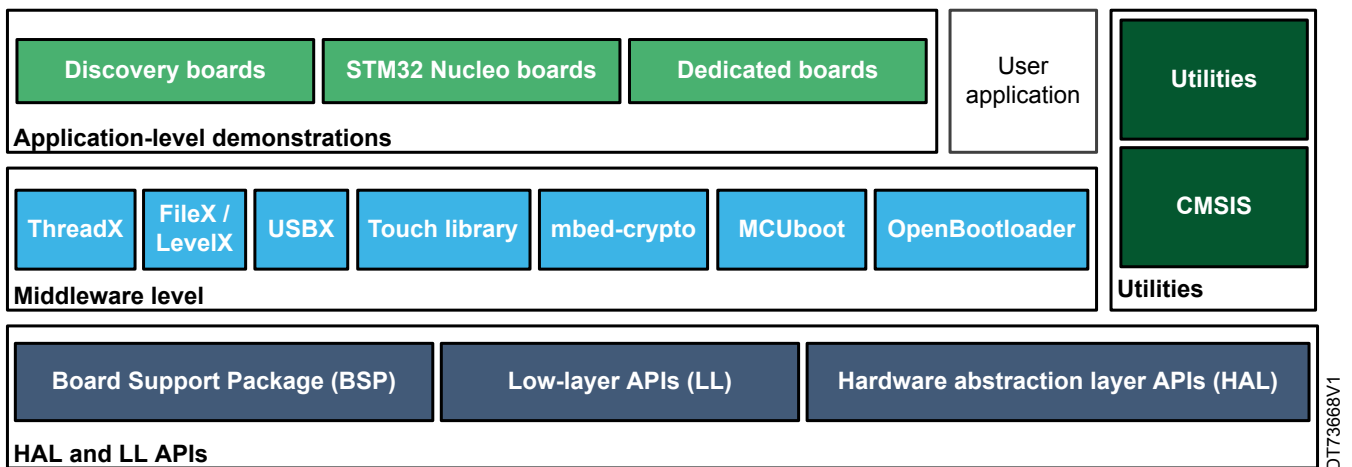


## Introduction to STM32Cube MCU Package examples for STM32U0 MCUs

### Introduction

The STM32CubeU0 MCU Package is delivered with a rich set of examples running on STMicroelectronics boards. The examples are organized by boards. They are provided with preconfigured projects for the main supported toolchains (refer to Figure 1. STM32CubeU0 firmware components).

**Figure 1. STM32CubeU0 firmware components**



## 1 Reference documents

---

The following items make up a reference set for the examples presented in this application note:

- The latest release of the [STM32CubeU0](#) MCU Package for the 32-bit microcontrollers in the STM32U0 series based on the Arm® Cortex®-M0+ processor.
- [Getting started with STM32CubeU0 for STM32U0 series](#) ()
- [Description of STM32U0 HAL and low-layer drivers](#) ()

*Note:* Arm and TrustZone are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.



## 2 STM32CubeU0 examples

The examples are classified depending on the STM32Cube level that they apply to. They are named as follows:

- **Examples**  
 These examples use only the HAL and BSP drivers (middleware not used). Their objective is to demonstrate the product or peripheral features and usage. They are organized per peripheral (one folder per peripheral, such as TIM). Their complexity level ranges from the basic usage of a given peripheral, such as PWM generation using a timer, to the integration of several peripherals, such as how to use DAC for a signal generation with synchronization from TIM6 and DMA. The usage of the board resources is reduced to the strict minimum.
- **Examples\_LL**  
 These examples use only the LL drivers (HAL drivers and middleware components not used). They offer an optimum implementation of typical use cases of the peripheral features and configuration sequences. The examples are organized per peripheral (one folder for each peripheral, such as TIM) and are principally deployed on Nucleo boards.
- **Examples\_MIX**  
 These examples use only HAL, BSP, and LL drivers (Middleware components are not used). They aim at demonstrating how to use both HAL and LL APIs in the same application in order to combine the advantages of both APIs:
  - HAL offers high-level function-oriented APIs with high portability level by hiding product/IPs complexity for end-users.
  - LL provides low-level APIs at the register level with better optimization.
 The examples are organized per peripheral (one folder for each peripheral, such as TIM) and are exclusively deployed on Nucleo boards.
- **Applications**  
 The applications demonstrate product performance and how to use the available middleware stacks. They are organized either by middleware (one folder per middleware, such as Azure® RTOS ThreadX) or product feature that requires high-level firmware bricks (such as LPBAM). The integration of applications that use several middleware stacks is also supported.
- **Demonstrations**  
 The demonstrations aim at integrating and running the maximum number of peripherals and middleware stacks to showcase the product features and performance.
- **Template project**  
 The template project is provided to enable the user to quickly build a firmware application using HAL and BSP drivers on a given board.
- **Template\_LL project**  
 The template LL projects are provided to enable the user to quickly build a firmware application using LL drivers on a given board.

The examples are located under `STM32Cube_FW_U0_VX.Y.Z\Projects\`.

The examples in the default product configuration with the Arm® TrustZone® disabled have the same structure:

- `*\Inc` folder, containing all header files
- `*\Src` folder, containing the sources code
- `*\EWARM`, `*\MDK-ARM`, and `*\STM32CubeIDE` folders, containing the preconfigured project for each toolchain
- `*\README.md` and `*\readme.html` file, describing the example behavior and the environment required to run the example

The examples with the Arm® TrustZone® enabled are suffixed with "\_TrustZone" (except TFM applications) and have the same structure:

- \*\`Secure`\Inc folder, containing all secure project header files
- \*\`Secure`\Src and \*\`Secure_nsclib`\ folders, containing all secure project sources code
- \*\`NonSecure`\Inc folder, containing all nonsecure project header files
- \*\`NonSecure`\Src folder, containing all nonsecure project sources code
- \*\`EWARM`, \*\`MDK-ARM`, and \*\`STM32CubeIDE` folders, containing the preconfigured project for each toolchain
- \*\`README.md` and \*\`readme.html` files, describing the example behavior and the environment required to run the example

To run the example, proceed as follows:

1. Open the example using your preferred toolchain.
2. Rebuild all files and load the image into target memory.
3. Run the example by following the \*\`README.md` and \*\`readme.html` instructions.

**Note:** Refer to "Development toolchains and compilers" and "Supported devices and evaluation boards" sections of the firmware package release notes to know more about the software/hardware environment used for the MCU Package development and validation. The correct operation of the provided examples is not guaranteed in other environments, for example, when using different compilers or board versions.









The examples can be tailored to run on any compatible hardware: simply update the BSP drivers for your board, provided it has the same hardware functions (LED, LCD, pushbuttons, and others). The BSP is based on a modular architecture that can be easily ported to any hardware by implementing low-level routines.

Table 1 contains the list of examples provided with the STM32CubeU0 MCU Package.

