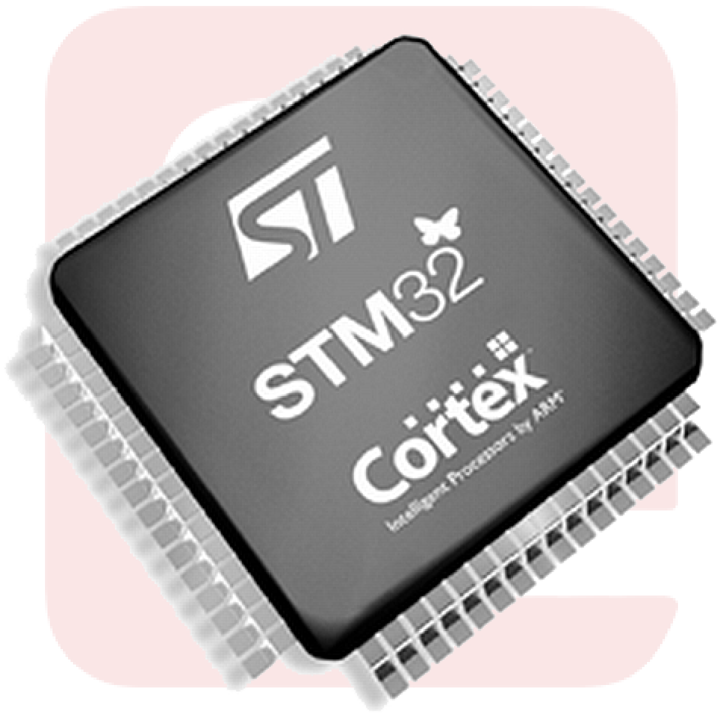
eYSIP 2017

Tutorial on STM32 Programming



Heethesh Vhavle
Sanam Shakya
Pushkar Raj

Duration of Internship: 22/05/2017 – 07/07/2017

2017, e-Yantra Publication

STM32 Programming

1.1 Introduction

The STM32 family of micro-controllers, based upon the ARM Cortex-M3 core, provides a foundation for building a vast range of embedded systems from simple battery powered dongles to complex real-time systems such as helicopter autopilots. This component family provides wide-ranging choices in memory sizes, available peripherals, performance, and power. Unfortunately, power and flexibility are achieved at a cost software development for the STM32 family can be extremely challenging for the uninitiated with a vast array of documentation and software libraries to wade through.[1]

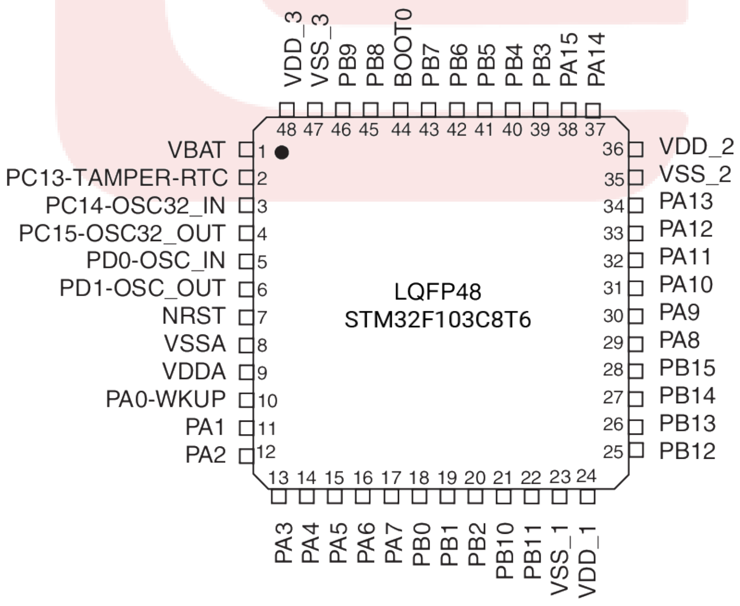


Figure 1.1: Pinout diagram of STM32F103C8T6 LQFP48 package

The STM32F103xx medium-density performance line family incorporates the high-performance ARM Cortex-M3 32-bit RISC core operating at a 72 MHz frequency, high-speed embedded memories (Flash memory up to 128 Kbytes and SRAM up to 20 Kbytes), and an extensive range of enhanced I/Os and peripherals connected to two APB buses. All devices offer two 12-bit ADCs, three general purpose 16-bit timers plus one PWM timer, as well as standard and advanced communication interfaces: up to two I2Cs and SPIs, three USARTs, an USB and a CAN.[2]

1.2 List of Hardware Components

To get started with STM32 micro-controllers, we will use the Nucleo Development board which provides an affordable and flexible way for users to try out new ideas and build prototypes. It has an on-board real-time debugger MCU called ST-LINK. This also makes the process of flashing the program easier. One can also use any other development board. Following is list of hardware choices.

