Documents for work with STM32

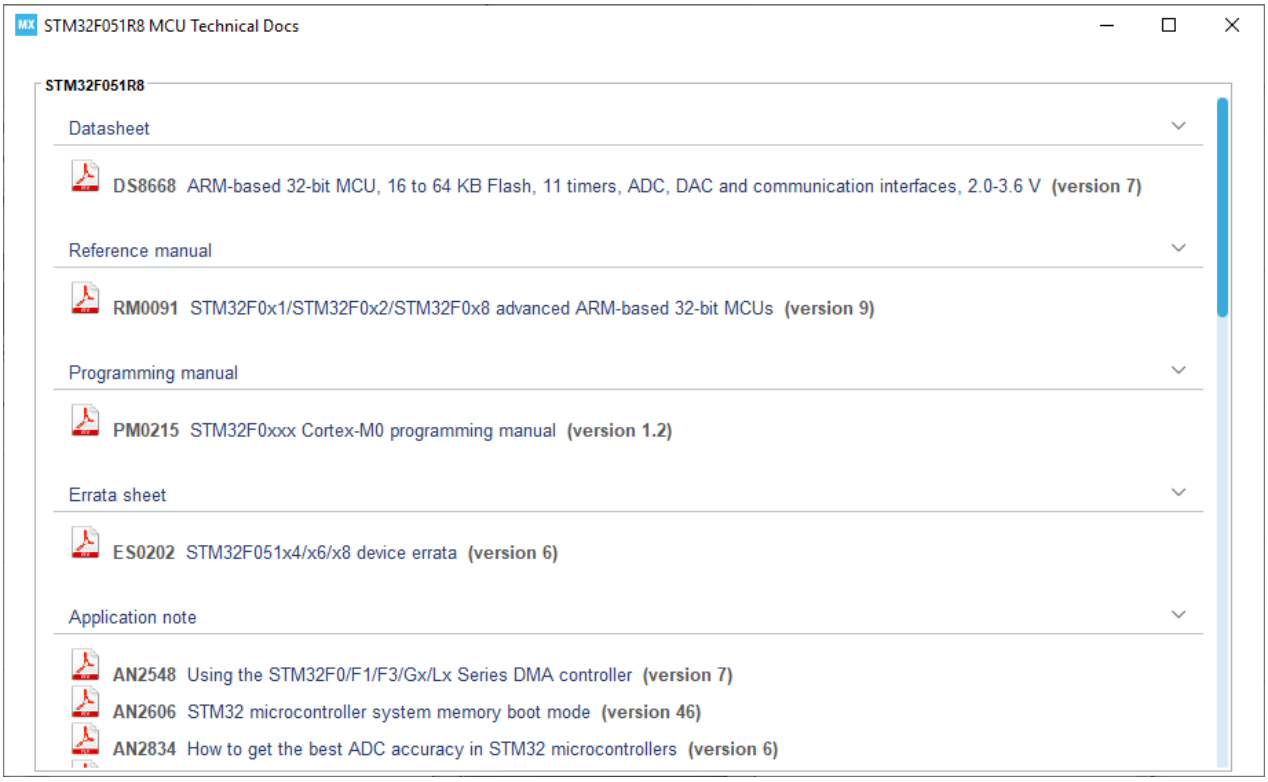
In embedded programming, documents have a very important role as they are the main reference sources for developers to know how the processor works and how to configure it. Those documents mainly come from the processor manufacture, including example Board Schematic, the Datasheet, Programming Manual, and the most important Reference Manual with details of the target microprocessor.

