://www.iar.com>

Installation

Atollic - TrueSTUDIO

Introduction

C/C++ IDE for ARM development.

Atollic TrueSTUDIO® is tested and verified on the following Operating Systems:

- Microsoft® Windows® Vista (32-bit version)
- Microsoft® Windows® Vista (64-bit version)
- Microsoft® Windows® 7 (32-bit version)
- Microsoft® Windows® 7 (64-bit version)
- Microsoft® Windows® 8 (64-bit version)
- Microsoft® Windows® 10 (64-bit version)
- Linux support expected end of 2016 Q4
- Mac OS X support expected 2017 Q2

TrueSTUDIO is only available as a **32-bit** application.

Installation

The Atollic TrueSTUDIO product is delivered as an executable installer. Please ensure that the user account, from which the installer is launched, has administrative privileges. There is no need for registration or internet connection during the installation. When TrueSTUDIO is installed it will run in Lite mode if no licenses are detected.

1. Go to: <http://atollic.com/resources/downloads/> .
2. Download the latest stable or the latest beta version.
3. Run the installer.

CoIDE

Introduction

CooCox CoIDE, a free and highly-integrated software development environment for ARM Cortex MCUs. Description from [ST Microelectronics](#):

CoIDE is a free software development environment based on Eclipse and GCC tool chain, which has been customized and simplified to give users an easy access to ARM® Cortex®-M microcontrollers.

This product is supplied by a third party not affiliated to ST. For complete and latest information on the specification and packages of the purchased parts, refer to the third party's website www.coocox.org.

Key Features

- Complete support for STM32 microcontrollers, STM32 Nucleo boards as well as STM32Cube software libraries.
- GCC C/C++ compiler.
- GDB-based debugger.
- Simplified Eclipse IDE.
- ST-Link support.
- Multi-language support: English, Chinese.

Installation

Read [Integrated development environments \(IDEs\) online](#):

<https://riptutorial.com/stm32/topic/7741/integrated-development-environments--ides->

Chapter 3: UART - Universal Asynchronous Receiver/Transmitter (serial communication)

Introduction

This topic is about serial communication using the Universal Asynchronous Receiver/Transmitter (UART) peripheral of the STM32 microcontrollers.

Examples

Echo application - HAL library

In this example the microcontroller echos back the received bytes to the sender using UART RX interrupt.

```
#include "stm32f4xx.h"

UART_HandleTypeDef huart2;

/* Single byte to store input */
uint8_t byte;

void SystemClock_Config(void);

/* UART2 Interrupt Service Routine */
void USART2_IRQHandler(void)
{
    HAL_UART_IRQHandler(&huart2);
}

/* This callback is called by the HAL_UART_IRQHandler when the given number of bytes are
received */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if (huart->Instance == USART2)
    {
        /* Transmit one byte with 100 ms timeout */
        HAL_UART_Transmit(&huart2, &byte, 1, 100);

        /* Receive one byte in interrupt mode */
        HAL_UART_Receive_IT(&huart2, &byte, 1);
    }
}

void uart_gpio_init()
{
    GPIO_InitTypeDef GPIO_InitStructure;

    __GPIOA_CLK_ENABLE();

    /**USART2 GPIO Configuration
    PA2      -----> USART2_TX
```

```

PA3      -----> USART2_RX
*/
GPIO_InitStruct.Pin = GPIO_PIN_2 | GPIO_PIN_3;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_LOW;
GPIO_InitStruct.Alternate = GPIO_AF7_USART2;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}

void uart_init()
{
    __USART2_CLK_ENABLE();

    huart2.Instance = USART2;
    huart2.Init.BaudRate = 115200;
    huart2.Init.WordLength = UART_WORDLENGTH_8B;
    huart2.Init.StopBits = UART_STOPBITS_1;
    huart2.Init.Parity = UART_PARITY_NONE;
    huart2.Init.Mode = UART_MODE_TX_RX;
    huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart2.Init.OverSampling = UART_OVERSAMPLING_16;
    HAL_UART_Init(&huart2);

    /* Peripheral interrupt init*/
    HAL_NVIC_SetPriority(USART2_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(USART2_IRQn);
}

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

    uart_gpio_init();
    uart_init();

    HAL_UART_Receive_IT(&huart2, &byte, 1);

    while(1)
    {

    }
}

```

This example used an STM32F4 Discovery (STM32F407VG), GPIO and alternate function values should be changed according to the STM32 microcontroller in use.

Transmit large amount of data using DMA and interrupts - HAL library

In this example 2000 bytes will be transferred using DMA, **Transmit Half Complete** and **Transmit Complete** interrupts achieving the best performance.

The first half of the transmit buffer is loaded with new data by the CPU in the **Transmit Half Complete** interrupt callback while the second half of the buffer is being transmitted by the DMA in the background.