- NUCLEO-F103RB STM32 Nucleo Development Board
  Datasheet — Chip Datasheet — Vendor Link

- STM32F103C8 Micro-controller
  Datasheet — Reference Manual — Guide Book — Vendor Link

1.3 Installation of Softwares and Tools

This section gives a brief overview on setting up the IDE and the various methods of flashing the program on to the micro-controller. The primary software resource is the GNU software development tool-chain including gcc, objcopy, objdump, and the debugger gdb. The GNU ARM Embedded toolchains are integrated and validated packages featuring the ARM Embedded GCC compiler, libraries and other GNU tools necessary for bare-metal software development on devices based on the ARM Cortex-M processors.[1, 3]

1.3.1 Integrated Development Environment (IDE)

Various different options are available for choosing and IDE such as ARM Keil MDK, EWARM, TrueSTUDIO and open source options such as the Eclipse IDE. The Eclipse IDE requires the tool-chain, debug and build tools to be installed manually. The tool-chains are available for cross-compilation on Microsoft Windows, Linux and Mac OS X host operating systems. The tool-chain can be downloaded from here.[3]

The IDE recommended for this tutorial is Attolic TrueSTUDIO, which is based on the Eclipse IDE. One of the main advantages is that TrueSTUDIO IDE already comes with the GCC tool-chains for ARM and the required debugging tools.

1.3.2 Debugging Tools

One of the key feature of Nucleo board is that it already provides the on-chip programmer for STM32 micro-controllers. In order to use this, ST-LINK drivers need to be installed.

Although TrueSTUDIO has built-in debugging tools, some of the features are not available in the free version of the software. Instead, we can use the official ST-LINK Utility tool.

1.3.3 Flashing the Program

There are three ways to upload the firmware on to the micro-controller. One of the easiest way is to simply generate a *binary (*.bin)* file, connect your board via USB and it should be detected as an external storage device. Program can be flashed by just dragging and dropping the binary file on the device.

The ST-LINK Utility tool also supports flashing the program on to the board. One can alternatively use the ST Link V2 Programmer. The header for this programmer contains connections for 5V, 3.3V, SWCLK, SWDIO, SWIM, Reset (RST/NRST) and GND. The connector on the opposite side of this device is a USB connector and is intended to be plugged into the computer for programming.

Another method of flashing the program is by using the built-in UART Boot-loader. Further details are explained here. An USART to USB driver chip like FT232 can be used along with STM32 Flash Loader Demonstrator to flash the program.

1.4 Programming the STM32 Micro-controller

Programming the STM32 series micro-controllers is done in various layers namely, CMSIS, HAL, BSP and Application code. The software stack is shown in *Figure 1.2*.

The *Cortex Micro-controller Software Interface Standard (CMSIS)* supports developers in creating reusable software components for ARM Cortex-M based systems. It mostly contains definitions for the various registers.