#arm #stm32 #docs

Last update: 2021-06-03 17:15:59

Table of Content

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



List of documents for a target

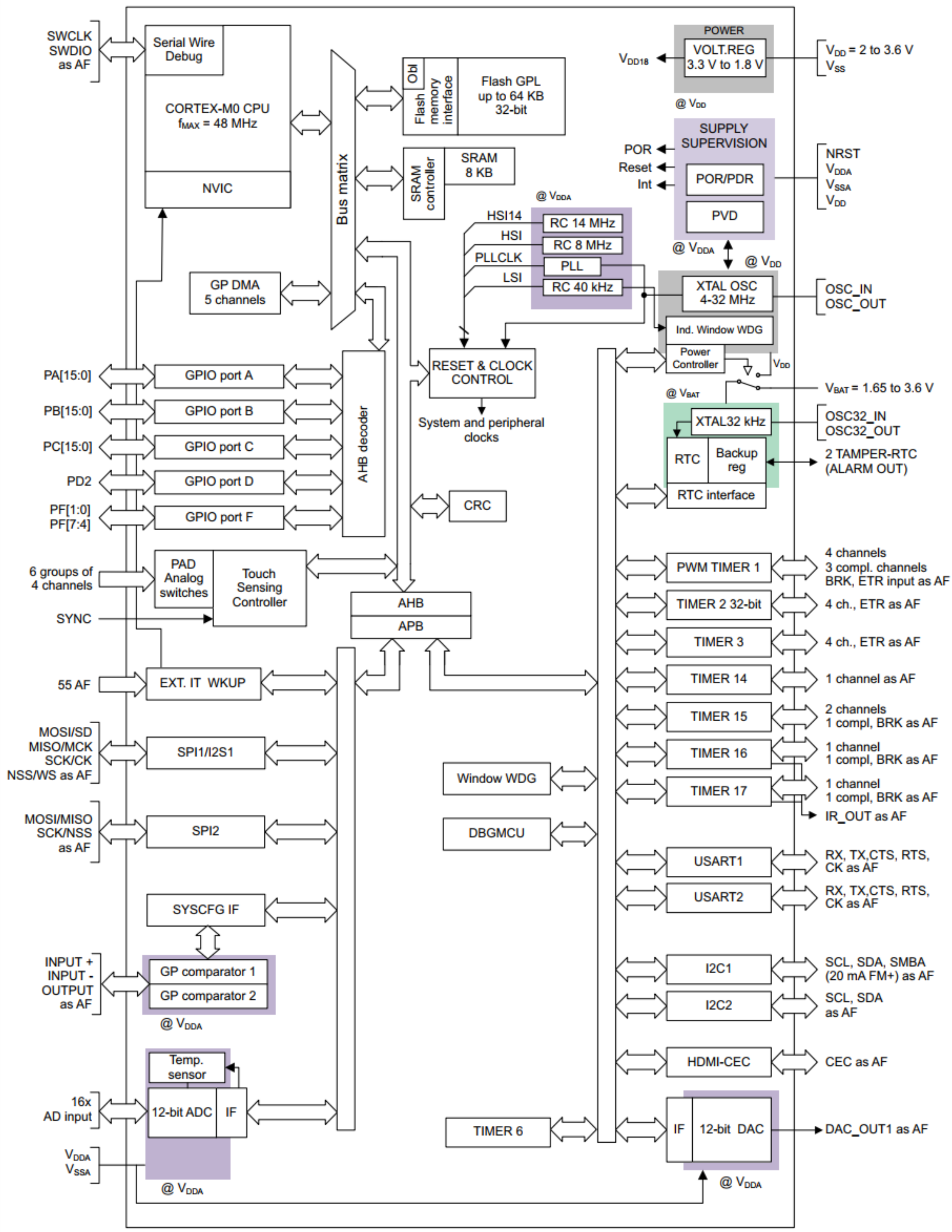
1. The Datasheet

This document contains highlight of the target microprocessor with main features and capabilities. Many people are confused with Reference Manual, but when comparing the content, they are written for different purpose. This document is helpful when designing a PCB. It gives recommended layout for things like signal characteristic, NRST pin, ADC pins, Boot mode, etc.

Datasheet provides the following:

- General description including product line, speed, memory, operating voltage, temperature range
- Device overview with block diagram, available peripherals and functions
- Pinouts and pin descriptions
- Memory map and memory ranges
- Electrical Characteristics
- Package information, for modeling PCB footprints
- Ordering Information

Excerpt from *DS8668 - STM32F051x4 STM32F051x6 STM32F051x8 Datasheet*



The block diagram of STM32F051xx

Boot modes

At startup, the boot pin and boot selector option bit are used to select one of the three boot options:

- boot from User Flash memory
- boot from System Memory
- boot from embedded SRAM

The boot loader is located in System Memory. It is used to reprogram the Flash memory by using USART on pins **PA14**/**PA15** or **PA9**/**PA10**.

Pinout table

I/O structure with marker `FT` for 5V-tolerant I/O, `TT` or `TC` for 3.3V-only I/O.
Unless otherwise specified by a note, all I/Os are set as floating inputs during and after reset.

Pin number						Pin name (function upon reset)	Pin type	I/O structure	Notes	Pin functions	
LQFP64	UFBGA64	LQFP48/UFQFPN48	WLCSP36	LQFP32	UFQFPN32					Alternate functions	Additional functions
7	E1	7	D5	4	4	NRST	I/O	RST	-	Device reset input / internal reset output (active low)	
8	E3	-	-	-	-	PC0	I/O	TTa	-	EVENTOUT	ADC_IN10
9	E2	-	-	-	-	PC1	I/O	TTa	-	EVENTOUT	ADC_IN11
10	F2	-	-	-	-	PC2	I/O	TTa	-	EVENTOUT	ADC_IN12
11	G1	-	-	-	-	PC3	I/O	TTa	-	EVENTOUT	ADC_IN13
12	F1	8	D6	16	0	VSSA	S	-	(3)	Analog ground	
13	H1	9	E5	5	5	VDDA	S	-	-	Analog power supply	

The pinout description

Pin name	AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7
PA0	-	USART2_CTS	TIM2_CH1_ETR	TSC_G1_IO1		-	-	COMP1_OUT
PA1	EVENTOUT	USART2_RTS	TIM2_CH2	TSC_G1_IO2			-	-
PA2	TIM15_CH1	USART2_TX	TIM2_CH3	TSC_G1_IO3	-	-	-	COMP2_OUT
PA3	TIM15_CH2	USART2_RX	TIM2_CH4	TSC_G1_IO4	-	-	-	-
PA4	SPI1_NSS, I2S1_WS	USART2_CK	-	TSC_G2_IO1	TIM14_CH1	-	-	-

Alternate functions

Bus	Boundary address	Size	Peripheral
	0x4800 1800 - 0x5FFF FFFF	~384 MB	Reserved
AHB2	0x4800 1400 - 0x4800 17FF	1 KB	GPIOF
	0x4800 1000 - 0x4800 13FF	1 KB	Reserved
	0x4800 0C00 - 0x4800 0FFF	1 KB	GPIOD
	0x4800 0800 - 0x4800 0BFF	1 KB	GPIOC
	0x4800 0400 - 0x4800 07FF	1 KB	GPIOB
	0x4800 0000 - 0x4800 03FF	1 KB	GPIOA

Memory map and boundary address

Symbol	Parameter	Conditions	Min	Max	Unit
f_{HCLK}	Internal AHB clock frequency	-	0	48	MHz
f_{PCLK}	Internal APB clock frequency	-	0	48	
V_{DD}	Standard operating voltage	-	2.0	3.6	V
V_{DDA}	Analog operating voltage (ADC and DAC not used)	Must have a potential equal to or higher than V_{DD}	V_{DD}	3.6	V
	Analog operating voltage (ADC and DAC used)		2.4	3.6	
V_{BAT}	Backup operating voltage	-	1.65	3.6	V
V_{IN}	I/O input voltage	TC and RST I/O	–0.3	V_{DDIOx}+0.3	V
		TTa I/O	–0.3	V_{DDA}+0.3[1]	
		FT and FTf I/O	–0.3	5.5[1]	
		BOOT0	0	5.5	

