UM2561

User manual

Getting started with the X-CUBE-OUT2 industrial digital output software expansion for STM32Cube

Introduction

The X-CUBE-OUT2 expansion software package for STM32Cube runs on the STM32 and includes a driver for the ISO8200AQ.

The software provides an affordable and easy-to-use solution for the development of 8-channel digital output modules, letting you easily evaluate the ISO8200AQ communication and industrial load driving features.

The expansion is built on STM32Cube software technology to ease portability across different STM32 microcontrollers.

The software comes with a sample implementation of the driver running on the X-NUCLEO-OUT02A1 expansion board connected to a NUCLEO-F401RE or NUCLEO-F334R8 development board.

You can also perform evaluation of 16-channel digital output modules by connecting two X-NUCLEO-OUT02A1 and activating the daisy chaining feature.

1 Acronyms and abbreviations

Table 1. List of acronyms

Acronym	Description
API	Application programming interface
BSP	Board support package
CMSIS	Cortex® microcontroller software interface standard
HAL	Hardware abstraction layer
SPI	Serial port interface
IDE	Integrated development environment
LED	Light emitting diode

2 What is STM32Cube?

STM32Cube™ represents the STMicroelectronics initiative to make developers' lives easier by reducing development effort, time and cost. STM32Cube covers the STM32 portfolio.

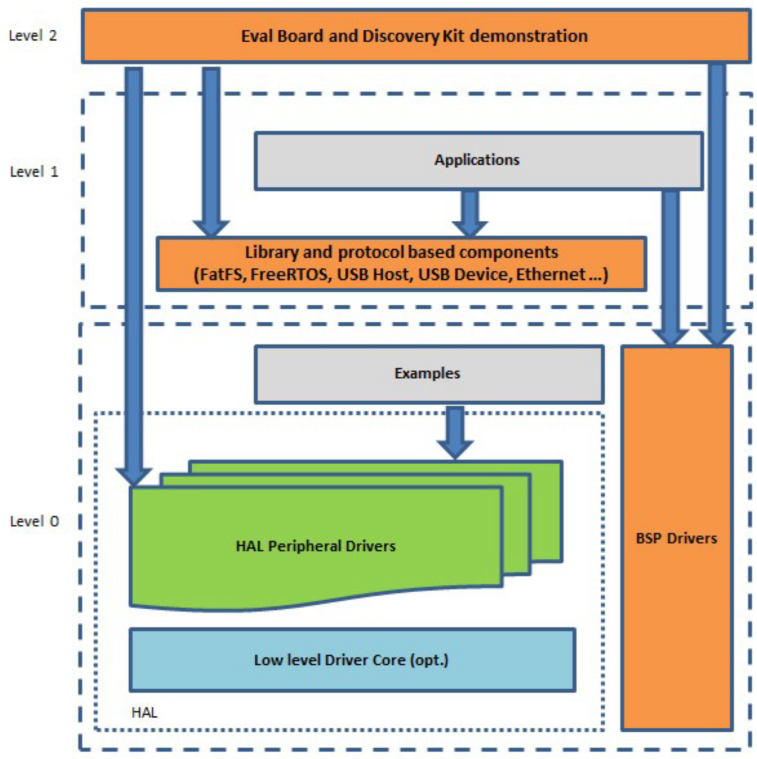
STM32Cube version 1.x includes:

- STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform specific to each series (such as the STM32CubeF4 for the STM32F4 series), which includes:
  - the STM32Cube HAL embedded abstraction-layer software, ensuring maximized portability across the STM32 portfolio
  - a consistent set of middleware components such as RTOS, USB, TCP/IP and graphics
  - all embedded software utilities with a full set of examples

2.1 STM32Cube architecture

The STM32Cube firmware solution is built around three independent levels that can easily interact with one another, as described in the diagram below.