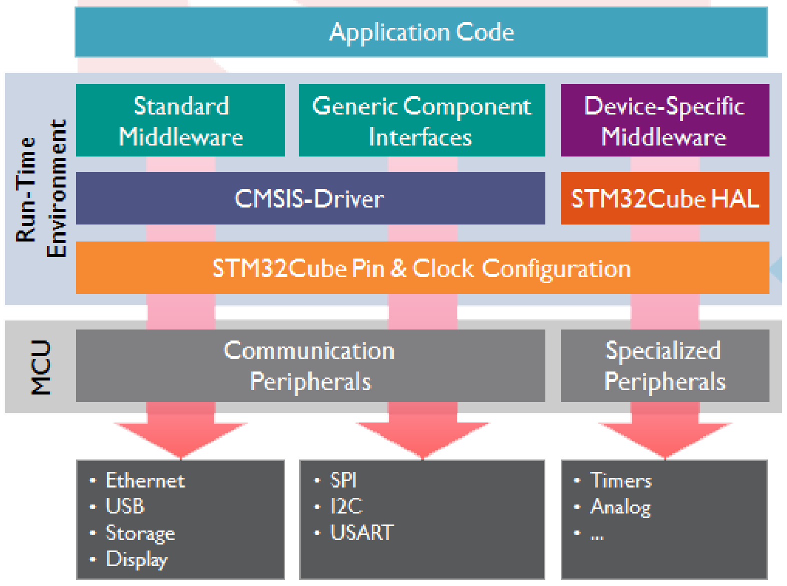


Figure 1.2: Software stack for STM32 programming

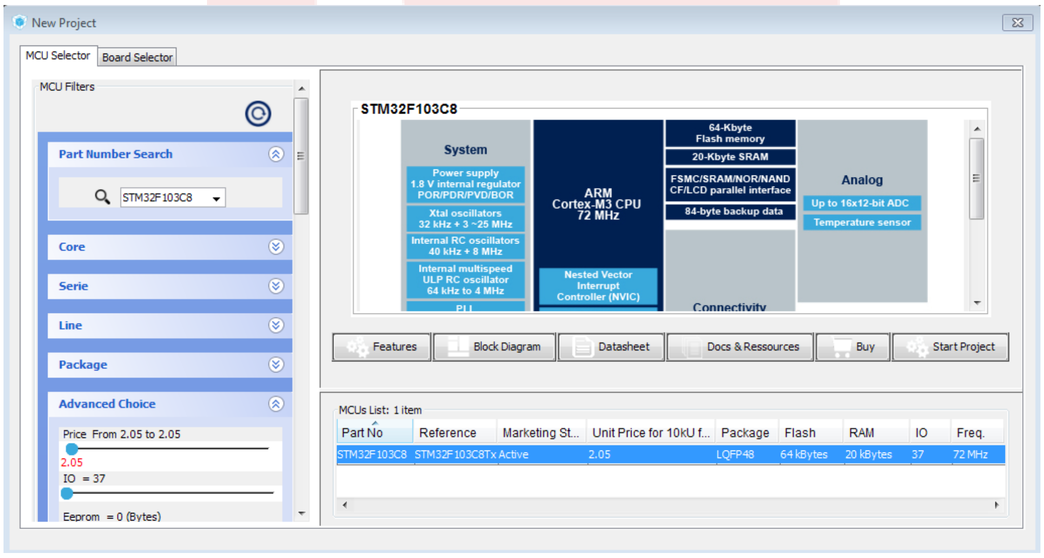
The *Hardware Abstraction Layer (HAL)* drivers were designed to offer a rich set of APIs and to interact easily with the application upper layers. Each driver consists of a set of functions covering the most common peripheral features such as GPIO, UART, I2C, CAN and so on.[4]

1.5 Project Setup using STM32CubeMX

When a STM32 micro-controller starts, it needs an hardware configuration to work correctly. Unfortunately, the development of one's own HAL requires a deep knowledge of the specific micro-controller. ST provides a dedicated tool that generates the initialization code and provides us the HAL peripheral drivers. This tool is called STM32CubeMX and it is very useful.[5] Let us start with a hello world project to blink a LED connected to GPIO pin.

Step 1

Open STM32CubeMX and create a *New Project*. Then select the target micro-controller as *STM32F103C8Tx* as shown in *Figure 1.3*.



STM32F103C8

Price From 2.05 to 2.05
2.05
IO = 37
Eeprom = 0 (Bytes)

MCUs List: 1 item

Part No	Reference	Marketing St...	Unit Price for 10kU f...	Package	Flash	RAM	IO	Freq.
STM32F103C8	STM32F103C8Tx	Active	2.05	LQFP48	64 kBytes	20 kBytes	37	72 MHz

Figure 1.3: Selecting the target micro-controller

Step 2

The micro-controller with its pinout will be displayed as shown in *Figure 1.4*. The on-board LED on Nucleo board is connected to *Port A — Pin 5*. Click on *PA5* and the select *GPIO_Output* to configure it as output pin.