Operation condition

OSPEEDRy [1:0] value[1]	Symbol	Parameter	Conditions	Min	Max	Unit
x0	$f_{max(IO)out}$	Maximum frequency[3]	C_L = 50 pF	-	2	MHz
	$t_{f(IO)out}$	Output fall time		-	125	ns
	$t_{r(IO)out}$	Output rise time		-	125	
01	$f_{max(IO)out}$	Maximum frequency[3]	C_L = 50 pF	-	10	MHz
	$t_{f(IO)out}$	Output fall time		-	25	ns
	$t_{r(IO)out}$	Output rise time		-	25	

Speed modes on IO

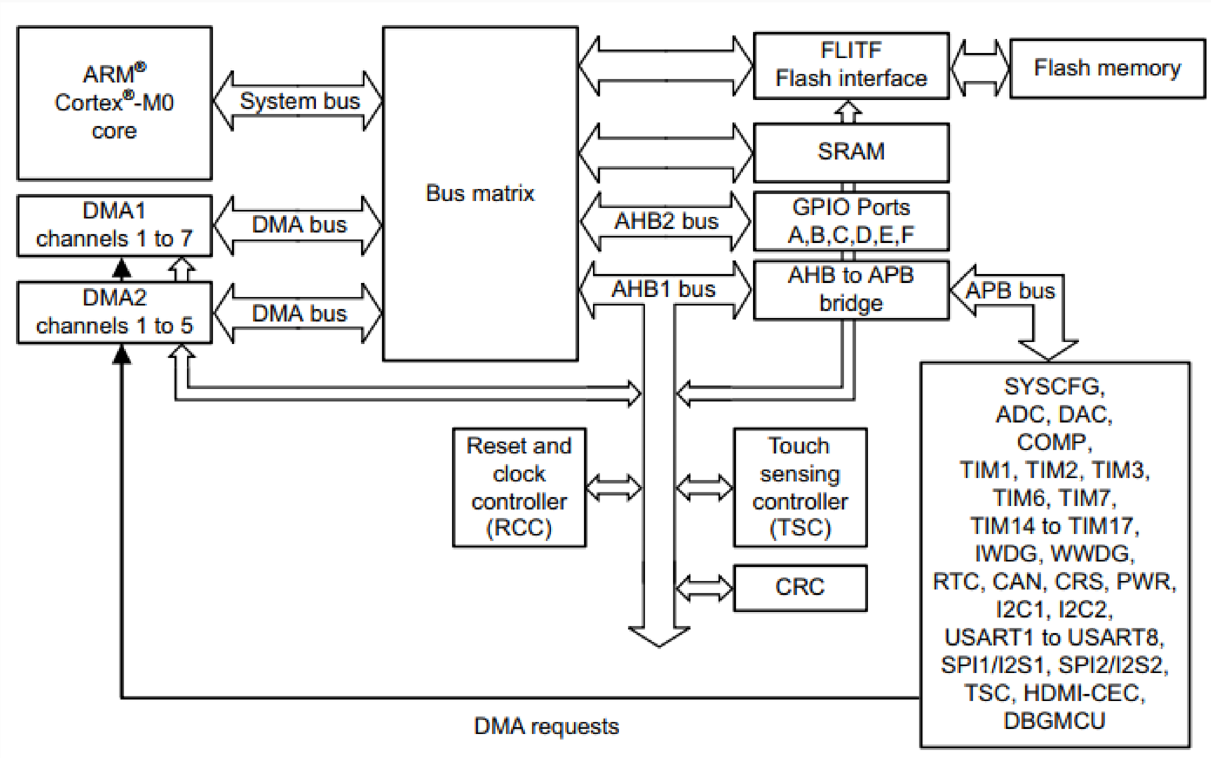
2. The Reference Manual

This is by far the most important document in order to program the target device. It defines all information about the core and peripheral at register level with bit-by-bit description. By using only this document, developer still can program the chip without any higher level API - usually called Bare-metal programming.

Reference Manual provides the following:

- System Architecture with bus, peripherals, and connections
- Memory map and boundary address
- Boot configuration and vector table relocation
- Peripheral with details features, descriptions, and structure
- **Register name and bit-fields for all accessible registers**
- Code examples using CMSIS header files

> **Excerpt from** *RM0091 - STM32F0x1/STM32F0x2/STM32F0x8 advanced ARM®-based 32-bit MCUs*



System architecture for STM32F0x

Boot modes

The boot mode configuration is latched on the 4th rising edge of SYSCLK after a reset, and is also re-sampled when exiting from Standby mode. After this startup delay has elapsed, the CPU always fetches the top-of-stack value from address `0x00000000`, then starts code execution from the boot memory at `0x00000004`.

Depending on the selected boot mode, main Flash memory, system memory or SRAM is accessible as follows:

- Boot from main Flash memory: the main Flash memory is aliased in the boot memory space `0x00000000`, but still accessible from its original memory space `0x08000000`.
- Boot from system memory: the system memory is aliased in the boot memory space `0x00000000`, but still accessible from its original memory space (`0x1FFFEC00` on STM32F03x and STM32F05x devices, `0x1FFFC400` on STM32F04x devices, `0x1FFFC800` on STM32F07x and `0x1FFFD800` on STM32F09x devices).
- Boot from the embedded SRAM: the SRAM is aliased in the boot memory space `0x00000000`, but it is still accessible from its original memory space `0x20000000`.