Figure 1. Firmware architecture


Level 2
Eval Board and Discovery Kit demonstration
Level 1
Applications
Library and protocol based components
(FatFS, FreeRTOS, USB Host, USB Device, Ethernet ...)
Level 0
Examples
HAL Peripheral Drivers
Low level Driver Core (opt.)
HAL
BSP Drivers

Level 0: This level is divided into three sub-layers:

- Board Support Package (BSP): this layer offers a set of APIs relative to the hardware components in the hardware boards (Audio codec, IO expander, Touchscreen, SRAM driver, LCD drivers. etc…); it is based on modular architecture allowing it to be easily ported on any hardware by just implementing the low level routines. It is composed of two parts:

  - Component: is the driver relative to the external device on the board and not related to the STM32, the component driver provides specific APIs to the external components of the BSP driver, and can be ported on any other board.
  - BSP driver: links the component driver to a specific board and provides a set of easy to use APIs. The API naming convention is BSP_FUNCT_Action(): e.g., BSP_LED_Init(), BSP_LED_On().
- Hardware Abstraction Layer (HAL): this layer provides the low level drivers and the hardware interfacing methods to interact with the upper layers (application, libraries and stacks). It provides generic, multi-instance and function-oriented APIs to help offload user application development time by providing ready to use processes. For example, for the communication peripherals (I^2C, UART, etc.) it provides APIs for peripheral initialization and configuration, data transfer management based on polling, interrupt or DMA processes, and communication error management. The HAL Drivers APIs are split in two categories: generic APIs providing common, generic functions to all the STM32 series and extension APIs which provide special, customized functions for a specific family or a specific part number.
- Basic peripheral usage examples: this layer houses the examples built around the STM32 peripherals using the HAL and BSP resources only.

Level 1: This level is divided into two sub-layers:

- Middleware components: set of libraries covering USB Host and Device Libraries, STemWin, FreeRTOS, FatFS, LwIP, and PolarSSL. Horizontal interaction among the components in this layer is performed directly by calling the feature APIs, while vertical interaction with low-level drivers is managed by specific callbacks and static macros implemented in the library system call interface. For example, FatFs implements the disk I/O driver to access a microSD drive or USB Mass Storage Class.
- Examples based on the middleware components: each middleware component comes with one or more examples (or applications) showing how to use it. Integration examples that use several middleware components are provided as well.

Level 2: This level is a single layer with a global, real-time and graphical demonstration based on the middleware service layer, the low level abstraction layer and basic peripheral usage applications for board-based functions.

3 X-CUBE-OUT2 software expansion for STM32Cube

3.1 Overview

X-CUBE-OUT2 is a software package that expands the functionality provided by STM32Cube .

The key features of the package are:

- Complete software to build applications for the ISO8200AQ galvanic isolated octal high-side smart power solid state-relay
- GPIOs, SPI, PWMs and IRQs configuration
- Fault and Power Good interrupts handling
- Daisy chaining support
- Sample implementation available on the X-NUCLEO-OUT02A1 expansion board when connected to a NUCLEO-F401RE or NUCLEO-F334R8 development board
- Easy portability across different MCU families, thanks to STM32Cube
- Free, user-friendly license terms

This software enables control of the digital output 8 channels in singular or grouped manner. It also allows programming them to be switched on and off periodically. In this last case, a specific frequency and duty cycle can be set for each channel.

The package also includes an example wich applies different patterns on the 8-channel digital output. A new pattern is applied every time the STM32 Nucleo board user button is pressed.

3.2 Architecture

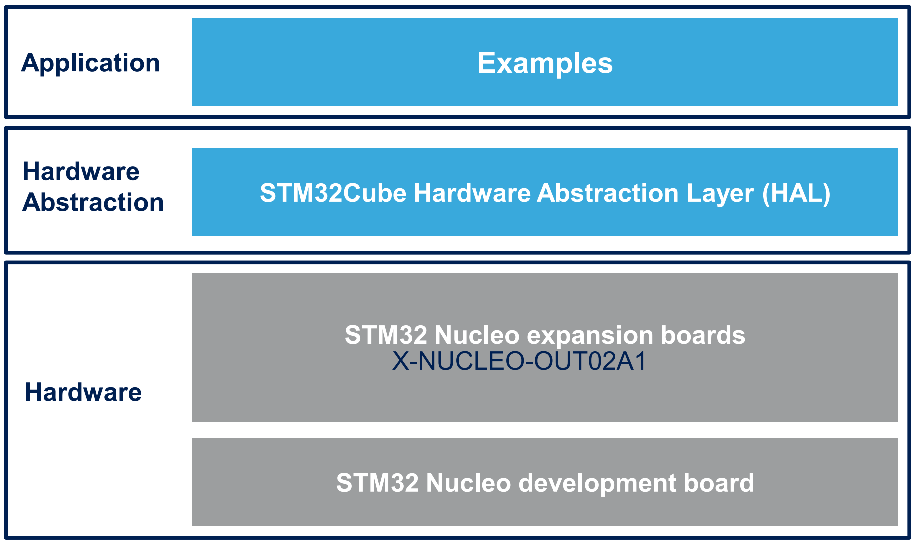
This software is a fully compliant expansion of STM32Cube architecture for the development of applications for 8-channel digital output modules.