In this table, the label **MX** means that the projects are created using STM32CubeMX, the STM32Cube initialization code generator. Those projects can be opened with this tool to modify the projects themselves. The other projects are manually created to demonstrate the product features. In this table, the label TrustZone® means that the projects are created for devices with Arm® TrustZone® enabled. Read the project \*\`README.md` and \*\`readme.html` files for user option bytes configuration.

Table 1. STM32CubeU0 firmware examples

STM32CubeMX-generated examples are highlighted with the icon .

Level	Module name	Project name	Description	STM32U0 83C-DK	NUCLEO- U083RC	NUCLEO- U031R8
Templates_ROT	-	-	This project provides a OEMiROT boot path application template. The boot is performed through the OEMiROT boot path after the authenticity and integrity checks of the project firmware and project data image.	-	X	-
	Total number of templates_rot: 1			0	1	0
Templates_Board	-	-	This project provides a reference template for the NUCLEO-U083RC board, based on the STM32Cube HAL API and the BSP drivers that can be used to build any firmware application.	-		
	Total number of templates_board: 2			0	1	1
Templates	-	Starter project	This project provides a reference template that can be used to build any firmware application.	X	X	X
	Total number of templates: 3			1	1	1
Demonstration	-	Demo	The STM32Cube demonstration platform comes on top of the STM32Cube as a firmware package that offers a full set of software components based on a modular architecture. All modules can be reused separately in standalone applications.	X	-	-
	Total number of demonstrations: 1			1	0	0
Templates_LL	-	Starter project	This project provides a reference template, through the LL API, which can be used to build any firmware application.	X	X	X
	Total number of templates_ll: 3			1	1	1
Examples	-	BSP	How to use the different BSP drivers of the board.	X	-	-
	ADC	ADC_AnalogWatchdog	How to use an ADC peripheral with an ADC analog watchdog to monitor a channel and detect when the corresponding conversion data is outside the window thresholds.	-		-
		ADC_ContinuousConversion_TriggerSW	How to use an ADC peripheral to perform continuous ADC conversions on a channel, from a software start.	-		-
		ADC_MultiChannelSingleConversion	How to use an ADC peripheral to convert several channels. ADC conversions are performed successively in a scan sequence.	-		-
		ADC_Oversampling	How to use an ADC peripheral with oversampling.	-		-
		ADC_SingleConversion_TriggerSW_IT	How to use an ADC to convert a single channel at each software start. The conversion is performed using the interrupt programming model.	-		

Level	Module name	Project name	Description	STM32U0-83C-DK	NUCLEO-U083RC	NUCLEO-U031R8
Examples	ADC	ADC_SingleConversion_TriggerTimer_DMA	How to use an ADC peripheral to perform a single ADC channel conversion at each trigger event from a timer. The converted data is transferred by DMA into a table located in RAM.	MX	-	MX
		ADC_TemperatureSensor	How to use an ADC peripheral to perform a single ADC conversion on the internal temperature sensor and calculate the temperature in degrees Celsius.	MX	-	-
	COMP	COMP_CompareGpioVsDaclnt_OutputGpio	How to configure the comparator peripheral to compare the external voltage applied on a specific pin with a sawtooth signal generated by a DAC.	-	MX	-
		COMP_CompareGpioVsVrefInt_IT	How to use a comparator peripheral to compare a voltage level applied on a GPIO pin to the internal voltage reference (V <sub>REFINT</sub> ), in interrupt mode.	-	MX	MX
		COMP_CompareGpioVsVrefInt_IT_SystemStopMode	How to use a comparator peripheral to compare a voltage level applied on a GPIO pin to the internal voltage reference (V <sub>REFINT</sub> ), in interrupt mode and during Stop mode.	-	MX	-
		COMP_CompareGpioVsVrefInt_OutputGpio	How to use a comparator peripheral to compare a voltage level applied on a GPIO pin to the internal voltage reference (V <sub>REFINT</sub> ) with the comparator output connected to a GPIO pin.	-	-	MX
		COMP_CompareGpioVsVrefInt_Window_IT	How to use a pair of comparator peripherals to compare a voltage level applied on a GPIO pin to two thresholds, the internal voltage reference (V <sub>REFINT</sub> ) and a fraction of the internal voltage reference (V <sub>REFINT</sub> /2), in interrupt mode.	-	MX	-
		COMP_OutputBlanking	How to use the comparator-peripheral output blanking feature. The output blanking feature is used in motor control to prevent the tripping of the current regulation upon short current spikes at the beginning of the PWM period.	-	MX	-
	CORTEX	CORTEXM_MPU	How to configure the MPU attributes of different MPU regions, then a memory area as privileged read-only, and attempt to perform read and write operations in different modes.	-	MX	-
		CORTEXM_ModePrivilege	How to modify the thread mode privilege access and stack. The thread mode is entered on reset or when returning from an exception.	-	MX	-
		CORTEXM_ProcessStack	How to modify the thread mode stack. The thread mode is entered on reset or when returning from an exception	-	MX	-
		CORTEXM_SysTick	How to use the default SysTick configuration with a 1 ms timebase to toggle LEDs.	MX	-	MX
	CRC	CRC_Bytes_Stream_7bit_CRC	How to configure the CRC using the HAL API. The CRC (cyclic redundancy check) calculation unit computes 7-bit CRC codes derived from buffers of 8-bit data (bytes). The user-defined generating polynomial is manually set to 0x65, that is $X^7 + X^6 + X^5 + X^2 + 1$ , as used in the Train Communication Network IEC 60870-5.	-	MX	-
		CRC_Data_Reversing_16bit_CRC	How to configure the CRC using the HAL API. The CRC (cyclic redundancy check) calculation unit computes a 16-bit CRC code derived from a buffer of 32-bit data (words). Input and output data reversal features are enabled. The user-defined generating polynomial is manually set to 0x1021, that is, $X^{16} + X^{12} + X^5 + 1$ , which is the CRC-CCITT generating polynomial.	-	MX	-