Physical remap

For application code which is located in a different address than `0x08000000`, some additional code must be added in order to be able to serve the application interrupts. A solution will be to relocate by software the vector table to the internal SRAM, at the initialization phase:

- Copy the vector table from the Flash (mapped at the base of the application load address) to the base address of the SRAM at `0x20000000`
- Remap SRAM at address `0x00000000`, using `SYSCFG` configuration register 1

Embedded boot loader

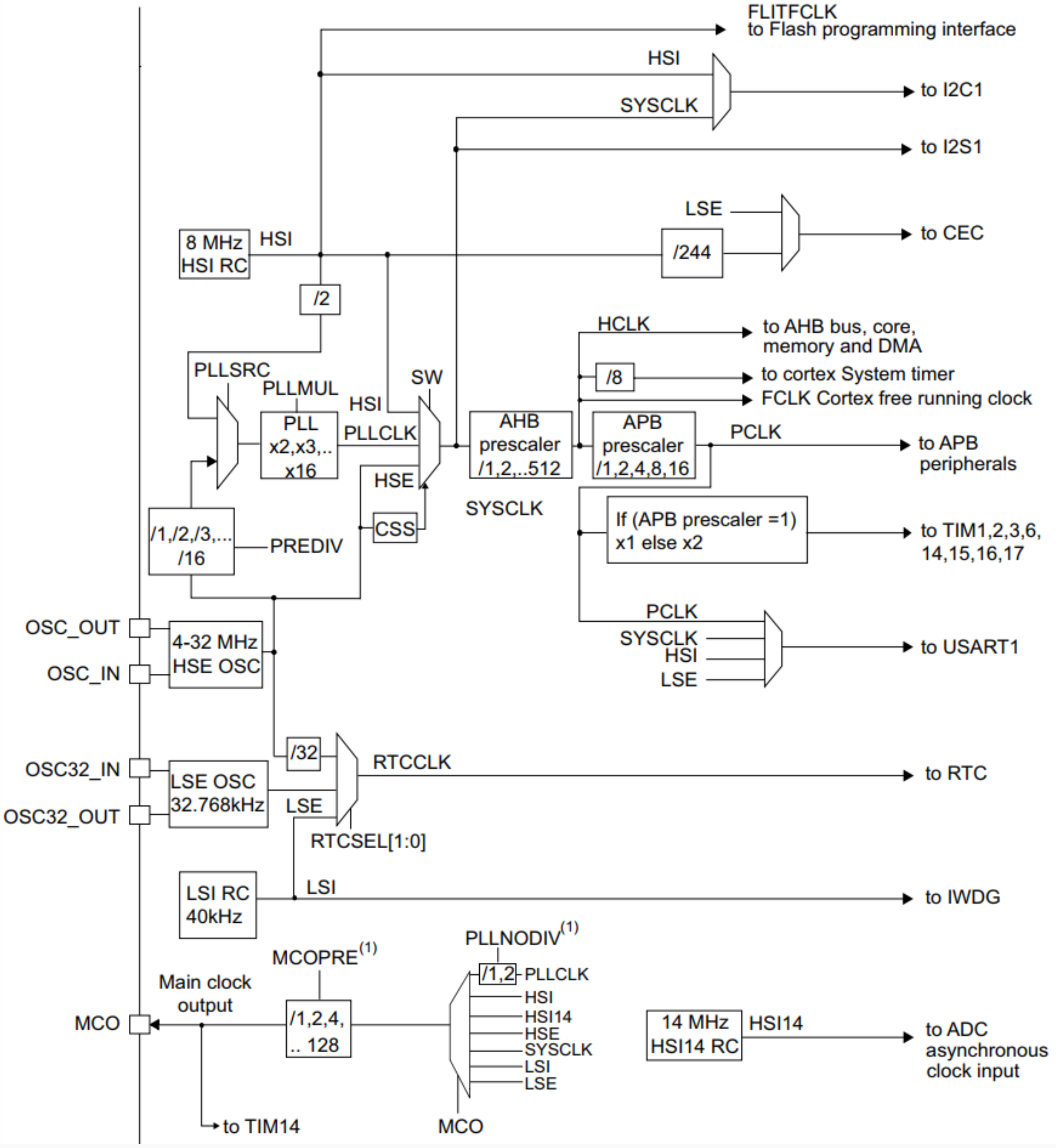
The embedded boot loader is located in the System memory, programmed by ST during production. It is used to reprogram the Flash memory using one of the following serial interfaces:

- USART on pins **PA14**/**PA15** or **PA9**/**PA10**
- I2C on pins **PB6**/**PB7** (STM32F04xxx, STM32F07xxx and STM32F09xxx devices only)
- USB DFU interface (STM32F04xxx and STM32F07xxx devices only)

Debug pin

During and just after reset, the alternate functions are not active and most of the I/O ports are configured in input floating mode, except the debug pins are in AF mode immediately:

- **PA14**: SWCLK in pull-down
- **PA13**: SWDIO in pull-up



The clock paths

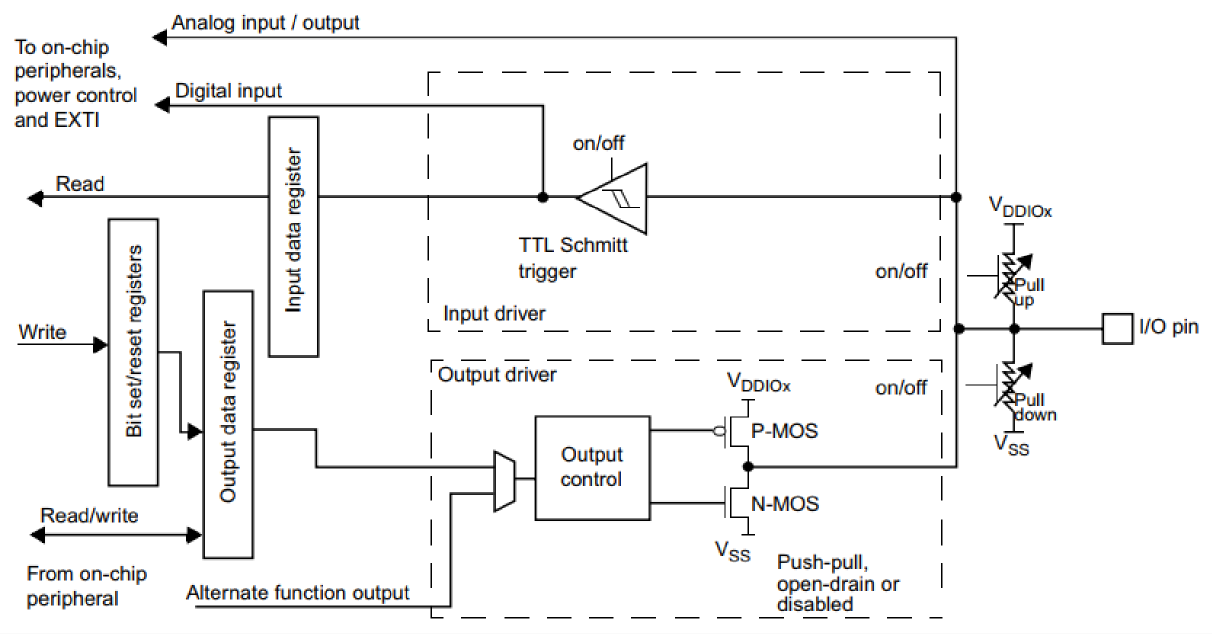
Example code

USART transmitter configuration:

```
/* (1) Oversampling by 16, 9600 baud */
/* (2) 8 data bit, 1 start bit, 1 stop bit, no parity */
USART1->BRR = 480000 / 96; /* (1) */
USART1->CR1 = USART_CR1_TE | USART_CR1_UE; /* (2) */
```

USART transfer:

```
if ((USART1->ISR & USART_ISR_TC) == USART_ISR_TC) {
    if (send == sizeof(stringtosend)) {
        send=0;
        USART1->ICR |= USART_ICR_TCCF; /* Clear transfer complete flag */
    } else {
        /* clear transfer complete flag and fill TDR with a new char */
        USART1->TDR = stringtosend[send++];
    }
}
```



The structure of an IO pin

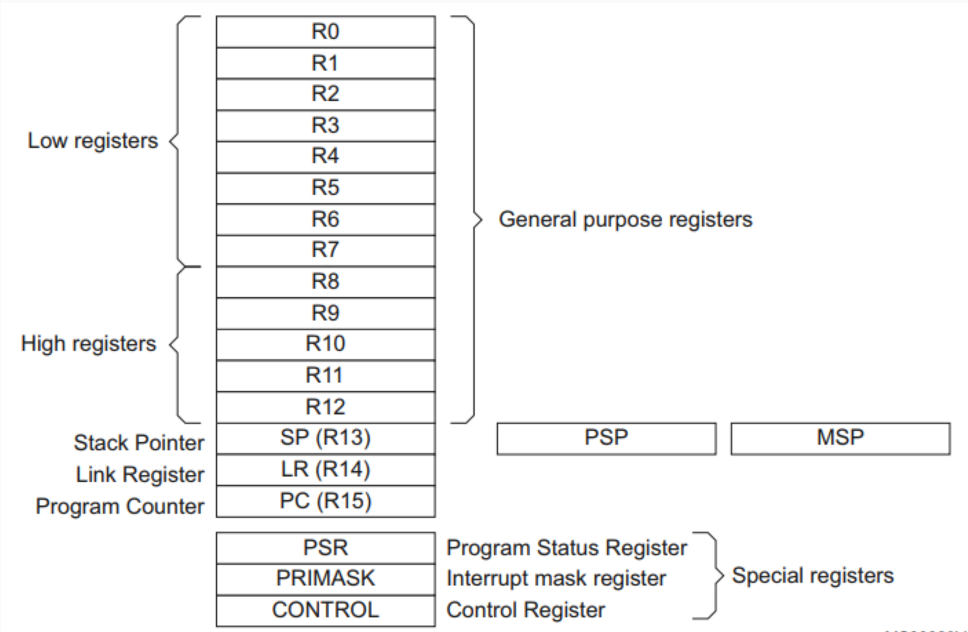
3. Programming Manual

This programming manual provides information for application and system-level software developers. It gives a full description of the STM32 Cortex™-M0 processor programming model, instruction set and core peripherals.

Programming Manual provides the following:

- Processor Modes, Stacks
- Memory model
- Exception model, the Vector table and the interrupt service routines
- Fault handling
- Power management: enter Sleep mode, Wake up
- The Instruction Set

- CMSIS intrinsic functions
- Core Peripherals:
  - Memory Protection Unit (MPU)
  - Nested vectored interrupt controller (NVIC)
  - System control block (SCB)
  - SysTick timer (STK)

> **Excerpt from** ***PM0215 - STM32F0xxx Cortex-M0 programming manual***
>
> **Processor modes**
>
> Thread mode: Used to execute application software.
>
> The processor enters Thread mode when it comes out of reset
>
> Handler mode: Used to handle exceptions.
>
> The processor returns to Thread mode when it has finished exception processing.
>
> 
>
>
> *Registers*
>
> **Stacks**
>
> The processor uses a full descending stack. This means the stack pointer indicates the last stacked item on the stack memory.
>
> The processor implements two stacks, with independent copies of the stack pointer:
>
> - The main stack and
> - The process stack

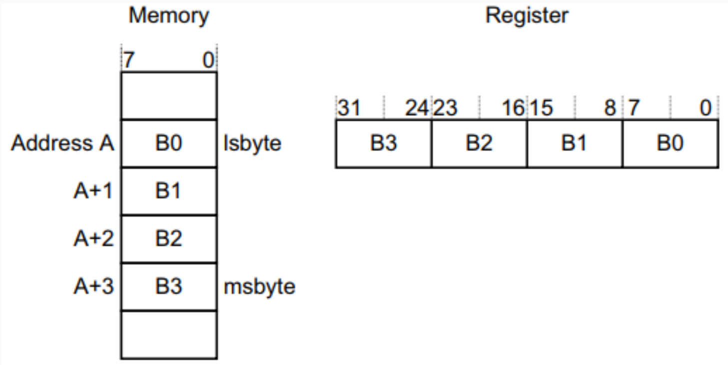
In Thread mode, the CONTROL register controls whether the processor uses the main stack or the process stack:

- `0` : Main Stack Pointer (MSP)(reset value). On reset, the processor loads the MSP with the value from address `0x00000000` .
- `1` : Process Stack Pointer (PSP).

In Handler mode, the processor always uses the main stack.

Memory endianness

The processor views memory in little-endian format. It stores the least significant byte (lsbyte) of a word at the lowest-numbered byte, and the most significant byte (msbyte) at the highest-numbered byte.