The software is based on the STM32CubeHAL hardware abstraction layer for the STM32 microcontroller. The package extends STM32Cube by providing a board support package (BSP) for the digital output expansion board based on ISO8200AQ.

The software layers used by the application software to access and use the industrial digital output expansion board are:

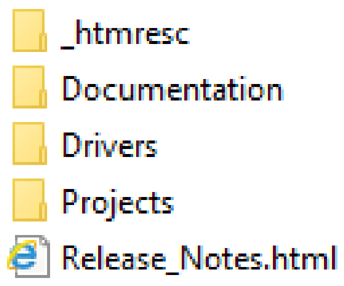
- **STM32Cube HAL layer**: consists of simple, generic and multi-instance APIs (application programming interfaces) which interact with the upper layer applications, libraries and stacks. These generic and extension APIs are based on a common framework so that overlying layers like middleware can function without requiring specific microcontroller unit (MCU) hardware information. This structure improves library code reusability and guarantees easy portability across other devices.
- **Board support package (BSP) layer**: provides software support for the STM32 Nucleo board peripherals, excluding the MCU. These specific APIs provide a programming interface for certain board specific peripherals like LEDs, user buttons, etc., and can also be used to fetch individual board version information. It also provides support for initializing, configuring and reading data.

Figure 2. X-CUBE-OUT2 software architecture


Application
Examples
Hardware Abstraction
STM32Cube Hardware Abstraction Layer (HAL)
Hardware
STM32 Nucleo expansion boards
X-NUCLEO-OUT02A1
STM32 Nucleo development board

3.3 Folder structure

Figure 3. X-CUBE-OUT2 package folder structure



The following folders are included in the software package:

- **Documentation** contains a compiled HTML file generated from the source code, detailing the software components and APIs.
- **Drivers** contains:
  - an **STM32Cube HAL** folder, located in the subfolders STM32F3xx_HAL_Driver or STM32F4xx_HAL_Driver. These files are not described here as they are not specific to the X-CUBE-OUT2 software but come directly from the STM32Cube framework.
  - a **CMSIS** folder which contains the Cortex® microcontroller software interface standard files from ARM. These files are vendor-independent hardware abstraction layer for the Cortex-M processor series. This folder comes also unchanged from the STM32Cube framework.
  - a **BSP** folder containing the codes required for the X-NUCLEO-OUT02A1 configuration, the ISO8200AQ driver and the relay API.
- **Projects** contains sample applications for the ISO8200AQ, provided for NUCLEO-F334R8 and NUCLEO-F401RE platforms.

3.3.1 BSP folder

The X-CUBE-OUT2 software uses four different BSP:

1. STM32F3XX-Nucleo/STM32F4XX-Nucleo BSPs
2. ST Relay BSP
3. ISO8200AQ BSP component
4. X-NUCLEO-OUT02A1 BSP component

3.3.1.1 *STM32F3XX-Nucleo/STM32F4XX-Nucleo BSPs*

Depending on the STM32 Nucleo board used, these BSPs provide an interface to configure and use the board peripherals with the X-NUCLEO-OUT02A1 expansion board.

Each STM32FXX-Nucleo subfolder contains couples of **stm32XXxx_nucleo.c/h** files which come from the STM32Cube framework without modification and provide the functions to handle the corresponding STM32 Nucleo board user button and LEDs.

3.3.1.2 *ST Relay BSP*

This BSP provides a common interface to access the driver functions of various ST relays, via a couple of *STRelay/ strelay.c/h* files.