Level	Module name	Project name	Description	STM32U0-83C-DK	NUCLEO-U083RC	NUCLEO-U031R8
Examples	CRC	CRC_Example	How to configure the CRC using the HAL API. The CRC (cyclic redundancy check) calculation unit computes the CRC code of a given buffer of 32-bit data words, using a fixed generator polynomial (0x4C11DB7).	MX	MX	MX
		CRC_UserDefinedPolynomial	How to configure the CRC using the HAL API. The CRC (cyclic redundancy check) calculation unit computes the 8-bit CRC code for a given buffer of 32-bit data words, based on a user-defined generating polynomial.	-	MX	-
	CRYP	CRYP_AESModes	How to use the CRYP peripheral to encrypt and decrypt data using the AES in chaining mode (ECB, CBC, CTR).	MX	MX	-
		CRYP_AES_GCM	How to use the CRYP peripheral to encrypt and decrypt data using AES in Galois/Counter mode (GCM).	-	MX	-
		CRYP_GCM_GMAC_CCM_Modes	How to use the CRYP AES peripheral to encrypt and decrypt data as well as compute an authentication tag using the AES-based GCM algorithm with a 128-bit long key, compute an authentication tag using the AES-based GMAC algorithm with a 256-bit long key, encrypt data as well as compute an authentication tag using the AES-based CCM algorithm with a 256-bit long key.	-	MX	-
		CRYP_GCM_GMAC_CCM_Suspension	How to use the CRYP AES peripheral to suspend then resume the AES GCM and GMAC CCM processing of a message to carry out the encryption, decryption or authentication tag computation of a higher-priority message (CCM).	-	MX	-
	DAC	DAC_SignalsGeneration	How to use the DAC peripheral to generate several signals using the DMA controller and the DAC internal wave generator.	MX	MX	MX
		DAC_SimpleConversion	How to use the DAC peripheral to do a simple conversion.	-	MX	-
	DMA	DMA_FLASHToRAM	How to use a DMA to transfer a word data buffer from flash memory to an embedded SRAM, through the HAL API.	MX	MX	MX
	FLASH	FLASH_ChangeOptionBytes	How to configure and modify the option bytes of the flash memory controller.	-	X	-
		FLASH_EraseProgram	How to configure and use the FLASH HAL API to erase and program the internal flash memory. At the beginning of the main program, the HAL_Init() function is called to reset all the peripherals, initialize the flash interface and the SysTick.	MX	MX	-
		FLASH_FastProgram	How to configure and use the FLASH HAL API to erase and fast program the internal flash memory.	-	-	MX
		FLASH_WriteProtection	How to configure and use the FLASH HAL API to enable and disable the write protection of the internal flash memory.	-	MX	-
	GPIO	GPIO_EXTI	How to configure external interrupt lines.	MX	MX	MX



Level	Module name	Project name	Description	STM32U0 83C-DK	NUCLEO- U083RC	NUCLEO- U031R8
Examples	GPIO	GPIO_IOToggle	How to configure and use GPIOs through the HAL API.	MX	MX	MX
	HAL	HAL_TimeBase_RTC_ALARM	How to customize the HAL using the RTC alarm as main source of timebase, instead of the SysTick.	-	MX	-
		HAL_TimeBase_RTC_WKUP	How to customize the HAL using the RTC wake-up as the main source of timebase, instead of the SysTick.	-	MX	-
		HAL_TimeBase_TIM	How to customize the HAL using a general-purpose timer as the main source of timebase instead of the SysTick.	-	MX	-
	I2C	I2C_Sensor_Private_Command_IT	How to handle I2C data buffer transmission/reception between an STM32U0xx Nucleo board and the X-NUCLEO-IKS01A3, using an interrupt.	MX	-	-
		I2C_TwoBoards_AdvComIT	How to handle several I2C data buffer transmission/reception between a master and a slave device, using an interrupt.	-	MX	-
		I2C_TwoBoards_ComDMA	How to handle I2C data buffer transmission/reception between two boards, via DMA.	MX	MX	-
		I2C_TwoBoards_ComIT	How to handle I2C data buffer transmission/reception between two boards, using an interrupt.	-	MX	MX
		I2C_TwoBoards_ComPolling	How to handle I2C data buffer transmission/reception between two boards, in polling mode.	-	MX	-
		I2C_TwoBoards_RestartAdvComIT	How to perform multiple I2C data buffer transmission/reception between two boards, in interrupt mode, and with a restart condition.	-	MX	-
		I2C_TwoBoards_RestartComIT	How to handle single I2C data buffer transmission/reception between two boards, in interrupt mode, and with restart condition.	-	MX	-
		I2C_WakeUpFromStop	How to handle I2C data buffer transmission/reception between two boards, using an interrupt, when the device is in Stop mode.	-	MX	-
		I2C_WakeUpFromStop2	How to handle I2C data buffer transmission/reception between two boards, using an interrupt, when the device is in Stop 2 mode.	-	MX	-
		IWDG	IWDG_Reset	How to handle the IWDG reload counter and simulate a software fault that generates an MCU IWDG reset after a preset laps of time.	-	MX
	IWDG_WindowMode		How to periodically update the IWDG reload counter, and simulate a software fault that generates an MCU IWDG reset after a preset laps of time.	MX	MX	MX
	LCD	LCD_Blink_Frequency	How to use the embedded LCD glass controller and set the LCD blink mode and blinking frequency.	MX	-	-





Level	Module name	Project name	Description	STM32U0 83C-DK	NUCLEO- U083RC	NUCLEO- U031R8
Examples	LCD	LCD_SegmentsDrive	How to use the embedded LCD glass controller to drive the Pacific Display Devices on-board LCD glass.	MX	-	-
	LPTIM	LPTIM_Encoder	How to configure the LPTIM peripheral in encoder mode.	-	MX	-
		LPTIM_PWMExternalClock	How to configure and use, through the HAL LPTIM API, the LPTIM peripheral with an external counter clock, to generate a PWM signal at the lowest power consumption.	-	MX	-
		LPTIM_PWM_LSE	How to configure and use, through the HAL LPTIM API, the LPTIM peripheral with LSE as counter clock, to generate a PWM signal, in a low-power mode.	-	MX	-
		LPTIM_PulseCounter	How to configure and use, through the LPTIM HAL API, the LPTIM peripheral to count pulses.	-	MX	-
		LPTIM_Timeout	How to implement, through the HAL LPTIM API, a timeout with the LPTIM peripheral, to wake up the system from a low-power mode.	MX	MX	-
		OPAMP	OPAMP_Calibration	How to calibrate the OPAMP.	-	MX
	OPAMP_Follower		How to configure the OPAMP peripheral in follower mode interconnected with DAC and COMP.	-	MX	-
	OPAMP_PGA		How to use the built-in PGA mode (OPAMP programmable gain).	MX	MX	-
	PWR	PWR_LPMode_RTC	How to put the system in different available low-power modes, and wake up from these modes by using an interrupt generated by the RTC wake-up timer.	-	MX	-
		PWR_LPRun	How to enter and exit the Low-power run mode.	-	MX	-
		PWR_ModesSelection	How to configure the system to measure the current consumption in different low-power modes.	-	X	-
		PWR_PVD	How to configure the programmable voltage detector by using an external interrupt line. The external DC supply must be used to supply V <sub>DD</sub> .	-	MX	-
		PWR_Shutdown	How to enter Shutdown mode and wake up from this mode using an external reset or a WKUP pin.	-	-	MX
		PWR_Sleep	How to enter Sleep mode and wake up from this mode by using an interrupt.	MX	-	-
		PWR_Standby	How to enter Standby mode and wake up from this mode by using an external reset or the WKUP pin.	MX	-	-