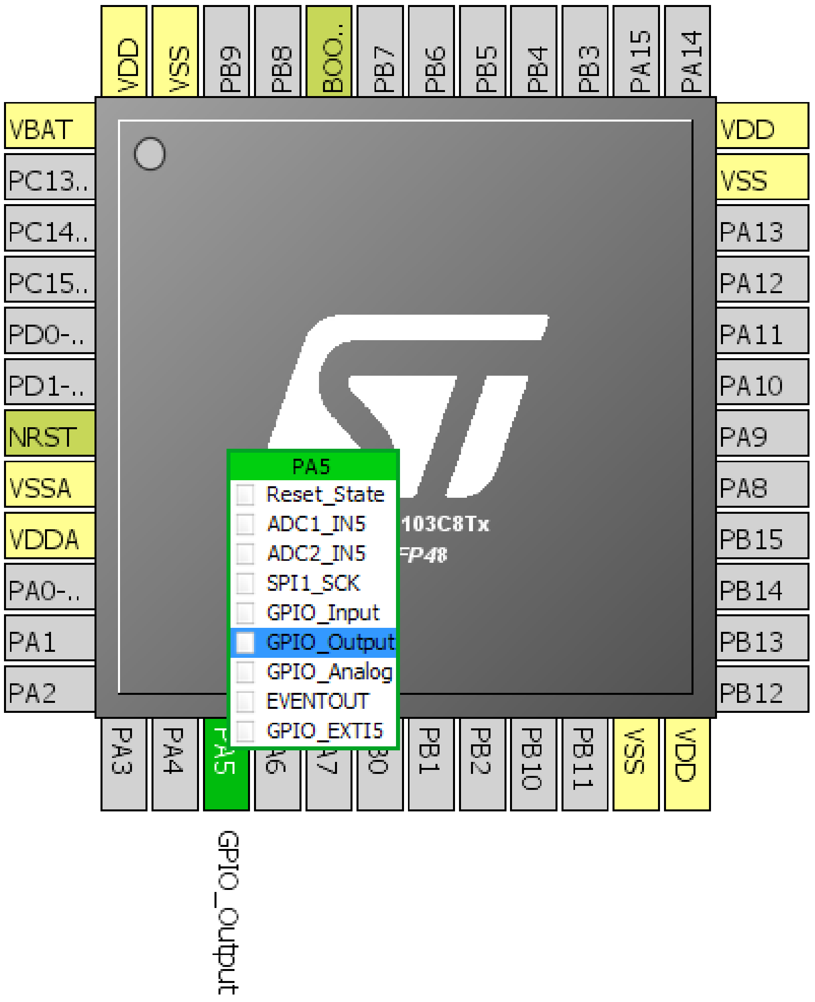


Figure 1.4: Configuring PA5 as GPIO output pin

Step 3

The next step is to configure the clock sources and set the system frequency. Click on the *Clock Configuration* tab. Set HCLK as 64 MHz as shown in *Figure 1.5*.