The files define all the functions to configure and control the power switch driver and mapped to the functions of the relay driver component used on the given expansion board via the structure file `relayDrv_t` (defined in *Components\Common\relay.h.*).

The structure defines a list of function pointers filled during its instantiation in the corresponding relay driver component: for theX-CUBE-OUT2, the instance of the structure is called `iso8200aqDrv` (see file: *BSP \Components\ iso8200aq\ iso8200aq.c*).

As the relay control BSP is common for all relay driver expansion board, not all its functions are available for a given expansion board. Therefore, during the instantiation of the `relayDrv _t` structure in the driver component, the unavailable functions are replaced by a null pointer.

3.3.1.3 *ISO8200AQ BSP component*

The ISO8200AQ BSP component provides the driver functions of the ISO8200AQ power switch driver in the folder *stm32_cube\Drivers\BSP\Components\ ISO8200AQ*.

This folder contains 3 files:

- **iso8200aq.c**: ISO8200AQ driver core functions
- **iso8200aq.h**: declaration of the ISO8200AQ driver functions and their associated definitions
- **iso8200aq _target_config.h**: predefines values for the ISO8200AQ parameters and for the relay device context

3.3.1.4 *X-NUCLEO-OUT02A1 BSP component*

The X-NUCLEO-OUT02A1 BSP component contains a couple of *x_nucleo_out02a1.c/h* files dedicated to the configuration of the SPI, the timers, the GPIOs, the interrupt enabling/disabling required for the X-NUCLEO-OUT02A1 expansion board.

3.3.2 Projects folder

For each STM32 Nucleo platform, one example project is available from the folder *\Projects\ Multi\Examples \PowerManagement*:

- **OUT02A1_ExampleWith1Driver**: examples of control functions in configuration with one ISO8200AQ

Each example has a folder dedicated to the targeted IDE:

- **EWARM** containing the project files for IAR
- **MDK-ARM** containing the project files for Keil
- **SW4STM32** containing the project files for OpenSTM32

Each example also contains the following code files:

- **inc\main.h**: main header file
- **inc\ stm32xxxx_hal_conf.h**: HAL configuration file
- **inc\stm32xxxx_it.h**: header for the interrupt handler
- **src\main.c**: main program (code of the example which is based on the library for ISO8200AQ)
- **src\stm32xxxx_hal_msp.c**: HAL initialization routines
- **src\stm32xxxx_it.c**: interrupt handler
- **src\system_stm32xxxx.c**: system initialization
- **src\clock_xx.c**: clock initialization

3.4 Software required resources

The control of the ISO8200AQ by the MCU and their communication is performed via SPI and GPIOs.

For a ISO8200AQ device (one X-NUCLEO-OUT02A1 expansion board), this requires the use of the 4 SPI signals (MISO, MOSI, CLK and SS) plus 3 GPIOs (EN, FAULT and PGOOD pins).

The *\Drivers\BSP\X_NUCLEO_OUT02A1\x_nucleo_out02a1.h* file defines the used resources.

It handles up to two devices (such as in the case of the daisy chaining with two boards).

It is easily possible to change the used SPI (which is common to all boards) and the GPIOs which are used by each device.

The software default configuration is aligned with the X-NUCLEO-OUT02A1 expansion board software.

The software alternate configurations are set by changing some resistors on the board (for further details, refer to X-NUCLEO-OUT02A1 user manual freely available at www.st.com).

The following table details the possible software configurations and the *flagS* which have to be defined to use them.

