Enabling EtherCAT Distributed Clocks Mode using XMC4300/4800 MCU

About this document

Scope and purpose

This application note describes how to set up the EtherCAT Distributed Clocks functionality using XMC4300/4800 as slave devices and an industrial PC as the master device. The application demonstrates the Distributed Clocks mode of an EtherCAT slave node using the generated slave stack code for the XMC4300/4800 MCU in runtime with TwinCAT 3.

Intended audience

This application note is intended for anyone who wants to use XMC4300/4800 MCU for EtherCAT-related applications.

Associated part family

XMC4000 MCU family - XMC4300/4800 MCU series are the first-ever highly integrated Arm® Cortex®-M4-based microcontrollers with EtherCAT-integrated, bringing real-time Ethernet communication into an unrivaled level for ease of implementation and cost effectiveness. XMC4000 is the promising successor to traditional DSP/DSC and 32-bit MCU solutions. The XMC4000 family addresses industrial market needs and is ideal for digital power conversion, motor control, sense and control, and IO applications. It comes with a comprehensive set of common, fast, and precise analog/mixed signal, Timer/PWM, and communication peripherals.

Table of contents

About this document 1
Table of contents 2
1 Introduction 3
2 EtherCAT Distributed Clocks (DC) mode 4
3 DC mode setup with XMC4300/4800 6
3.1 Hardware setup 6
3.2 Software setup 6
4 Testing the DC mode in TwinCAT 3 14
References 22
Revision history 23
Disclaimer 24

1 Introduction

EtherCAT (Ethernet for Control Automation Technology) is an industrial Ethernet standard optimized for the high-performance demands of modern automation systems. Unlike traditional Ethernet, EtherCAT employs master-slave architecture with efficient "on-the-fly" data processing, enabling high-speed communication and precise synchronization. This mechanism significantly reduces latency, making EtherCAT ideal for real-time applications.

To address applications requiring sub-microsecond synchronization, EtherCAT introduces the Distributed Clocks (DC) mode, a hardware-based synchronization feature that ensures all slave devices maintain precise alignment with the master clock. DC mode minimizes jitters to the nanosecond range, making it essential for high-precision use cases, such as robotics, motion control, and data acquisition.

By leveraging the capabilities of DC mode, developers can achieve deterministic system behavior and highly accurate synchronization, both of which are essential for modern industrial automation. Enabling the use of Infineon's XMC4300 and XMC4800 microcontrollers as EtherCAT slave nodes offers an effective solution. These MCUs seamlessly support the implementation of EtherCAT DC mode, providing a reliable and cost-efficient way for achieving high-precision synchronization in industrial systems.