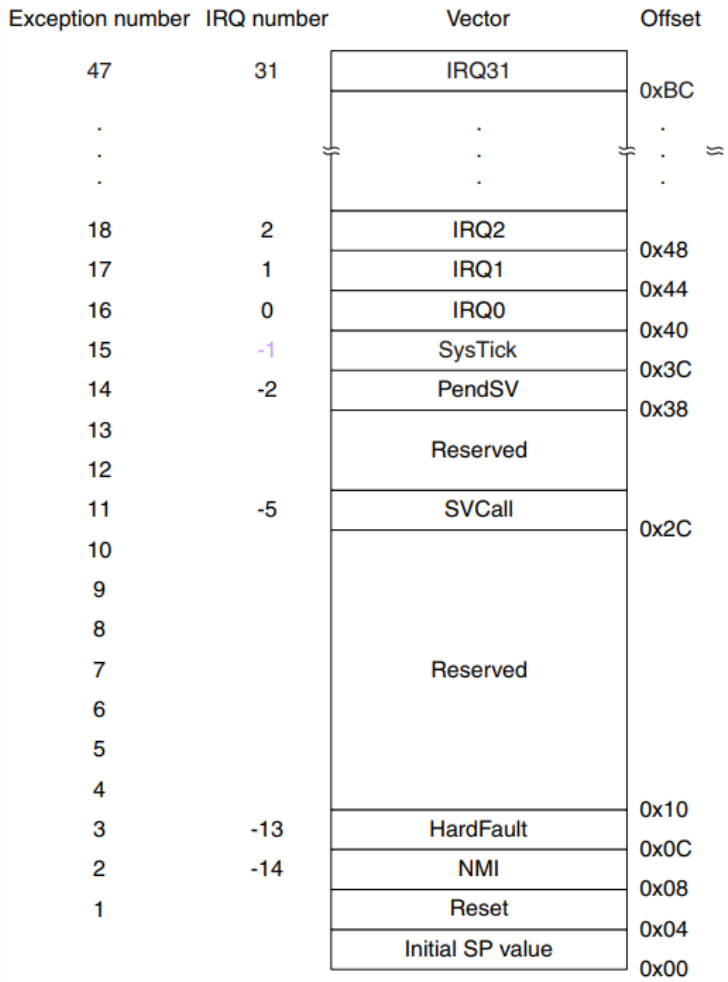
The Little-Endian memory layout

Exception number(1)	IRQ number(1)	Exception type	Priority	Vector address or offset(2)	Activation
1	-	Reset	-3, the highest	0x00000004	Asynchronous
2	-14	NMI	-2	0x00000008	Asynchronous
3	-13	Hard fault	-1	0x0000000C	Synchronous
4-10	-	Reserved	-	-	-
11	-5	SVCall	Configurable(3)	0x0000002C	Synchronous
12-13	-	Reserved	-	-	-
14	-2	PendSV	Configurable(3)	0x00000038	Asynchronous
15	-1	SysTick	Configurable(3)	0x0000003C	Asynchronous
16 - 47	0 - 31	Interrupt (IRQ)	Configurable(3)	0x00000040 and above (4)	Asynchronous

The exception types

Vector table

On system reset, the vector table is fixed at address `0x00000000` . The least-significant bit of each vector must be 1, indicating that the exception handler is Thumb code.



The exception vector table

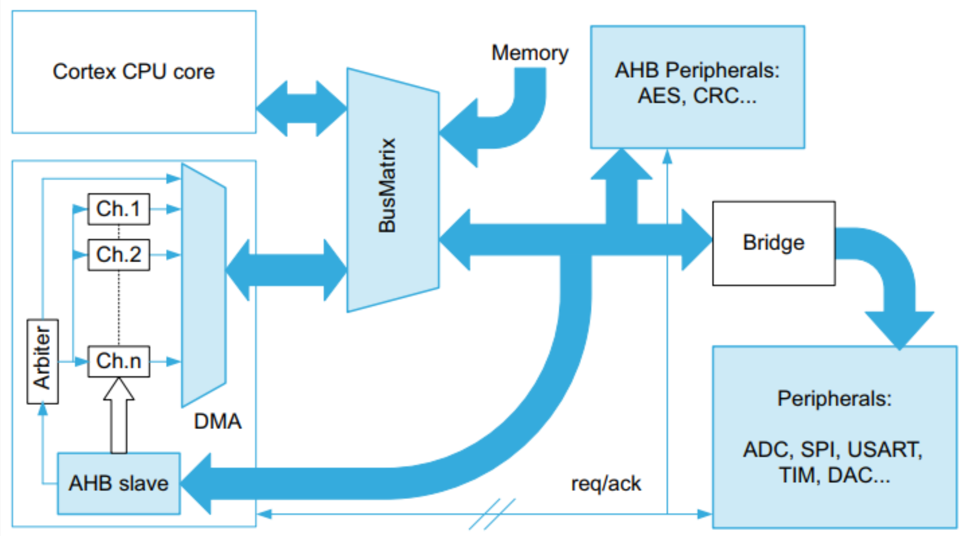
4. Application Note

There many Application Note documents provided by ST. Each document present the usage, design, and advice for a specific application or feature.