Table 2. Default and alternative software configurations for the resources

Signal/Pin	Flag to enable/ Used resources	Default configuration	Alternate configuration 1	Alternate configuration 2
SPI (common to all devices)	Flag to enable	BSP_ALL_DEVICES_USE_SPI1	BSP_ALL_DEVICES_USE_SPI2	None
	Used resources	SPI1 with: CLK: PA5 MISO: GPIO PA6 MOSI: GPIO PA7	SPI2 with: CLK: PB13 MISO: GPIO PC2 MOSI: GPIO PC3	None
SS (common to all devices)	Flag to enable	BSP_ALL_DEVICES_USE_SPI_SS_DEFAULT	BSP_RELAY_BOARD_SPI_SS_ALT1_PIN	BSP_RELAY_BOARD_SPI_SS_ALT2_PIN
	Used resources	GPIO PB6	GPIO PA4	GPIO PA8
FAULT PIN	Flag to enable	BSP_DEVICE_0_USE_FAULT_DEFAULT (for 1st device) BSP_DEVICE_1_USE_FAULT_DEFAULT (for 2nd device)	BSP_DEVICE_0_USE_FAULT_ALT1 (for 1st device) BSP_DEVICE_1_USE_FAULT_ALT1 (for 2nd device)	BSP_DEVICE_0_USE_FAULT_ALT2 (for 1st device) BSP_DEVICE_1_USE_FAULT_ALT2 (for 2nd device)
	Used resources	GPIO PB0	GPIO PC1	GPIO PB10
PGOOD PIN	Flag to enable	BSP_DEVICE_0_USE_PGOOD_DEFAULT (for 1st device) BSP_DEVICE_1_USE_PGOOD_DEFAULT (for 2nd device)	BSP_DEVICE_0_USE_PGOOD_ALT1 (for 1st device) BSP_DEVICE_1_USE_PGOOD_ALT1 (for 2nd device)	BSP_DEVICE_0_USE_PGOOD_ALT2 (for 1st device) BSP_DEVICE_1_USE_PGOOD_ALT2 (for 2nd device)
	Used resources	GPIO PB9	GPIO PA1	GPIO PA10
EN PIN (Enable)	Flag to enable	BSP_DEVICE_0_USE_OUT_EN_DEFAULT (for device 1) BSP_DEVICE_1_USE_PGOOD_DEFAULT (for 2nd device)	BSP_DEVICE_0_USE_OUT_EN_ALT1 (for device 1) BSP_DEVICE_1_USE_PGOOD_ALT1 (for 2nd device)	BSP_DEVICE_0_USE_OUT_EN_ALT2 (for device 1) BSP_DEVICE_1_USE_PGOOD_ALT2 (for 2nd device)
	Used resources	GPIO PB5	GPIO PA0	GPIO PC0

For the handling of the FAULT and PGOOD signals, the software enables the associated interrupt line.

A customized handler can be associated to these interrupts via `BSP_STRelay_AttachFaultInterrupt` and `BSP_STRelay_AttachPGoodInterrupt` functions.

The software also uses two timers:

- **TIM3**: a PWM timer to generate the periodic patterns on the different output channels
- **TIM4** (for STM32F401) and **TIM16** (for STM32F334): a guard timer to insert a delay between two requests on the SPI bus

3.5 APIs

The X-CUBE-OUT2 software API is defined in the ST Relay BSP. Its functions are prefixed by `BSP_STRelay_`.

Detailed technical information about the APIs available to the user can be found in a compiled HTML file located inside the “Documentation” folder of the software package where all the functions and parameters are fully described.

3.6 Sample application description

A sample application using the X-NUCLEO-OUT02A1 expansion board with either NUCLEO-F401RE or NUCLEO-F334R8 boards is provided in the “Projects” directory. Ready to be built projects are available for multiple IDEs.

In this example, different patterns are applied to the 8-channel digital output. A pattern change is requested by a user button press.

At startup, the 8 channels of the digital ouput are all switched off. After every user button press:

1. the first channel switches on
2. the second channel switches on... and so on
3. the 8th channel switches on
4. all channels switch off
5. channel 1 remains off, channels 1 to 7 blink at a frequency of 1 Hz and a duty cycle of 50%, channel 8 is always on
6. all channels switch on
7. channels 1 to 4 blink at 0.2 Hz and with a respective duty cycle of 12%, 37%, 62%, 87%. Channel 2 to 8 blink at a frequency of 0.1 Hz and with a respective duty cycle of 12%, 37%, 62%, 87%.
8. all channels switch off

Then, if the user button is pressed again, the sequence restarts at step 1.

In the meantime, if the level of the FAULT or the PGOOD pins goes low, an interrupt is generated to call the FAULT or the PGOOD handlers where customized code can be executed.

3.7 Daisy chaining handling

The daisy chaining is natively supported by the X-CUBE-OUT2 software and the provided application can be easily modified to handle several boards.