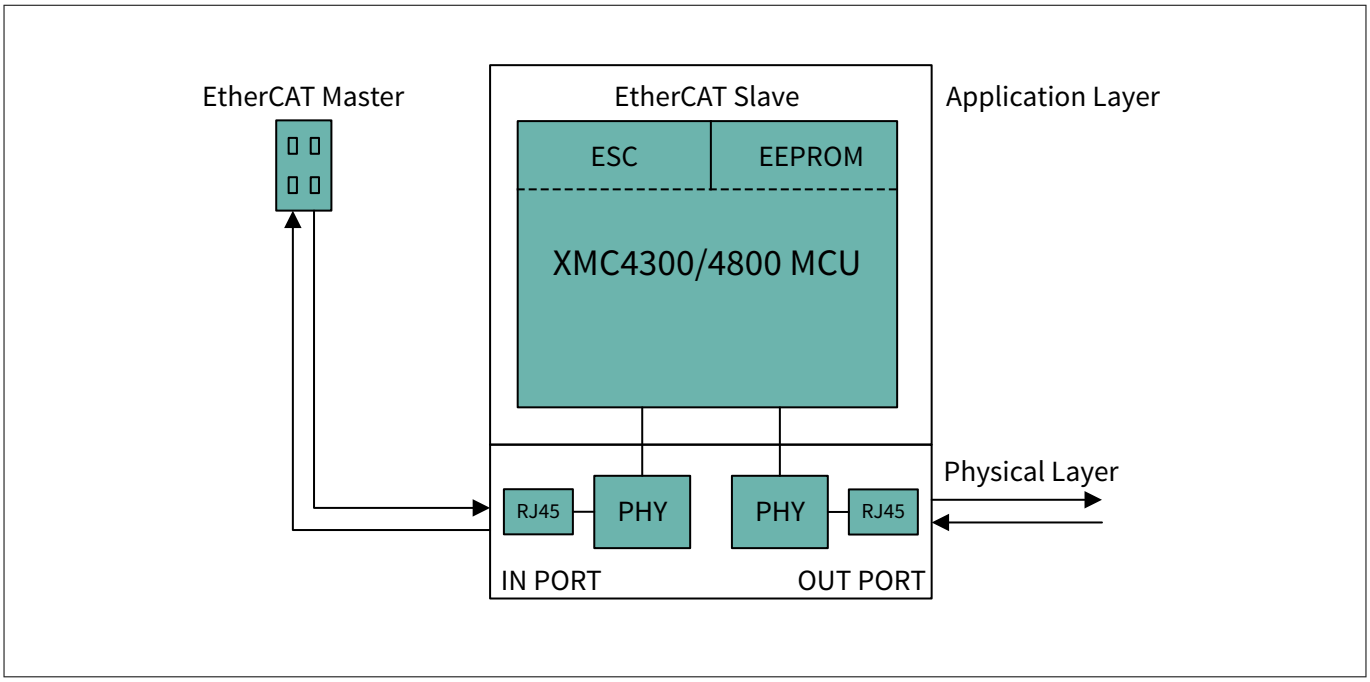


Figure 1 EtherCAT master/slave generic block diagram for XMC4300/4800

2 EtherCAT Distributed Clocks (DC) mode

In EtherCAT DC mode, the master assigns one slave as the Reference Clock, while all other slaves synchronize their local clocks to this reference. During initialization, the master measures propagation delays (the time it takes for frames to travel) and compensates for these offsets by issuing periodic synchronization updates. This continuous alignment eliminates clock drift caused by oscillator variations, ensuring that all devices operate on a unified global time.

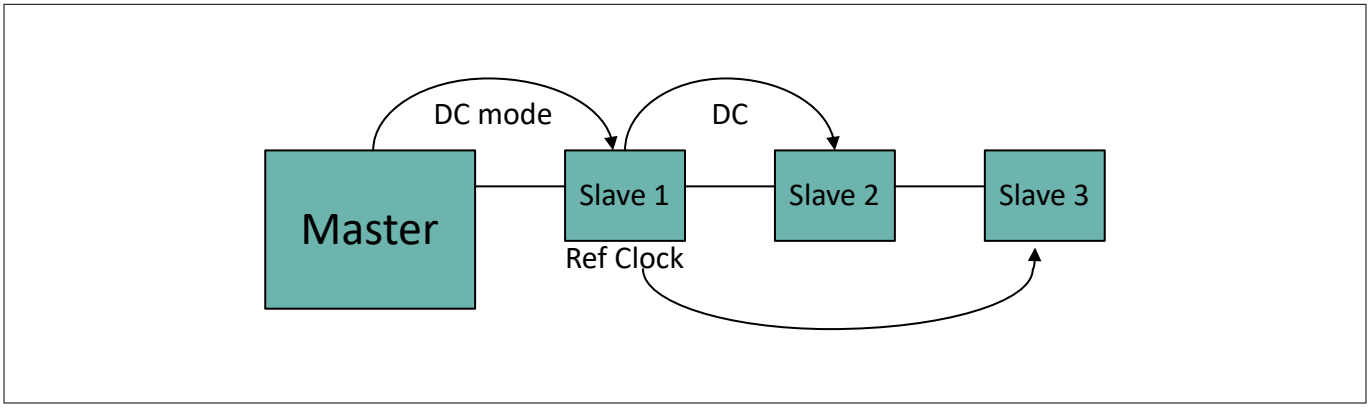


Figure 2 EtherCAT master/slave DC mode clocks

Synchronization relies on two events: Sync0 and Sync1. These events are periodic signals generated by each slave's Distributed Clock unit. These signals trigger time-critical tasks, such as sampling inputs or actuating outputs, at precise intervals defined by the cycle time (for example, 125 µs). In addition to Sync0, which aligns tasks across all slaves, the optional Sync1 provides additional timing flexibility for applications that require secondary synchronization events.