Application Note provides the following:

- Peripherals architecture in hardware and software
- Operation characteristic

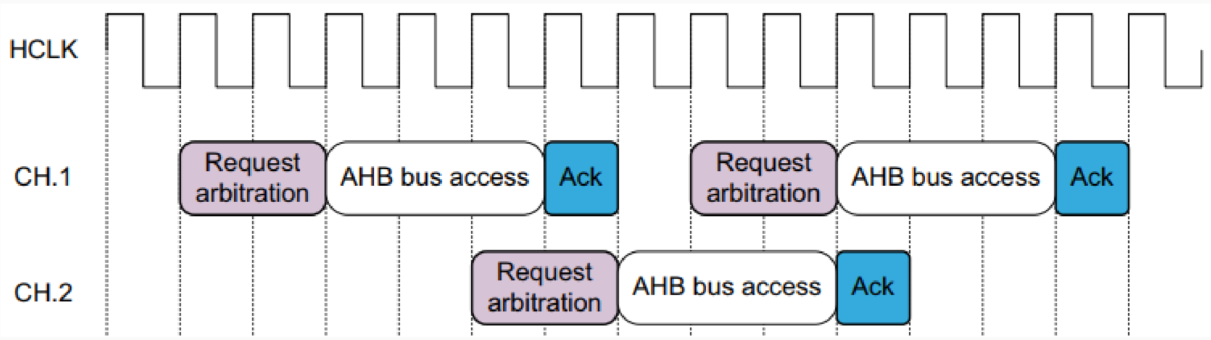
> **Excerpt from** ***AN2548 - Using the STM32F0/F1/F3/Gx/Lx Series DMA controller***
>
> **DMA transfer timing**
>
> For the case where only one DMA channel is active, a new hardware back-to-back request can not be handled by the DMA before the completion of the previous one, adding one AHB clock cycle for the final idle phase of the DMA request-acknowledge handshake protocol.



DMA Block diagram

When more than one channel is requesting a DMA transfer, the DMA request arbitration can be performed meanwhile the two last cycles of when the AHB bus is accessed by the DMA. Request arbitration overhead is then masked by the AHB bus transfer time.

In case not only two channels, but two DMA controllers are used (in products that offer this possibility), two DMA transfers can be processed in parallel, as long as they are not conflicting within the bus matrix, not accessing the same slave device.

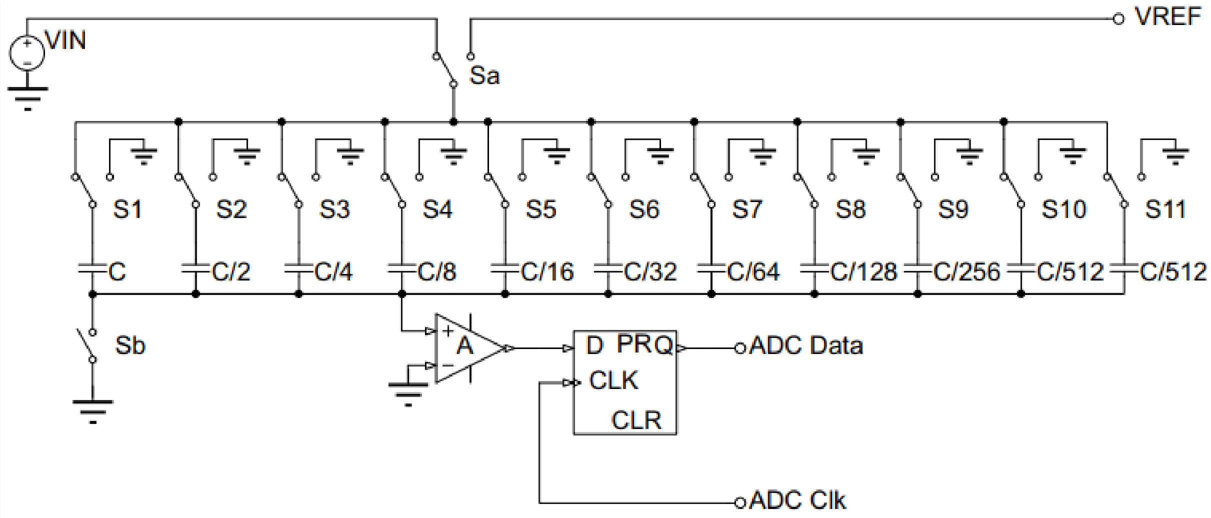


Timing of Two DMA channel on AHB Bus

Excerpt from *AN2834 - How to get the best ADC accuracy in STM32 microcontrollers*

SAR ADC internal structure

The ADC embedded in STM32 microcontrollers uses the SAR (successive approximation register) principle, by which the conversion is performed in several steps. The number of conversion steps is equal to the number of bits in the ADC converter. Each step is driven by the ADC clock. Each ADC clock produces one bit from result to output. The ADC internal design is based on the switched-capacitor technique.



Basic schematic of SAR switched-capacitor ADC

How to get the best ADC accuracy

- Reduce the effects of ADC-related ADC errors
  Offset and gain errors can be easily compensated using the STM32 ADC self-calibration feature or by microcontroller firmware.
- Minimize ADC errors related to external environment of ADC
  It is recommended to connect capacitors with good high-frequency characteristics between the power and ground lines. That is, a 0.1 μF and a 1 to 10 μF capacitor should be placed close to the power source.
  In most STM32 microcontrollers, the VDD and VSS pins are placed close to each other. So are the VREF+ and VSSA pins. A capacitor can therefore be connected very close to the microcontroller with very short leads. For multiple VDD and VSS pins, use separate decoupling capacitors.

5. Platform API Manual

When using a software platform as a base for application development, the API manual document provides the usage and use case of available functions, settings, and parameters.

STM32 MCUs come with Hardware Abstract Layer (HAL) and Low-Layer (LL) library which are used in code generation from CubeMX.

> **Excerpt from** *UM1785 - Description of STM32F0 HAL and low-layer drivers*
>
> **GPIO Firmware driver API description**
>
> 1. Enable the GPIO AHB clock using the following function `__HAL_RCC_GPIOx_CLK_ENABLE()`.
> 2. Configure the GPIO pin(s) using `HAL_GPIO_Init()`.

- Configure the IO mode using "Mode" member from `GPIO_InitTypeDef` structure
- Activate Pull-up, Pull-down resistor using "Pull" member from `GPIO_InitTypeDef` structure.
- In case of Output or alternate function mode selection: the speed is configured through "Speed" member from `GPIO_InitTypeDef` structure.
- In alternate mode is selection, the alternate function connected to the IO is configured through "Alternate" member from `GPIO_InitTypeDef` structure.
- Analog mode is required when a pin is to be used as ADC channel or DAC output.
- In case of external interrupt/event selection the "Mode" member from `GPIO_InitTypeDef` structure select the type (interrupt or event) and the corresponding trigger event (rising or falling or both).