To manage two boards, in the *main.c* file, you have to:

1. replace `BSP_STRelay_SetNbDevices(BSP_STRELAY_BOARD_ID_ISO8200AQ, 1);` with `BSP_STRelay_SetNbDevices(BSP_STRELAY_BOARD_ID_ISO8200AQ, 2);`
2. add the initialization of the second board with `BSP_STRelay_Init(BSP_STRELAY_BOARD_ID_ISO8200AQ, NULL);` (one call to `BSP_STRelay_Init` is required for each expansion board)
3. drive the digital output of the second ISO8200AQ with the same control functions used for the first one provided to change the device ID. For example: `BSP_STRelay_SetOneChannelStatus (0, 4, 1);` switches the first ISO8200AQ channel 4 on, whereas `BSP_STRelay_SetOneChannelStatus(1, 4, 1);` switches the second ISO8200AQ channel 4 on

The GPIO configuration of the second X-NUCLEO-OUT02A1 expansion board must be aligned with the configuration specified for device 1 in the *Drivers\BSP\X_NUCLEO_OUT02A1\x_nucleo_out02a1.h* file.

4 System setup guide

4.1 Hardware description

4.1.1 STM32 Nucleo platform

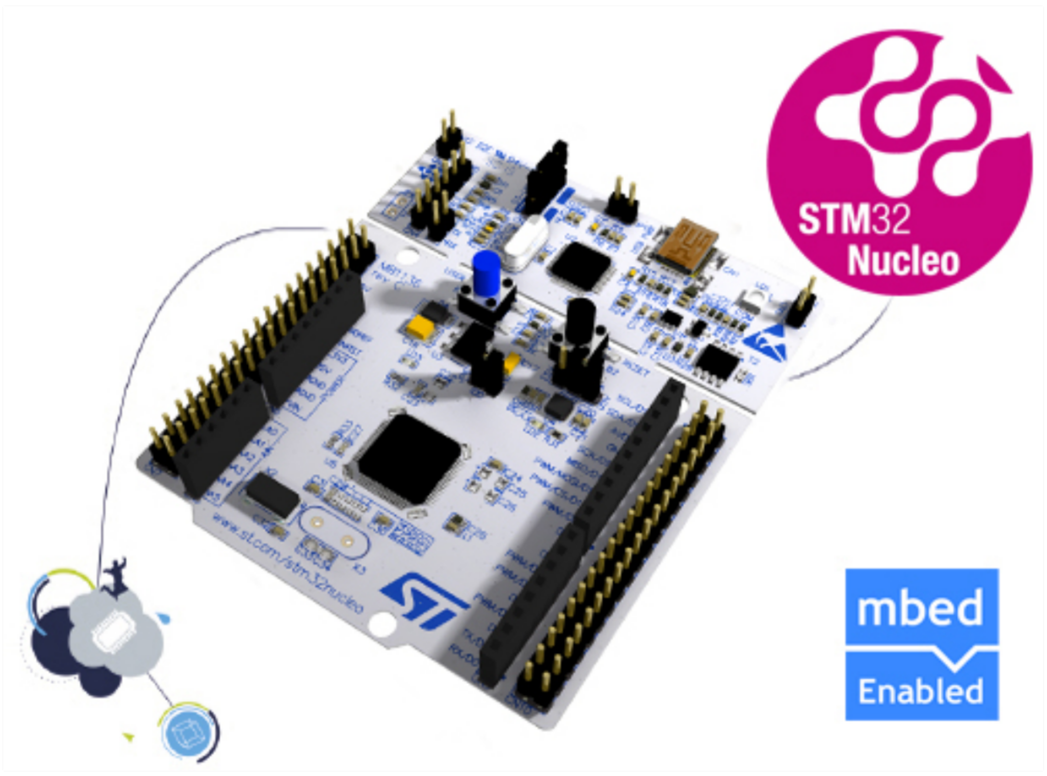
STM32 Nucleo development boards provide an affordable and flexible way for users to test solutions and build prototypes with any STM32 microcontroller line.

The Arduino™ connectivity support and ST morpho connectors make it easy to expand the functionality of the STM32 Nucleo open development platform with a wide range of specialized expansion boards to choose from.

The STM32 Nucleo board does not require separate probes as it integrates the ST-LINK/V2-1 debugger/ programmer.

The STM32 Nucleo board comes with the comprehensive STM32 software HAL library together with various packaged software examples.