A crucial factor in synchronization is the global shift time, which determines when the Sync0 signal is triggered relative to other EtherCAT cycle activities. The global shift time must occur after all slaves have received and processed the EtherCAT frame, ensuring synchronized local clocks. This timing also ensures that there is sufficient time for outputs to be prepared and validated before the next EtherCAT frame is transmitted. By carefully adjusting the global shift time, EtherCAT ensures alignment at the nanosecond level for operations such as input latching and output preparation, enabling deterministic performance across the network.

Jitter, the variation in the timing of periodic events, such as Sync0, relative to their expected intervals, is another critical consideration. EtherCAT minimizes jitter through hardware-based synchronization and periodic clock corrections, aligning all slaves to the Reference Clock. Although some jitter may still occur due to factors including clock drift and network communication variations, EtherCAT achieves a sub-100 nanosecond jitter in DC mode, making it well suited for precision-critical applications.

The EtherCAT frame plays a crucial role in synchronization and data exchange. The frame itself carries both process data and synchronization information between the master and slaves. Frames are transmitted cyclically, with their timing tightly controlled by the master. As the frame propagates through the network, each slave processes its portion of the data while forwarding the rest, thereby minimizing latency. These frames also contain timing information, which each slave uses to align its local clock. This alignment compensates for propagation delays, ensuring synchronization even in large or complex networks. Furthermore, the efficient handling of EtherCAT frames enables the network to operate deterministically, even under tight cycle time constraints.

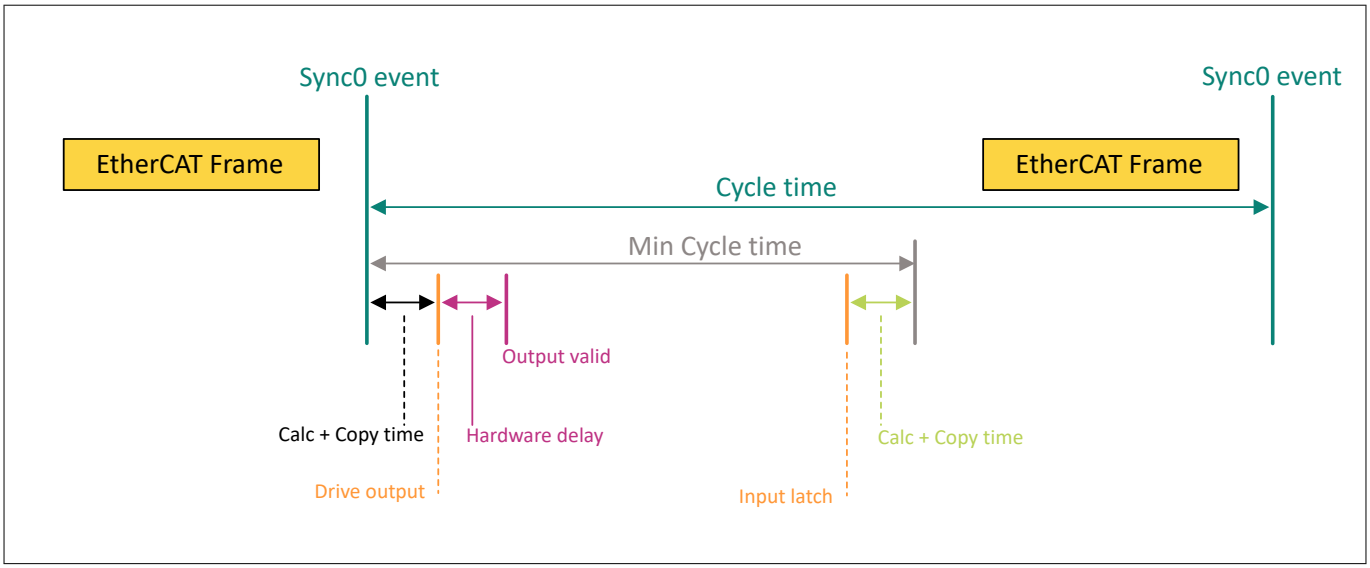


Figure 3 EtherCAT DC mode events and timings

The calculation copy time is the interval during which each slave processes the received EtherCAT frame, extracts its input data, and prepares its output data for the next cycle. This step ensures that the incoming data is handled correctly, and outputs are ready for the Sync0 or Sync1 signals. This deterministic processing avoids delays that could disrupt network synchronization.