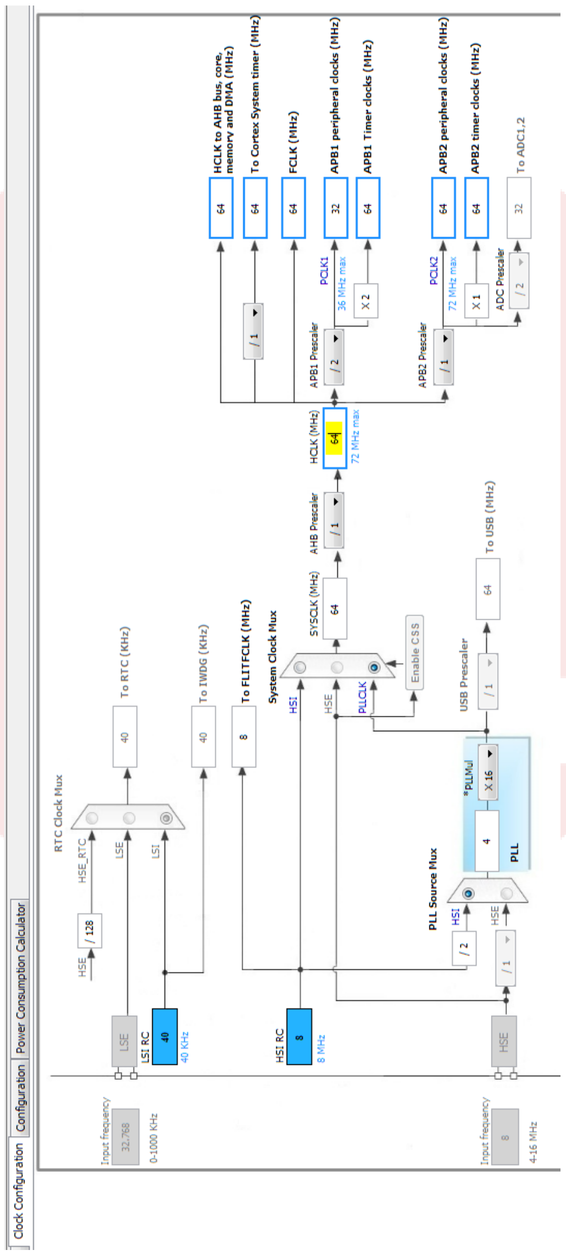


Figure 1.5: Configuring the clock source and system frequency

Step 4

The final step is to generate the code and create a TrueSTUDIO project. Click the *Generate source code* button as highlighted in *Figure 1.7*. The *Project Settings* will open, where you configure the project name, location and the IDE. Finally, click *OK* as shown in **??**.

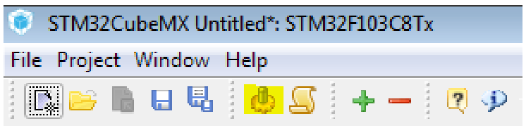
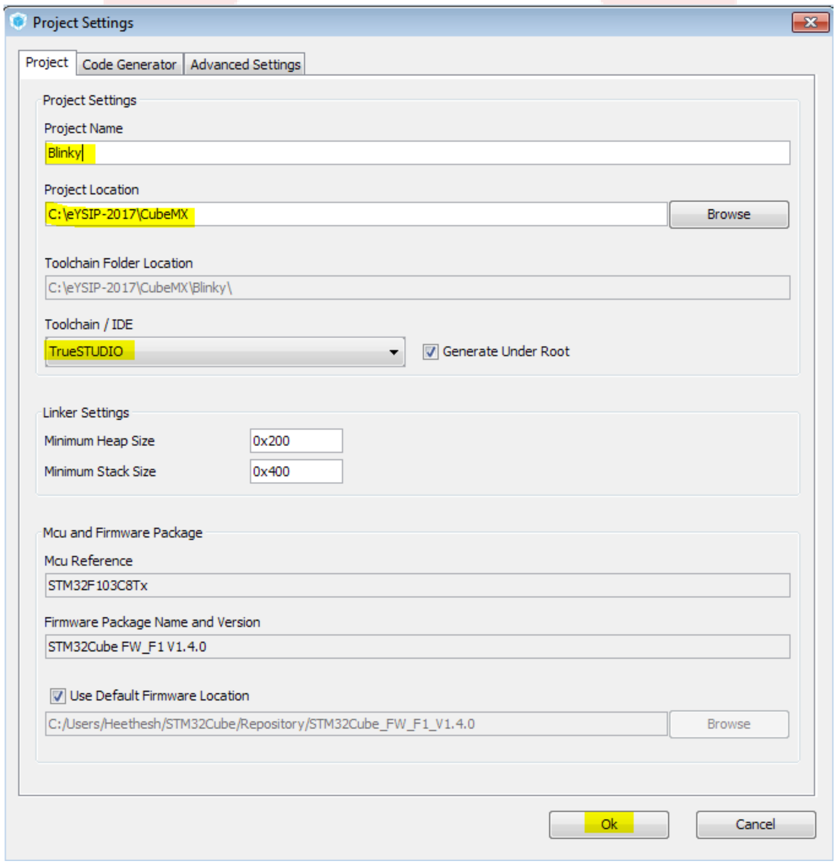