Level	Module name	Project name	Description	STM32U0 83C-DK	NUCLEO- U083RC	NUCLEO- U031R8
Examples	PWR	PWR_STOP0	How to enter Stop 0 mode and wake up from this mode by using an interrupt.	-	MX	-
		PWR_STOP1	How to enter Stop 1 mode and wake up from this mode using an interrupt.	-	MX	-
		PWR_STOP2	How to enter Stop 2 mode and wake up from this mode by using an external reset or a wake-up interrupt.	MX	MX	-
		PWR_STOP2_RTC	How to enter Stop 2 mode and wake up from this mode by using an external reset or the RTC wake-up timer.	-	X	X
	RCC	RCC_CRs_Synchronization_IT	How to configure the clock recovery service (CRS) in interrupt mode, using the RCC HAL API.	-	MX	-
		RCC_CRs_Synchronization_Polling	How to configure the clock recovery system (CRS) in polling mode, using the RCC HAL API.	-	MX	-
		RCC_ClockConfig	How to configure the system clock (SYSCLK) and modify the clock settings in Run mode, using the RCC HAL API.	MX	MX	-
		RCC_LSEConfig	How to enable/disable the low-speed external (LSE) RC oscillator (about 32 KHz) at runtime, using the RCC HAL API.	-	-	MX
		RCC_LSIConfig	How to enable/disable the low-speed internal (LSI) RC oscillator (about 32 KHz) at runtime, using the RCC HAL API.	-	MX	-
		RCC_SwitchClock	How to switch off the system clock (SYSCLK) from low-frequency clock to a high-frequency clock, using the RCC HAL API.	MX	-	MX
	RNG	RNG_Config	How to configure the RNG using the HAL API. This example uses the RNG to generate 32-bit long random numbers.	-	MX	MX
		RNG_MultiRNG	How to configure the RNG using the HAL API. This example uses the RNG to generate 32-bit long random numbers.	-	MX	-
		RNG_MultiRNG_IT	How to configure the RNG using the HAL API. This example uses RNG interrupts to generate 32-bit long random numbers.	-	MX	-
	RTC	RTC_ActiveTamper	How to configure the active tamper detection with backup registers erase.	-	MX	-
		RTC_Alarm	How to configure and generate an RTC alarm using the RTC HAL API.	MX	MX	MX
RTC_Calendar		How to configure the calendar using the RTC HAL API.	-	MX	-	

Level	Module name	Project name	Description	STM32U083C-DK	NUCLEO-U083RC	NUCLEO-U031R8
Examples	RTC	RTC_InternalTimeStamp	Demonstration the internal timestamp feature using the RTC HAL API.	MX	-	-
		RTC_LSI	How to use the LSI clock source autocalibration to get a precise RTC clock.	-	MX	-
		RTC_LowPower_STANDBY	How to enter Standby mode and wake up from this mode using the RTC alarm event.	-	MX	-
		RTC_LowPower_STANDBY_WUT	How to periodically enter and wake up from Standby mode thanks to the RTC wake-up timer (WUT).	MX	MX	-
		RTC_Tamper	How to configure the tamper detection with backup registers erase.	-	MX	-
		RTC_TimeStamp	How to configure the RTC HAL API to demonstrate the timestamp feature.	-	MX	-
	SPI	SPI_FullDuplex_ComDMA_Master	Data buffer transmission/reception between two boards via SPI, using DMA in master mode.	-	MX	-
		SPI_FullDuplex_ComDMA_Slave	Data buffer transmission/reception between two boards via SPI, using DMA in slave mode.	-	MX	-
		SPI_FullDuplex_ComIT	Data buffer transmission/reception between two boards via SPI, using the interrupt mode.	-	X	-
		SPI_FullDuplex_ComIT_Master	Data buffer transmission/reception between two boards via SPI, using the interrupt mode (master).	-	MX	-
		SPI_FullDuplex_ComIT_Slave	Data buffer transmission/reception between two boards via SPI, using the interrupt mode (slave).	-	MX	-
		SPI_FullDuplex_ComPolling_Master	Data buffer transmission/reception between two boards via SPI, using the polling mode (master).	-	MX	-
		SPI_FullDuplex_ComPolling_Slave	Data buffer transmission/reception between two boards via SPI, using the polling mode (slave).	-	MX	-
		SPI_HalfDuplex_ComIT_Master	Data buffer half-duplex transmission/reception between two boards, using an interrupt (master).	-	MX	-
		SPI_HalfDuplex_ComIT_Slave	Data buffer half-duplex transmission/reception between two boards, using an interrupt (slave).	-	MX	-
		SPI_HalfDuplex_ComPolling_Master	Data buffer half-duplex transmission/reception between two boards, in polling mode (master).	-	MX	-

Level	Module name	Project name	Description	STM32U0 83C-DK	NUCLEO- U083RC	NUCLEO- U031R8
Examples	SPI	SPI_HalfDuplex_ComPolling_Slave	Data buffer half-duplex transmission/reception between two boards, in polling mode (slave).	-	MX	-
	TIM	TIM_6Steps	How to configure the TIM1 peripheral to generate 6 steps.	-	MX	-
		TIM_DMA	How to use the DMA with timer update request to transfer data from memory to the timer capture compare register 3 (TIMx_CCR3).	-	MX	-
		TIM_DMABurst	How to update the timer channel 1 period and duty cycle using the timer DMA burst feature.	-	MX	-
		TIM_Encoder	How to configure the TIM1 peripheral in encoder mode to determinate the rotation direction.	-	MX	-
		TIM_ExtTriggerSynchro	How to synchronize the timer peripheral in cascade mode with an external trigger.	-	-	MX
		TIM_OCToggle	How to configure the timer peripheral to generate four different signals at four different frequencies.	-	MX	-
		TIM_OnePulse	How to use the timer peripheral to generate a single pulse when a rising edge of an external signal is received on the timer input pin.	-	MX	-
		TIM_PWMInput	How to use the timer peripheral to measure the frequency and duty cycle of an external signal.	-	-	MX
		TIM_PWMOutput	How to configure the timer peripheral in PWM (pulse width modulation) mode.	-	-	MX
		TIM_TimeBase	How to configure the timer peripheral to generate a timebase of one second with the corresponding interrupt request.	-	-	MX
	UART	LPUART_TwoBoards_ComIT	LPUART transmission (transmit/receive) in interrupt mode between two boards.	-	MX	MX
		LPUART_WakeUpFromStop	How to configure an LPUART to wake up the MCU from Stop mode when a given stimulus is received.	-	MX	MX
		UART_AutoBaudrate_Detection	How to use the HAL UART API for detecting automatically the baud rate.	-	MX	MX
		UART_HyperTerminal_DMA	UART transmission (transmit/receive) in DMA mode between a board and a HyperTerminal PC application.	-	MX	-
		UART_HyperTerminal_IT	UART transmission (transmit/receive) in interrupt mode between a board and a HyperTerminal PC application.	-	MX	MX