3. In case of external interrupt/event mode selection, configure NVIC IRQ priority mapped to the `EXTI` line using `HAL_NVIC_SetPriority()` and enable it using `HAL_NVIC_EnableIRQ()`.
4. `HAL_GPIO_DeInit()` allows to set register values to their reset value. It's also recommended to use it to un-configure pin which was used as an external interrupt or in event mode. That's the only way to reset corresponding bit in `EXTI` & `SYSCFG` registers.
5. To get the level of a pin configured in input mode use `HAL_GPIO_ReadPin()`.
6. To set/reset the level of a pin configured in output mode use `HAL_GPIO_WritePin()` / `HAL_GPIO_TogglePin()`.
7. To lock pin configuration until next reset use `HAL_GPIO_LockPin()`.
8. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG/SWD pins).
9. The LSE oscillator pins **OSC32_IN** and **OSC32_OUT** can be used as general purpose (**PC14** and **PC15**, respectively) when the LSE oscillator is off. The LSE has priority over the GPIO function.
10. The HSE oscillator pins **OSC_IN** and **OSC_OUT** can be used as general purpose **PF0** and **PF1**, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

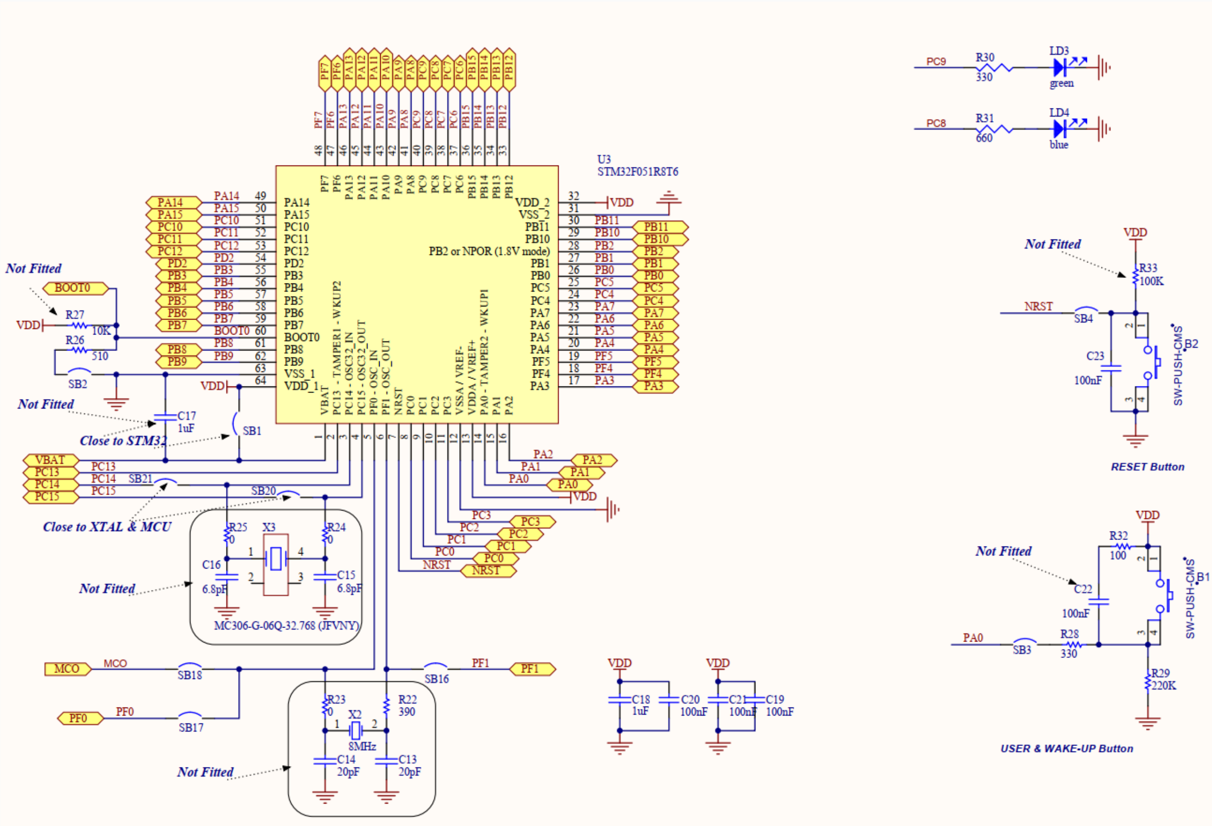
6. Mainboard schematic

It is better to get a schematic of the board which is under the development, to know the correct signal level and characteristic.

Mainboard schematic provides the following:

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

> When downloading schematic from ST, please check the version of hardware on the board, such as *MB1034B*. In old Manual Document, there is a section for schematic.

> **Excerpt from** ***MB1034 - STM32F0DISCOVERY schematic***
>
> 
>
>
> *STMF0DISCOVERY schematic*

7. Board-specific document

When using an official board from ST, there are some board-specific documents provided to users:

- **Peripheral firmware example**
- **Migration and compatibility guidelines**

8. Application integration

When using RTOS or other application later, it is recommended to read their guides and API documents.

For example:

UM1722 - Developing applications on STM32Cube with RTOS

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

- FreeRTOS: overview, APIs, memory management, low power managements, and configuration
- CMSIS-RTOS: a higher layer to communicate between CMSIS and FreeRTOS
- Usage to create thread, use Semaphore, Queues, and Timer

CMSIS - Cortex Microcontroller Software Interface Standard

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

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

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

9. Source Code

Reading a source code and understanding how it works is one of a good way to know about the target system. There are comments in the source code too, and they usually explain about a corner case, issue, or the particular purpose of the implementation.

10. Website

Yep, search on the internet, read them all, sometime ask people, and try to answer other’s question. All those actions can help in learning not only programming but also other fields.