Figure 1.6: Generate source code button



Project Settings

Project Name: Blinky

Project Location: C:\eYSIP-2017\CubeMX

Toolchain Folder Location: C:\eYSIP-2017\CubeMX\Blinky\

Toolchain / IDE: TrueSTUDIO

[✓] Generate Under Root

Linker Settings

Minimum Heap Size: 0x200

Minimum Stack Size: 0x400

Mcu and Firmware Package

Mcu Reference: STM32F103C8Tx

Firmware Package Name and Version: STM32Cube FW_F1 V1.4.0

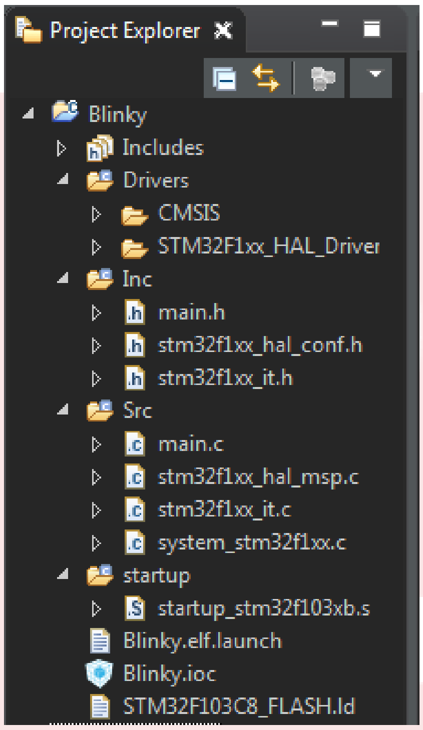
[✓] Use Default Firmware Location

C:/Users/Heethesh/STM32Cube/Repository/STM32Cube_FW_F1_V1.4.0

Figure 1.7: Project settings window

1.6 Programming in TrueSTUDIO

Now that the project has been setup and all the initialization is done we can begin with the programming. Open the TrueSTUDIO project file *.cproject*. In the project explorer window on the left side, a bunch of folder and files created by *STM32CubeMX* will be visible as shown in *Figure 1.8*.



```
Project Explorer
◢ Blinky
    ▷ Includes
    ◢ Drivers
        ▷ CMSIS
        ▷ STM32F1xx_HAL_Driver
    ◢ Inc
        ▷ main.h
        ▷ stm32f1xx_hal_conf.h
        ▷ stm32f1xx_it.h
    ◢ Src
        ▷ main.c
        ▷ stm32f1xx_hal_msp.c
        ▷ stm32f1xx_it.c
        ▷ system_stm32f1xx.c
    ◢ startup
        ▷ startup_stm32f103xb.s
      Blinky.elf.launch
      Blinky.ioc
      STM32F103C8_FLASH.ld
```

Figure 1.8: Project explorer in TrueSTUDIO

Open the *main.c* file and you see the code in a template format. If you want to configure new peripherals in *STM32CubeMX* along with the existing project, it will overwrite the code in the *main.c* file. To prevent that from happening, various user code sections have been provided for the user to write the code and *STM32CubeMX* retains this part of the code when regenerating the files next time.

The main function should like the snippet below. Note that the template comments have been removed here for better readability.

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

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

  /* Initialize all configured peripherals */
  MX_GPIO_Init();

  while (1)
  {
  }
}
```

The HAL library provides us with functions to read, write and toggle GPIO pins. It also provides us a delay function which is based on the *SysTick* counter. The default resolution of the delay function is configured to 1 ms. Following are snippets showing two different methods two blink the LED.

```c
while (1)
{
  // Write logic HIGH to the pin
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, 0);

  // 1s delay
  HAL_Delay(1000);

  // Write logic LOW to the pin
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, 1);

  // 1s delay
  HAL_Delay(1000);
}
```

```
while (1)
{
  // Toggle GPIO pin
  HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);

  // 1s delay
  HAL_Delay(1000);
}
```

The on-board LED on the Nucleo board will now blink every 1 s. If you are using any other development board, connect and LED to Pin A5 on the micro-controller board.

All the project files used are uploaded along with this tutorial for reference.

A note on Interrupt Handlers

The *stm32f1xx_it.c* file in *Src* folder contains the interrupt and exception handlers. The user has to implement these functions as per requirement. The *SysTick* handler is also in the same file.

Further reading

Geoffrey Brown's book, *Discovering the STM32 Microcontrollers* covers the programming on STM32 in detail. Le Tan Phuc's blog covers more advanced tutorials on *STM32CubeMX*.

References

[1] Geoffrey Brown, *Discovering the STM32 Microcontrollers*, 2016.

[2] STMicroelectronics, *Mainstream Performance line, ARM Cortex-M3 MCU*.

[3] ARM Developer, *GNU ARM Embedded Toolchain*.

[4] STMicroelectronics, *STM32F1xx HAL Manual*

[5] Carmine Noviello, *Setting up a GCC/Eclipse toolchain for STM32Nucleo - Part 1*