Figure 4. STM32 Nucleo board



Information regarding the STM32 Nucleo board is available at www.st.com/stm32nucleo

4.1.2 X-NUCLEO-OUT02A1 expansion board

The X-NUCLEO-OUT02A1 industrial digital output expansion board for STM32 Nucleo is based on the ISO8200AQ galvanic isolated octal high-side smart power solid state-relay.

It provides an affordable and easy-to-use solution for the development of 8-channel digital output modules, letting you easily evaluate the ISO8200AQ communication and industrial load driving features.

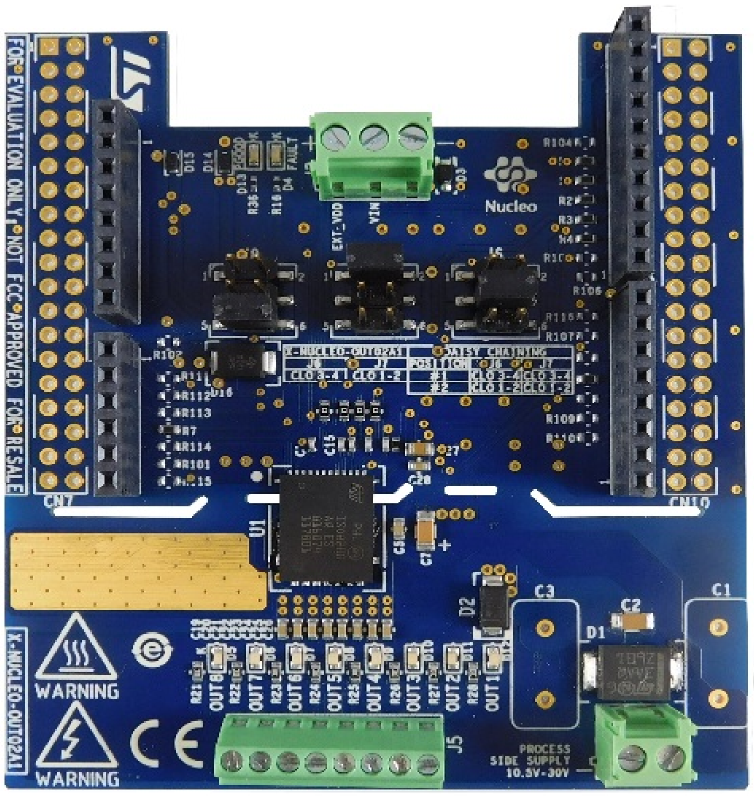
The X-NUCLEO-OUT02A1 can be connected to a NUCLEO-F401RE or NUCLEO-F334R8 development board via Arduino™ UNO R3 connectors.

You can also evaluate the 16-channel digital output modules by connecting two X-NUCLEO-OUT02A1 expansion boards and activating the daisy chaining feature.

The X-NUCLEO-OUT02A1 interfaces with the STM32 controller via SPI and GPIO pins and is compatible with the Arduino™ UNO R3 (default configuration) and ST morpho (optional, not mounted) connectors.

Industrial PLC functionality with 8 inputs and 16 outputs can be added with the X-NUCLEO-PLC01A1 expansion board.

Figure 5. **X-NUCLEO-OUT02A1 expansion board**



4.2 Hardware setup

The following hardware components are needed:

1. One STM32 Nucleo development platform (suggested order code: NUCLEO-F401RE or NUCLEO-F334R8)
2. One industrial digital output expansion board (order code: X-NUCLEO-OUT02A1)
3. One USB type A to Mini-B USB cable to connect the STM32 Nucleo to the PC
4. An external power supply (10.5 - 30 V) and the associated wires to supply the X-NUCLEO-OUT02A1 expansion board

4.3 Software setup

The following software components are required to set up a suitable development environment for creating applications for the STM32 Nucleo equipped with the X-NUCLEO-OUT02A1 expansion board:

- X-CUBE-OUT2: an expansion for STM32Cube dedicated to application development which require the use of the ISO8200AQ device. The X-CUBE-OUT2 firmware and related documentation is available on www.st.com.
- Development tool-chain and Compiler: the STM32Cube expansion software supports the three following environments:
  - IAR Embedded Workbench for ARM® (IAR-EWARM) toolchain + ST-LINK
  - RealView Microcontroller Development Kit (MDK-ARM-STM32) toolchain + ST-LINK

– System Workbench for STM32 (SW4STM32) + ST-LINK

4.4 System setup guide

This section describes how to setup different hardware parts before developing and executing an application on the STM32 Nucleo board with the GNSS expansion board.