Level	Module name	Project name	Description	STM32U0-83C-DK	NUCLEO-U083RC	NUCLEO-U031R8
Examples	UART	UART_LowPower_HyperTerminal_DMA	Low-power UART transmission (transmit/receive) in DMA mode between a board and a HyperTerminal PC application.	MX	-	-
		UART_Printf	Rerouting of the C library printf function to the UART.	-	MX	-
		UART_ReceptionToldle_CircularDMA	How to use the HAL UART API for reception to IDLE event in circular DMA mode.	MX	MX	-
		UART_TwoBoards_ComDMA	UART transmission (transmit/receive) in DMA mode between two boards.	MX	MX	-
		UART_TwoBoards_ComIT	UART transmission (transmit/receive) in interrupt mode between two boards.	-	MX	-
		UART_TwoBoards_ComPolling	UART transmission (transmit/receive) in polling mode between two boards.	-	MX	-
		UART_WakeUpFromStop	How to configure a UART to wake up the MCU from Stop 1 mode when a given stimulus is received.	-	MX	-
	USART	USART_SlaveMode	USART-SPI communication (transmit/receive) between two boards where the USART is configured as a slave.	-	MX	-
		USART_SlaveMode_DMA	USART-SPI communication (transmit/receive) with DMA between two boards where the USART is configured as a slave.	-	MX	-
	WWDG	WWDG_Example	How to configure the HAL API to periodically update the WWDG counter and simulate a software fault that generates an MCU WWDG reset when a predefined time period has elapsed.	-	MX	-
<b>Total number of examples: 159</b>				<b>29</b>	<b>103</b>	<b>27</b>
Examples_LL	ADC	ADC_AnalogWatchdog_Init	How to use an ADC peripheral with an ADC analog watchdog to monitor a channel and detect when the corresponding conversion data is outside the window thresholds.	-	MX	-
		ADC_ContinuousConversion_TriggerSW_Init	How to use an ADC peripheral to convert a single channel continuously, from a software start.	-	MX	-
		ADC_ContinuousConversion_TriggerSW_LowPower_Init	How to use an ADC to convert a single channel with ADC low-power features auto wait and auto power-off.	-	MX	-
		ADC_Oversampling_Init	How to use an ADC peripheral with oversampling.	-	MX	-
		ADC_SingleConversion_TriggerSW_DMA_Init	How to use an ADC peripheral to perform a single ADC conversion on a channel at each software start. The converted data is transferred by DMA into a table located in RAM.	-	MX	-

Level	Module name	Project name	Description	STM32U0-83C-DK	NUCLEO-U083RC	NUCLEO-U031R8
Examples_LL	ADC	ADC_SingleConversion_TriggerSW_IT_Init	How to use ADC to convert a single channel at each software start. The conversion is performed using the interrupt programming model.	-	MX	-
		ADC_SingleConversion_TriggerSW_Init	How to use ADC to convert a single channel at each software start. The conversion is performed using the polling programming model.	-	MX	-
		ADC_SingleConversion_TriggerTimer_DMA_Init	How to use an ADC peripheral to perform a single ADC conversion on a channel at each trigger event from a timer. The converted data is transferred by DMA into a table in RAM.	-	MX	-
		ADC_TemperatureSensor_Init	How to use an ADC peripheral to perform a single ADC conversion on the internal temperature sensor and calculate the temperature in degrees Celsius.	-	MX	-
	COMP	COMP_CompareGpioVsVrefInt_IT_Init	How to use a comparator peripheral to compare a voltage level applied on a GPIO pin to the internal voltage reference (V <sub>REFINT</sub> ), in interrupt mode.	-	MX	-
		COMP_CompareGpioVsVrefInt_OutputGpio_Init	How to use a comparator peripheral to compare a voltage level applied on a GPIO pin to the internal voltage reference (V <sub>REFINT</sub> ) with a comparator output connected to a GPIO pin.	-	MX	-
		COMP_CompareGpioVsVrefInt_Window_IT_Init	How to use a pair of comparator peripherals to compare a voltage level applied on a GPIO pin to two thresholds: the internal voltage reference (V <sub>REFINT</sub> ) and a fraction of the internal voltage reference (V <sub>REFINT</sub> / 2), in interrupt mode.	-	MX	-
	CORTEX	CORTEX_MPU	How to configure the MPU attributes of different MPU regions, then configure a memory area as privileged read-only, and attempt to perform read and write operations in different modes.	-	MX	-
	CRC	CRC_UserDefinedPolynomial	How to configure and use the CRC calculation unit to compute an 8-bit CRC code for a given data buffer, based on a user-defined generating polynomial. The peripheral initialization is done using LL unitary service functions for optimization purposes (performance and size).	-	MX	-
	CRS	CRS_Synchronization_IT	How to configure the clock recovery system in interrupt mode through the STM32U0xx CRS LL API. The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).	-	MX	-
		CRS_Synchronization_Polling	How to configure the clock recovery system in polling mode through the STM32U0xx CRS LL API. The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).	-	MX	-
	DAC	DAC_GenerateConstantSignal_TriggerSW_Init	How to use the DAC peripheral to generate a constant voltage signal.	-	MX	-
DAC_GenerateConstantSignal_TriggerSW_LP_Init		How to use the DAC peripheral to generate a constant voltage signal with the DAC low-power feature sample-and-hold. To be effective, a capacitor must be connected to the DAC channel output, and the sample-and-hold timings must be tuned depending on the capacitor value. This example is based on the STM32U0xx DAC LL API. The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).	-	MX	-	



Level	Module name	Project name	Description	STM32U0-83C-DK	NUCLEO-U083RC	NUCLEO-U031R8
Examples_LL	DAC	DAC_GenerateWaveform_TriggerHW_Init	How to use the DAC peripheral to generate a voltage waveform from a digital data stream transferred by DMA. This example is based on the STM32U0xx DAC LL API. The peripheral initialization uses LL initialization functions to demonstrate LL init usage.	-	MX	-
	DMA	DMA_CopyFromFlashToMemory_Init	How to use a DMA channel to transfer a word data buffer from flash memory to an embedded SRAM. The peripheral initialization uses LL initialization functions to demonstrate LL init usage.	-	MX	-
	EXTI	EXTI_ToggleLedOnIT_Init	How to configure the EXTI and use GPIOs to toggle the user LEDs available on the board when a user button is pressed. This example is based on the STM32U0xx LL API. The peripheral initialization is done using the LL initialization function to demonstrate LL init usage.	-	MX	-
	GPIO	GPIO_InfiniteLedToggling_Init	How to configure and use GPIOs to toggle the on-board user LEDs every 250 ms. This example is based on the STM32U0xx LL API. The peripheral is initialized with the LL initialization function to demonstrate LL init usage.	MX	MX	MX
	I2C	I2C_OneBoard_AdvCommunication_DMAAndIT_Init	How to exchange data between an I2C master device in DMA mode and an I2C slave device in interrupt mode. The peripheral is initialized with LL unitary service functions to optimize for performance and size.	-	MX	-
		I2C_OneBoard_Communication_DMAAndIT_Init	How to transmit data bytes from an I2C master device using DMA mode to an I2C slave device using interrupt mode. The peripheral is initialized with LL unitary service functions to optimize for performance and size.	-	MX	-
		I2C_OneBoard_Communication_IT_Init	How to handle the reception of one data byte from an I2C slave device by an I2C master device. Both devices operate in interrupt mode. The peripheral is initialized with the LL initialization function to demonstrate LL init usage.	-	MX	-
		I2C_OneBoard_Communication_PollingAndIT_Init	How to transmit data bytes from an I2C master device using polling mode to an I2C slave device using interrupt mode. The peripheral is initialized with LL unitary service functions to optimize for performance and size.	-	MX	-
		I2C_TwoBoards_MasterRx_SlaveTx_IT_Init	How to handle the reception of one data byte from an I2C slave device by an I2C master device. Both devices operate in interrupt mode. The peripheral is initialized with LL unitary service functions to optimize for performance and size.	-	MX	-
		I2C_TwoBoards_MasterTx_SlaveRx_DMA_Init	How to transmit data bytes from an I2C master device using DMA mode to an I2C slave device using DMA mode. The peripheral is initialized with LL unitary service functions to optimize for performance and size.	-	MX	-
		I2C_TwoBoards_MasterTx_SlaveRx_Init	How to transmit data bytes from an I2C master device using polling mode to an I2C slave device using interrupt mode. The peripheral is initialized with LL unitary service functions to optimize for performance and size.	-	MX	-
		I2C_TwoBoards_WakeUpFromStop2_IT_Init	How to handle the reception of a data byte from an I2C slave device in Stop 2 mode by an I2C master device, both using interrupt mode. The peripheral is initialized with LL unitary service functions to optimize for performance and size.	-	MX	-
	I2C_TwoBoards_WakeUpFromStop_IT_Init	How to handle the reception of a data byte from an I2C slave device in Stop0 mode by an I2C master device, both using interrupt mode. The peripheral is initialized with LL unitary service functions to optimize for performance and size.	-	MX	-	