The concept of output valid marks the exact moment when a slave's prepared output data becomes valid and ready to be acted upon by actuators or other connected devices. This event is triggered by the Sync0 or Sync1 signal, ensuring all slaves update their outputs simultaneously. By aligning output valid timing with synchronization signals, EtherCAT guarantees consistent and simultaneous output updates across the network, which is important for time-critical tasks such as motion coordination.

Input latching ensures that data from sensors or other inputs is captured at the same moment across all slave devices. This synchronization is triggered by Sync0 or Sync1, causing the devices to latch their inputs simultaneously. The data is then stored in their internal process data image for use in the current or subsequent cycles. This precise synchronization is essential for maintaining consistent input data across the entire network, a requirement that is critical for real-time data acquisition and control systems relying on timing accuracy.

Finally, propagation delay, the time taken for an EtherCAT frame to travel through cables and slaves, is another critical consideration. As the frame propagates through the network, each slave introduces a minimal delay due to its on-the-fly processing. During network initialization, the master measures these delays and compensates for them to ensure precise synchronization. The compensation for these delays is particularly important in large networks or systems with long cables, as unaccounted delays could lead to significant timing misalignments.

By tightly integrating concepts, such as global shift time, jitter reduction, propagation delay compensation, and deterministic handling of inputs and outputs, EtherCAT achieves precise synchronization across the network. This high level of precision and reliability ensures that EtherCAT can meet the demands of modern industrial applications, from motion control to advanced data acquisition and automation. For more information on the DC mode, refer to EtherCAT Distributed Clocks - default settings webpage of Beckhoff Information system.

3 DC mode setup with XMC4300/4800

The primary objective of this application note is to facilitate the implementation of DC mode for the EtherCAT protocol, utilizing the XMC4300/4800 as slave nodes in conjunction with an Industrial PC (CX2033) as the master node. For runtime testing of DC mode, TwinCAT 3 is employed.

3.1 Hardware setup

In an industrial network setup using EtherCAT, a typical configuration includes a master device (PLC or industrial PC), one or multiple slave devices, and the necessary connectors to establish communication between the devices. Specifically, this application note employs the following hardware components:

- An industrial PC (Beckhoff CX2033 as master)
- An XMC4300 and XMC4800 (as slaves)
- A PC (for diagnostics/testing)
- Connectors (LAN and micro-USB)