Then, in the **Transmit Complete** the second half of the transmit buffer is loaded by the new data by the CPU while the first half (previously updated) is being transmitted by the DMA in the background.

```
#include "stm32f4xx.h"

uint8_t dma_buffer[2000];
volatile uint8_t toggle = 0;

UART_HandleTypeDef huart2;
DMA_HandleTypeDef hdma_usart2_tx;

void uart_gpio_init()
{
    GPIO_InitTypeDef GPIO_InitStructure;

    __GPIOA_CLK_ENABLE();

    /**USART2 GPIO Configuration
    PA2      -----> USART2_TX
    PA3      -----> USART2_RX
    */
    GPIO_InitStructure.Pin = GPIO_PIN_2|GPIO_PIN_3;
    GPIO_InitStructure.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStructure.Pull = GPIO_PULLUP;
    GPIO_InitStructure.Speed = GPIO_SPEED_HIGH;
    GPIO_InitStructure.Alternate = GPIO_AF7_USART2;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStructure);
}

void uart_dma_init()
{
    /* DMA controller clock enable */
    __DMA1_CLK_ENABLE();

    /* Peripheral DMA init*/
    hdma_usart2_tx.Instance = DMA1_Stream6;
    hdma_usart2_tx.Init.Channel = DMA_CHANNEL_4;
    hdma_usart2_tx.Init.Direction = DMA_MEMORY_TO_PERIPH;
    hdma_usart2_tx.Init.PeriphInc = DMA_PINC_DISABLE;
    hdma_usart2_tx.Init.MemInc = DMA_MINC_ENABLE;
    hdma_usart2_tx.Init.PeriphDataAlignment = DMA_MDATAALIGN_BYTE;
    hdma_usart2_tx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;
    hdma_usart2_tx.Init.Mode = DMA_NORMAL;
    hdma_usart2_tx.Init.Priority = DMA_PRIORITY_LOW;
    hdma_usart2_tx.Init.FIFOMode = DMA_FIFOMODE_DISABLE;
    HAL_DMA_Init(&hdma_usart2_tx);

    __HAL_LINKDMA(&huart2,hdmatx,hdma_usart2_tx);

    /* DMA interrupt init */
    HAL_NVIC_SetPriority(DMA1_Stream6_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(DMA1_Stream6_IRQn);
}

void uart_init()
{
    __USART2_CLK_ENABLE();

    huart2.Instance = USART2;
```

```

huart2.Init.BaudRate = 115200;
huart2.Init.WordLength = UART_WORDLENGTH_8B;
huart2.Init.StopBits = UART_STOPBITS_1;
huart2.Init.Parity = UART_PARITY_NONE;
huart2.Init.Mode = UART_MODE_TX_RX;
huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart2.Init.OverSampling = UART_OVERSAMPLING_16;
HAL_UART_Init(&huart2);

/* Peripheral interrupt init*/
HAL_NVIC_SetPriority(USART2_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(USART2_IRQn);
}

/* This function handles DMA1 stream6 global interrupt. */
void DMA1_Stream6_IRQHandler(void)
{
    HAL_DMA_IRQHandler(&hdma_usart2_tx);
}

void USART2_IRQHandler(void)
{
    HAL_UART_IRQHandler(&huart2);
}

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
    uint16_t i;
    toggle = !toggle;

    for(i = 1000; i < 1998; i++)
    {
        if(toggle)
            dma_buffer[i] = '&';
        else
            dma_buffer[i] = 'z';
    }

    dma_buffer[1998] = '\r';
    dma_buffer[1999] = '\n';
}

void HAL_UART_TxHalfCpltCallback(UART_HandleTypeDef *huart)
{
    uint16_t i;

    for(i = 0; i < 1000; i++)
    {
        if(toggle)
            dma_buffer[i] = 'y';
        else
            dma_buffer[i] = '|';
    }
}

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

    uart_gpio_init();
}

```

```
uart_dma_init();
uart_init();

uint16_t i;

for(i = 0; i < 1998; i++)
{
    dma_buffer[i] = 'x';
}

dma_buffer[1998] = '\r';
dma_buffer[1999] = '\n';

while(1)
{
    HAL_UART_Transmit_DMA(&huart2, dma_buffer, 2000);
}
}
```

The example was written for an STM32F4 Discovery board (STM32F407VG). The appropriate DMA instance, UART-DMA channel, GPIO and alternate function settings should be changed according to the STM32 microcontroller in use.

Read [UART - Universal Asynchronous Receiver/Transmitter \(serial communication\)](https://riptutorial.com/stm32/topic/9707/uart---universal-asynchronous-receiver-transmitter--serial-communication-) online:
<https://riptutorial.com/stm32/topic/9707/uart---universal-asynchronous-receiver-transmitter--serial-communication->

Credits

S. No	Chapters	Contributors
1	Getting started with stm32	Bence Kaulics , Community
2	Integrated development environments (IDEs)	Bence Kaulics
3	UART - Universal Asynchronous Receiver/Transmitter (serial communication)	Bence Kaulics