Level	Module name	Project name	Description	STM32U0 83C-DK	NUCLEO- U083RC	NUCLEO- U031R8
Examples_LL	LPTIM	LPTIM_PulseCounter_Init	To reduce power consumption, the MCU enters Stop mode after starting counting. Each time the counter reaches the maximum value (period/ autoreload), an interrupt is generated, the MCU is woken up from Stop mode, and LED4 toggles.	-	MX	-
	LPUART	LPUART_WakeUpFromStop2_Init	How to configure the GPIO and LPUART peripherals to allow characters received on LPUART_RX pin to wake up the MCU from low-power mode (Stop 2). This example is based on the LPUART LL API. The peripheral initialization uses the LL initialization function to demonstrate LL init usage.	-	MX	-
		LPUART_WakeUpFromStop_Init	How to configure the GPIO and LPUART peripherals to allow characters received on LPUART_RX pin to wake up the MCU from low-power mode (Stop). This example is based on the LPUART LL API. The peripheral initialization uses the LL initialization function to demonstrate LL init usage.	-	MX	-
	OPAMP	OPAMP_Follower_Init	How to use the OPAMP peripheral in follower mode interconnected with DAC and COMP. This example is based on the STM32U0xx OPAMP LL API. The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).	-	MX	-
		OPAMP_PGA_Init	How to use the OPAMP peripheral in PGA mode (OPAMP programmable gain) with DAC and COMP.	-	MX	-
	PWR	PWR_EnterStandbyMode	How to enter Standby mode and wake up from this mode by using an external reset or a wakeup pin.	-	MX	-
		PWR_EnterStopMode	How to enter Stop 0 mode.	-	MX	-
		PWR_LPRunMode_SRAM1	How to execute code in Low-power run mode from SRAM1.	-	MX	-
		PWR_OptimizedRunMode	How to increase/decrease frequency and V <sub>CORE</sub> and enter/exit the Low-power run mode.	-	MX	-
	RCC	RCC_HWAutoMSICalibration	How to use the MSI clock source hardware autocalibration and LSE clock (PLL mode) to obtain a precise MSI clock.	-	MX	-
		RCC_OutputSystemClockOnMCO	How to configure the MCO pin (PA8) to output the system clock.	-	MX	-
		RCC_UseHSI_PLLasSystemClock	How to modify the PLL parameters in runtime.	-	MX	-
	RNG	RNG_GenerateRandomNumbers	How to configure the RNG to generate 32-bit long random numbers. The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).	-	MX	-
		RNG_GenerateRandomNumbers_IT	How to configure the RNG to generate 32-bit long random numbers using interrupts. The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).	-	MX	-



Level	Module name	Project name	Description	STM32U0 83C-DK	NUCLEO- U083RC	NUCLEO- U031R8
Examples_LL	RTC	RTC_Alarm_Init	How to configure the RTC LL API to configure and generate an alarm using the RTC peripheral. The peripheral initialization uses the LL initialization function.	-	MX	-
		RTC_Calendar_Init	How to configure the LL API to set the RTC calendar. The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).	-	MX	-
		RTC_ExitStandbyWithWakeUpTimer_Init	How to periodically enter and wake up from Standby mode thanks to the RTC wake-up timer (WUT).	-	MX	-
		RTC_Tamper_Init	How to configure the tamper using the RTC LL API. The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).	-	MX	-
		RTC_TimeStamp_Init	How to configure the timestamp using the RTC LL API. The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).	-	MX	-
	SPI	SPI_OneBoard_HalfDuplex_DMA_Init	How to configure the GPIO and SPI peripherals to transmit bytes from an SPI master device to an SPI slave device in DMA mode. This example is based on the STM32U0xx SPI LL API. The peripheral initialization uses the LL initialization function to demonstrate LL init usage.	-	MX	-
		SPI_OneBoard_HalfDuplex_IT_Init	How to configure the GPIO and SPI peripherals to transmit bytes from an SPI master device to an SPI slave device in interrupt mode. This example is based on the STM32U0xx SPI LL API. The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).	-	MX	-
		SPI_TwoBoards_FullDuplex_DMA_Master_Init	Data buffer transmission and reception via SPI using DMA mode. This example is based on the STM32U0xx SPI LL API. The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).	-	-	MX
		SPI_TwoBoards_FullDuplex_DMA_Slave_Init	Data buffer transmission and reception via SPI using DMA mode. This example is based on the STM32U0xx SPI LL API. The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).	-	-	MX
		SPI_TwoBoards_FullDuplex_IT_Master_Init	Data buffer transmission and reception via SPI using interrupt mode (master). This example is based on the STM32U0xx SPI LL API. The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).	-	MX	-
		SPI_TwoBoards_FullDuplex_IT_Slave_Init	Data buffer transmission and reception via SPI using interrupt mode (slave). This example is based on the STM32U0xx SPI LL API. The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).	-	MX	-
	TIM	TIM_BreakAndDeadtime_Init	How to configure the timer peripheral to generate three center-aligned PWM and complementary PWM signals, insert a defined deadtime value, use the break feature, and lock the break and dead-time configuration.	-	MX	-
		TIM_DMA_Init	How to use of the DMA with a timer update request to transfer data from the memory to the timer capture compare register 3 (TIMx_CCR3). This example is based on the STM32U0xx TIM LL API. The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).	-	MX	-