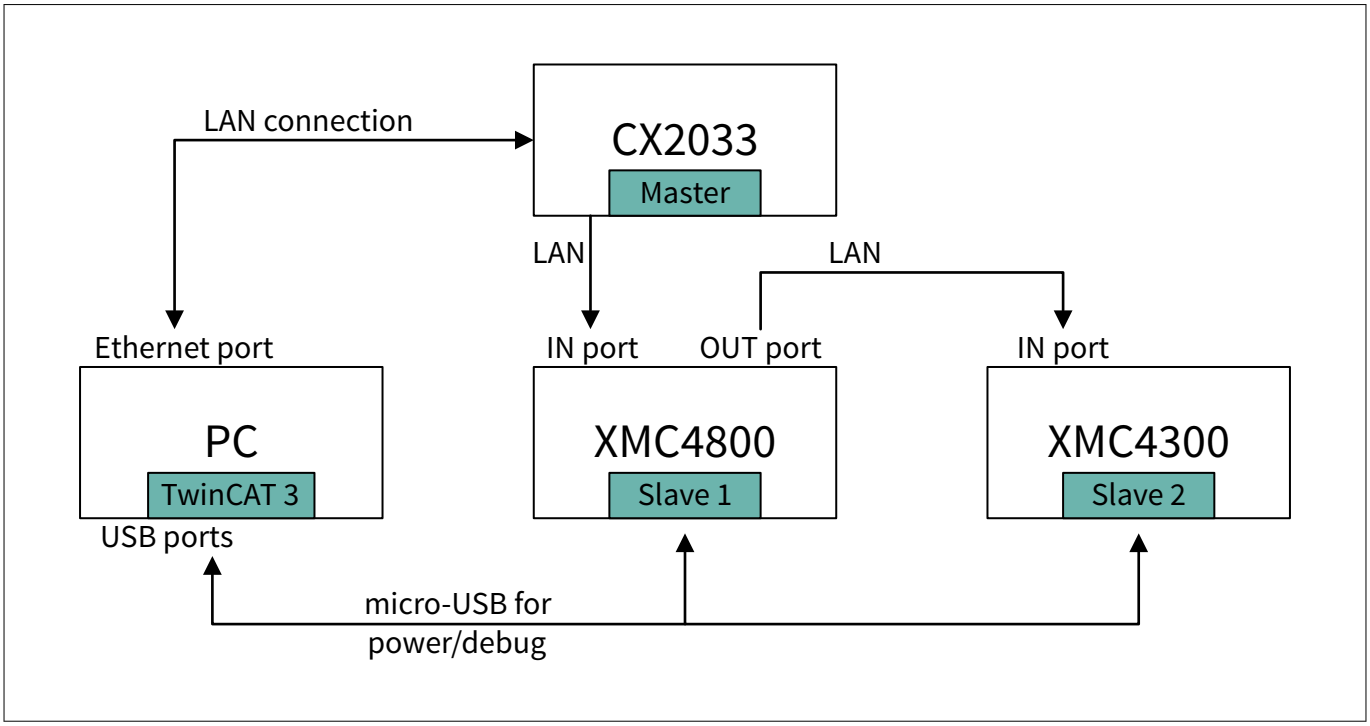


Figure 4 Hardware setup to run EtherCAT DC mode

Connect the master device to your local PC's Ethernet port to prepare it for the runtime. Connect the slave device(s) XMC4300/4800 to the master's Ethernet port with LAN connection, and power up the master device. XMC™ devices can be supplied from the micro-USB connection directly via X101 debug connector.

3.2 Software setup

The following software tools are required for software setup:

- EtherCAT Slave Stack Code (SSC) incl. OD-Tool v5.13
- TwinCAT 3
- ModusToolbox™ IDE v3.5 or later
- SEGGER JLink

To set up and run the software, do the following:

1. Generate the EtherCAT slave stack by following the steps outlined in the "Create the project" and "Operation" section in the README file of the XMC4300 EtherCAT SSC code example. If you are using an XMC4800 device, refer to the XMC4800 EtherCAT SSC code example
2. After generating the stack and making any necessary changes to the project, perform the following steps to flash the code in the respective XMC™ device. For more information, refer to the "Building and running the project" section in the README file
   a. Use SEGGER J-Flash Lite to clean the flash of XMC4300, which is located in **Program Files** > **SEGGER** > **JLink** > **JFlashLite**. Open the tool, select the target device as **XMC4300-256**, and then click **Erase Chip** as shown in Figure 5

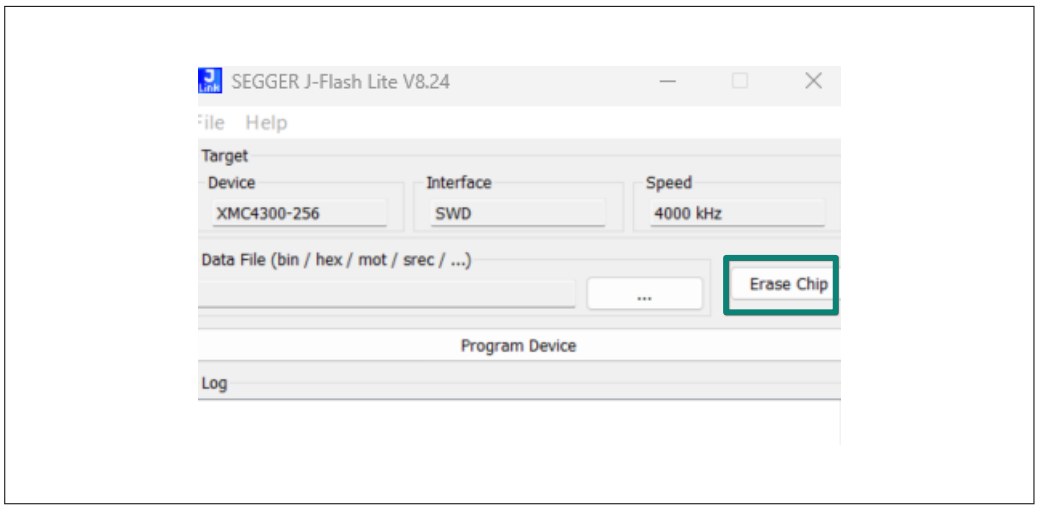


Figure 5 Flashing XMC4300

   b. Build the project. If there are any library-related issues, run the `make getlibs` command from the ModusToolbox™ IDE's terminal window, retry building, and then flash the code. If the previous steps are correct, the Ethernet IN-port should be flashing