4.4.1 Setup for one X-CUBE-OUT02A1 expansion board

The STM32 Nucleo has to be configured with the following jumpers position:

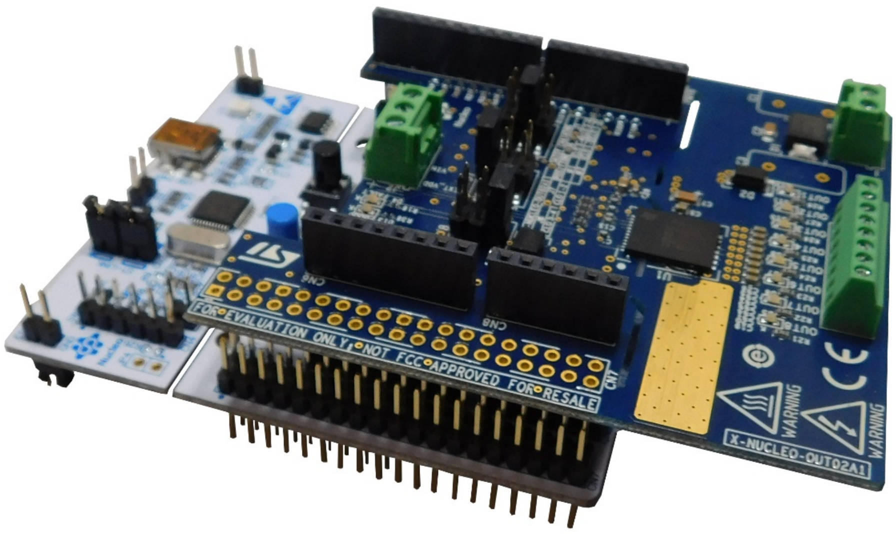
- JP1 off
- JP5 (PWR) on UV5 side
- JP6 (IDD) on

The X-NUCLEO-OUT02A1 expansion board configuration must be the following:

- jumper J7 and J8 closed on position 1-2
- jumper J6 closed on position 1-2

Step 1. Plug the X-NUCLEO-OUT02A1 expansion board on top of the STM32 Nucleo via the Arduino UNO connectors.

Figure 6. X-NUCLEO-OUT02A1 expansion board connected to an STM32 Nucleo development board



Step 2. Connect the STM32 Nucleo board to a PC with the USB cable through USB connector CN1 to power the board.

Step 3. Power the X-NUCLEO-OUT02A1 expansion board on by connecting J1 to the DC power supply (which must be set betwenn 10.5 and 30 V).

Step 4. Open your preferred toolchain (MDK-ARM from Keil, EWARM from IAR, or SW4STM32 from www.openstm32.org)

Step 5. Depending of the used STM32 Nucleo board, open the software project from:

– \stm32_cube\Projects\Multi\Examples\ PowerManagement\OUT02A1_ExampleWith1Driver \YourToolChainName\STM32F401RE-Nucleo for NUCLEO-F401RE

– \stm32_cube\Projects\Multi\Examples\ PowerManagement\OUT02A1_ExampleWith1Driver \YourToolChainName\ STM32F334R8-Nucleo for NUCLEO-F334R8

Step 6. To adapt the default parameters of the ISO8200AQ use:

– the `BSP_STRelay_Init` function with the NULL pointer and open the file: *stm32_cube\ Drivers\ BSP\ Components\ iso8200aq\ iso8200aq_target_config.h*

– or `BSP_STRelay_Init` function with the address of `initDevicesParameters` structure

Step 7. Rebuild all files and load your image into target memory.

Step 8. Run the example.

Each time the user button is pressed, a new pattern is applied to the digital output as described in Section 3.6 Sample application description.

Revision history

Table 3. **Document revision history**

Date	Version	Changes
05-Mar-2019	1	Initial release.

Contents

List of tables

List of figures