Level	Module name	Project name	Description	STM32U0-83C-DK	NUCLEO-U083RC	NUCLEO-U031R8
Examples_LL	TIM	TIM_InputCapture_Init	How to use the timer peripheral to measure a periodic signal frequency provided either by an external signal generator or by another timer instance. This example is based on the STM32U0xx TIM LL API. The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).	-	MX	-
		TIM_OnePulse_Init	How to configure a timer to generate a positive pulse in output compare mode with a length of $t_{PULSE}$ and after a delay of $t_{DELAY}$ . This example is based on the STM32U0xx TIM LL API. The peripheral initialization uses the LL initialization function to demonstrate LL Init.	-	MX	-
		TIM_OutputCompare_Init	How to configure the timer peripheral to generate an output waveform in different output compare modes. This example is based on the STM32U0xx TIM LL API. The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).	-	MX	-
		TIM_PWMOutput_Init	How to use of the timer peripheral to generate a PWM output signal and update the PWM duty cycle. This example is based on the STM32U0xx TIM LL API. The peripheral initialization uses the LL initialization function to demonstrate LL Init.	-	MX	-
		TIM_TimeBase_Init	How to configure the timer peripheral to generate a timebase. This example is based on the STM32U0xx TIM LL API. The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).	-	MX	-
	USART	USART_Communication_Rx_IT_Continuous_Init	How to configure the GPIO and USART peripherals to continuously receive characters from a HyperTerminal (PC) in asynchronous mode and using the interrupt mode. The peripheral initialization is done using LL unitary services functions for optimization purposes (performance and size).	-	MX	-
		USART_Communication_Rx_IT_Continuous_VCP_Init	How to configure the GPIO and USART peripherals to continuously receive characters from a HyperTerminal (PC) in asynchronous mode and using the interrupt mode (VCP). The peripheral initialization is done using LL unitary services functions for optimization purpose (performance and size).	-	MX	-
		USART_Communication_Rx_IT_Init	How to configure the GPIO and USART peripherals to receive characters from a HyperTerminal (PC) in asynchronous mode using the interrupt mode. The peripheral initialization is done using the LL initialization function to demonstrate LL init usage.	-	MX	-
		USART_Communication_Rx_IT_VCP_Init	How to configure the GPIO and USART peripherals to receive characters from a HyperTerminal (PC) in asynchronous mode using the interrupt mode (VCP). The peripheral initialization is done using LL initialization function to demonstrate LL init usage.	-	MX	-
		USART_Communication_TxRx_DMA_Init	How to configure the GPIO and USART peripherals to send characters asynchronously to/from a HyperTerminal (PC) in DMA mode. This example is based on the STM32U0xx USART LL API. Peripheral initialization is done using LL unitary services functions for optimization purposes (performance and size).	-	MX	-
		USART_Communication_Tx_IT_Init	How to configure the GPIO and USART peripherals to send characters asynchronously to a HyperTerminal (PC) in interrupt mode. This example is based on the STM32U0xx USART LL API. Peripheral initialization is done using LL unitary services functions for optimization purposes (performance and size).	-	MX	-

Level	Module name	Project name	Description	STM32U0 83C-DK	NUCLEO- U083RC	NUCLEO- U031R8
Examples_LL	USART	USART_Communication_Tx_IT_VCP_Init	How to configure the GPIO and USART peripherals to send characters asynchronously to a HyperTerminal (PC) in interrupt mode (VCP). This example is based on STM32U0xx USART LL API. The peripheral initialization is done using LL unitary services functions for optimization purpose (performance and size).	-	MX	-
		USART_Communication_Tx_Init	How to configure the GPIO and USART peripherals to send characters asynchronously to a HyperTerminal (PC) in polling mode. If the transfer cannot be completed within the allocated time, a timeout allows to exit from the sequence with a timeout error code. This example is based on the STM32U0xx USART LL API. The peripheral initialization is done using LL unitary services functions for optimization purposes (performance and size).	-	MX	-
		USART_Communication_Tx_VCP_Init	How to configure the GPIO and USART peripherals to send characters asynchronously to a HyperTerminal (PC) in polling mode (VCP). If the transfer cannot be completed within the allocated time, a timeout allows to exit from the sequence with a timeout error code. This example is based on STM32U0xx USART LL API. The peripheral initialization is done using LL unitary services functions for optimization purpose (performance and size).	-	MX	-
		USART_HardwareFlowControl_Init	How to configure the GPIO and USART peripherals to receive characters asynchronously from a HyperTerminal (PC) in interrupt mode with the hardware flow control feature enabled. This example is based on the STM32U0xx USART LL API. The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).	-	MX	-
		USART_SyncCommunication_FullDuplex_DMA_Init	How to configure the GPIO, USART, DMA, and SPI peripherals to transmit bytes between a USART and an SPI (in slave mode) in DMA mode. This example is based on the STM32U0xx USART LL API. The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).	-	MX	-
		USART_SyncCommunication_FullDuplex_IT_Init	How to configure the GPIO, USART, DMA, and SPI peripherals to transmit bytes between a USART and an SPI (in slave mode) in interrupt mode. This example is based on the STM32U0xx USART LL API (the SPI uses the DMA to receive/transmit characters sent from/received by the USART). The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).	-	MX	-
		USART_WakeUpFromStop1_Init	How to configure the GPIO and USART2 peripherals to allow the characters received on the USART_RX pin to wake up the MCU from low-power mode (Stop 1).	-	MX	-
		USART_WakeUpFromStop_Init	How to configure the GPIO and USART2 peripherals to allow the characters received on the USART_RX pin to wake up the MCU from low-power mode.	-	MX	-
	UTILS	UTILS_ConfigureSystemClock	How to use the UTILS LL API to configure the system clock using PLL with HSI as the source clock.	-	MX	-
		UTILS_ReadDeviceInfo	How to read the UID, Device ID and Revision ID and save them into a global information buffer.	-	MX	-
WWDG	WWDG_RefreshUntilUserEvent_Init	How to configure the WWDG to periodically update the counter and generate an MCU WWDG reset when a user button is pressed. The peripheral initialization uses the LL unitary service functions for optimization purposes (performance and size).	-	MX	-	