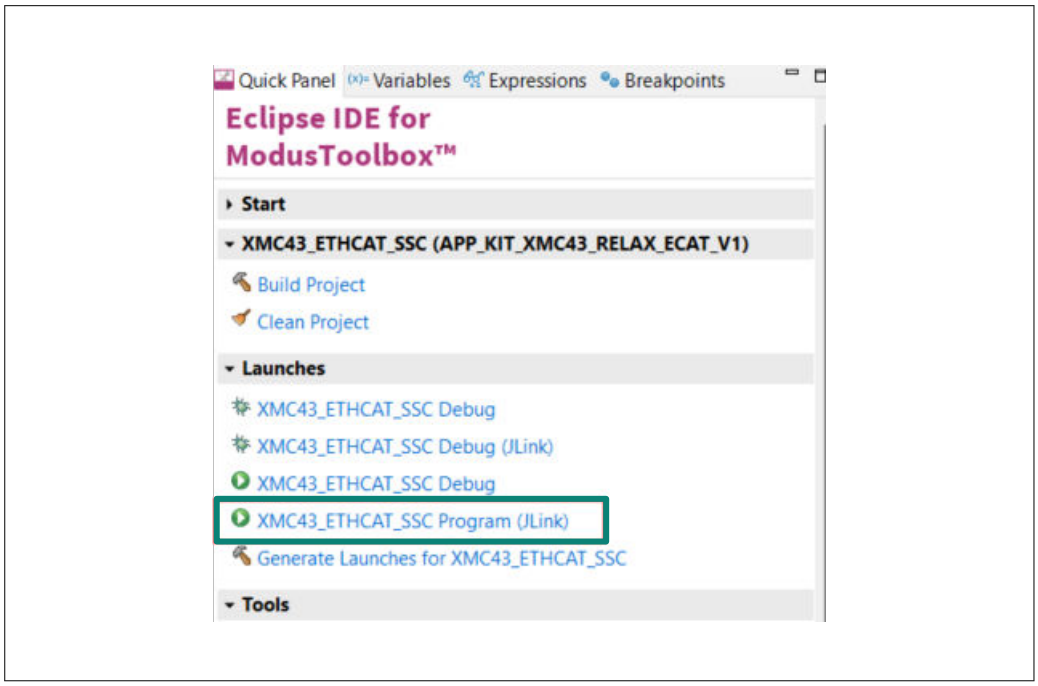


Figure 6 Build the project

The next steps regarding TwinCAT-related project creation and configuration, as outlined in the README file, can be skipped for now as they will be covered in detail in subsequent sections of this document

3. In the code example project, the Sync0 and Sync1 signals originating from the EtherCAT peripheral are used to trigger interrupts. The Sync0 signal is called cyclically to synchronize tasks among slaves in DC mode. The cycle time value can be configured using TwinCAT tasks, as will be explained in the following chapter.

In the main file of the code example, interrupt handler functions are used to call the Sync0 and Sync1 interrupt routines in DC mode as shown below

```
/*
* @brief SYNC0IRQHandler() - EtherCAT Interuupt Routine for SYNC0
*
* <b>Details of function</b><br>
* This routine is handling the SYNC0 Intrrupts and need to call the SSC Stack
*/
Void SYNC0IRQHandler(void)
{
     Sync0_Isr();
}

/*
* @brief SYNC1IRQHandler() - EtherCAT Interuupt Routine for SYNC1
*
* <b>Details of function</b><br>
* This routine is handling the SYNC1 Intrrupts and need to call the SSC Stack
*/

Void SYNC1IRQHandler(void)
{
     Sync1_Isr();
}
```