Level	Module name	Project name	Description	STM32U083C-DK	NUCLEO-U083RC	NUCLEO-U031R8
<b>Examples_LL</b>	<b>Total number of examples_LL: 82</b>			<b>1</b>	<b>78</b>	<b>3</b>
<b>Examples_MIX</b>	ADC	ADC_SingleConversion_TriggerSW_IT	How to use ADC to convert a single channel at each software start. The conversion is performed using the programming model: interrupt.	-	<b>MX</b>	-
	CRC	CRC_PolynomialUpdate	How to use the CRC peripheral through the STM32U0xx CRC HAL and LL API.	-	<b>MX</b>	-
	DMA	DMA_FLASHToRAM	How to use a DMA to transfer a word data buffer from flash memory to embedded SRAM through the STM32U0xx DMA HAL and LL API. The LL API is used for performance improvement.	-	<b>MX</b>	-
	I2C	I2C_OneBoard_ComSlave7_10bits_IT	How to perform I2C data buffer transmission/reception between one master and two slaves with different address sizes (7-bit or 10-bit). This example uses the STM32U0xx I2C HAL and LL API (LL API usage for performance improvement) and an interrupt.	-	<b>MX</b>	-
	OPAMP	OPAMP_Calibration	How to calibrate the OPAMP.	-	<b>MX</b>	-
	PWR	PWR_STANDBY_RTC	How to enter Standby mode and wake up from this mode by using an external reset or the RTC wake-up timer through the STM32U0xx RTC and RCC HAL, and LL API (LL API used for maximizing performance).	-	<b>MX</b>	-
		PWR_STOP1	How to enter Stop 1 mode and wake up from this mode by using an external reset or wake-up interrupt (all the RCC function calls use RCC LL API for minimizing footprint and maximizing performance).	-	<b>MX</b>	-
	SPI	SPI_FullDuplex_ComPolling_Master	Data buffer transmission/reception between two boards via SPI using polling mode (master).	-	<b>MX</b>	-
		SPI_FullDuplex_ComPolling_Slave	Data buffer transmission/reception between two boards via SPI using polling mode (slave).	-	<b>MX</b>	-
		SPI_HalfDuplex_ComPollingIT_Master	Data buffer transmission/reception between two boards via SPI using polling (LL driver) and interrupt modes (HAL driver), and in master mode.	-	<b>MX</b>	-
		SPI_HalfDuplex_ComPollingIT_Slave	Data buffer transmission/reception between two boards via SPI using polling (LL driver) and interrupt modes (HAL driver), and in master mode.	-	<b>MX</b>	-
	TIM	TIM_PWMInput	How to use the timer peripheral to measure an external signal frequency and duty cycle.	-	<b>MX</b>	-
	UART	UART_HyperTerminal_IT	How to use a UART to transmit data (transmit/receive) between a board and a HyperTerminal PC application in interrupt mode. This example describes how to use the USART peripheral through the STM32U0xx UART HAL and LL API, the LL API being used for performance improvement.	-	<b>MX</b>	-
		UART_HyperTerminal_TxPolling_RxIT	How to use a UART to transmit data (transmit/receive) between a board and a HyperTerminal PC application both in polling and interrupt modes. This example describes how to use the USART peripheral through the STM32U0xx UART HAL and LL API, the LL API being used for performance improvement.	-	<b>MX</b>	-



Level	Module name	Project name	Description	STM32U083C-DK	NUCLEO-U083RC	NUCLEO-U031R8
Examples_MIX	Total number of examples_mix: 14			0	14	0
Applications	-	OpenBootloader	This application uses the OpenBootloader middleware to demonstrate how to develop an IAP application and how to use it.	X	-	-
	FileX	Fx_SRAM_File_Edit_Standalone	This application provides an example of FileX stack usage on the NUCLEO-U083RC board, running in standalone mode (without ThreadX). It demonstrates how to create a Fat File system on the internal SRAM using FileX.	-	MX	-
	ROT	OEMiROT_Appli	This project provides a OEMiROT boot path application example. The boot is performed through OEMiROT boot path after authenticity and the integrity checks of the project firmware and project data image.	-	X	-
		OEMiROT_Boot	This project provides an OEMiROT example. The OEMiROT boot path performs the authenticity and the integrity checks of the project firmware and data images.	-	X	-
		OEMiROT_Loader	This application is a sample code of a standalone local loader using the Ymodem protocol. This application allows the download of a new version of the firmware and data images.	-	X	-
		OEMiSB_Appli	This project provides a OEMiSB boot path application example. The boot is performed through the OEMiSB boot path after integrity checks of the project firmware image.	-	-	X
		OEMiSB_Boot	This project provides an OEMiSB example. The OEMiSB boot path performs the authenticity check of the project firmware image.	-	-	X
		Tx_CMSIS_Wrapper	This application provides an example of CMSIS RTOS adaptation layer for Azure® RTOS ThreadX. It shows how to develop an application using the CMSIS RTOS 2 APIs.	X	-	-
	ThreadX	Tx_FreeRTOS_Wrapper	This application provides an example of Azure® RTOS ThreadX stack usage, it shows how to develop an application using the FreeRTOS™ adaptation layer for ThreadX.	-	X	-
		Tx_LowPower	This application provides an example of Azure® RTOS ThreadX stack usage. It shows how to develop an application using ThreadX low power feature.	MX	-	-
		Tx_Thread_Creation	This application provides an example of Azure® RTOS ThreadX stack usage. It shows how to develop an application using the ThreadX thread management APIs.	-	MX	-
		Tx_Thread_MsgQueue	This application provides an example of Azure® RTOS ThreadX stack usage. It shows how to develop an application using the ThreadX message queue APIs.	-	MX	-
		Tx_Thread_Sync	This application provides an example of Azure® RTOS ThreadX stack usage. It shows how to develop an application using the ThreadX synchronization APIs.	-	MX	-
	TouchSensing	TouchSensing_1touchkey	How to use the TSC to perform continuous acquisitions of one channel in interrupt mode.	MX	-	-



Level	Module name	Project name	Description	STM32U083C-DK	NUCLEO-U083RC	NUCLEO-U031R8
Applications	TouchSensing	TouchSensing_1touchkey_LowPower	How to use of the TSC to perform continuous acquisitions of one channel in interrupt mode.	MX	-	-
	USBX	Ux_Device_CDC_ACM	This application provides an example of Azure® RTOS USBX stack usage on the STM32U083C-DK board. It shows how to develop a USB Device communication Class CDC_ACM-based application.	MX	-	-
		Ux_Device_HID	This application provides an example of Azure® RTOS USBX stack usage on the STM32U083C-DK board. It shows how to develop a USB Device Human Interface HID mouse-based application.	MX	-	-
		Ux_Device_HID_CDC_ACM	This application provides an example of Azure® RTOS USBX stack usage on the STM32U083C-DK board. It shows how to develop a composite USB Device communication Class HID and CDC_ACM based application.	MX	-	-
		Ux_Device_HID_Standalone	This application provides an example of Azure® RTOS USBX stack usage on the STM32U083C-DK board. It shows how to develop a USB Device Human Interface HID mouse based bare metal application.	MX	-	-
	<b>Total number of applications: 20</b>				<b>10</b>	<b>8</b>
<b>Total number of projects: 285</b>				<b>42</b>	<b>207</b>	<b>35</b>



## Revision history

**Table 2. Document revision history**

Date	Version	Changes
26-Feb-2024	1	Initial release

## Contents

<b>1</b>	<b>Reference documents .....</b>	<b>2</b>
<b>2</b>	<b>STM32CubeU0 examples .....</b>	<b>3</b>
	<b>Revision history .....</b>	<b>23</b>
	<b>List of tables .....</b>	<b>25</b>



## List of tables

<b>Table 1.</b>	STM32CubeU0 firmware examples . . . . .	5
<b>Table 2.</b>	Document revision history . . . . .	23

**IMPORTANT NOTICE – READ CAREFULLY**

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to [www.st.com/trademarks](http://www.st.com/trademarks). All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2024 STMicroelectronics – All rights reserved