4. To enable or disable the user specific EtherCAT DC mode interrupts, the following functions are used in the main file

```
/**
* @brief mtb_xmc_enable_user_int() - Enabling of user specific EtherCAT Interrupt Routines
*
* <b>Details of function</b><br>
* This routine is called from ECAT_APP on request of SSC stack once interrupts (sync1/
sync0) need to be enabled
*/
Void mtb_xmc_enable_user_int()
{
    NVIC_EnableIRQ(ECAT_ERU_SYNC0_IRQN);
    NVIC_EnableIRQ(ECAT_ERU_SYNC1_IRQN);
}
/**
* @brief mtb_xmc_disable_user_int() - Disabling of user specific EtherCAT Interrupt
Routines
*
* <b>Details of function</b><br>
* This routine is called from ECAT_APP on request of SSC stack once interrupts (sync1/
sync0) need to be disabled
*/
{
    NVIC_DisableIRQ(ECAT_ERU_SYNC0_IRQN);
    NVIC_DisableIRQ(ECAT_ERU_SYNC1_IRQN);
}
```

5. After the EtherCAT code example project is successfully imported into your local workspace, open the TwinCAT XAE application to set up the TwinCAT 3 system to prepare the DC mode for testing by creating a new TwinCAT project

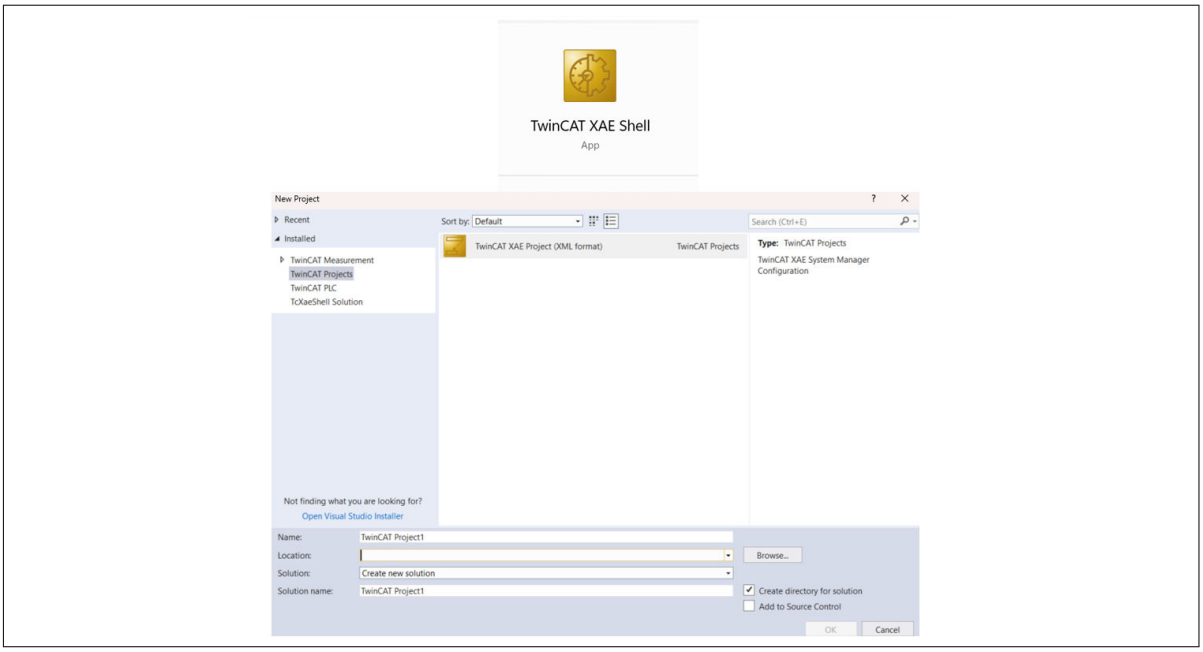


Figure 7 TwinCAT XAE Shell to create a new project for testing

6. Once the project has been created, add the generated slave device configuration files to the TwinCAT application. According to the instructions provided in the README file of the XMC™ EtherCAT code example, the slave device's generated XML file (`XMC_ESC.xml`) must be placed inside the following directory: `C:\TwinCAT\3.1\Config\Io\EtherCAT`. The generated XML file can be found in your local ModusToolbox™ workspace, typically located in the project folder under `imports–mtb-xmc-ecat–ssc_config`)
7. In the TwinCAT project, master/slave devices should be scanned or manually added. To do so, verify that the master device is properly powered up, connected to the PC, and correctly linked to the input/output ports of the XMC4300/4800 devices as explained in the Hardware setup
8. From the project window, select **SYSTEM**. Then, click **Choose Target** to search (Ethernet) options for the master device as shown in Figure 8

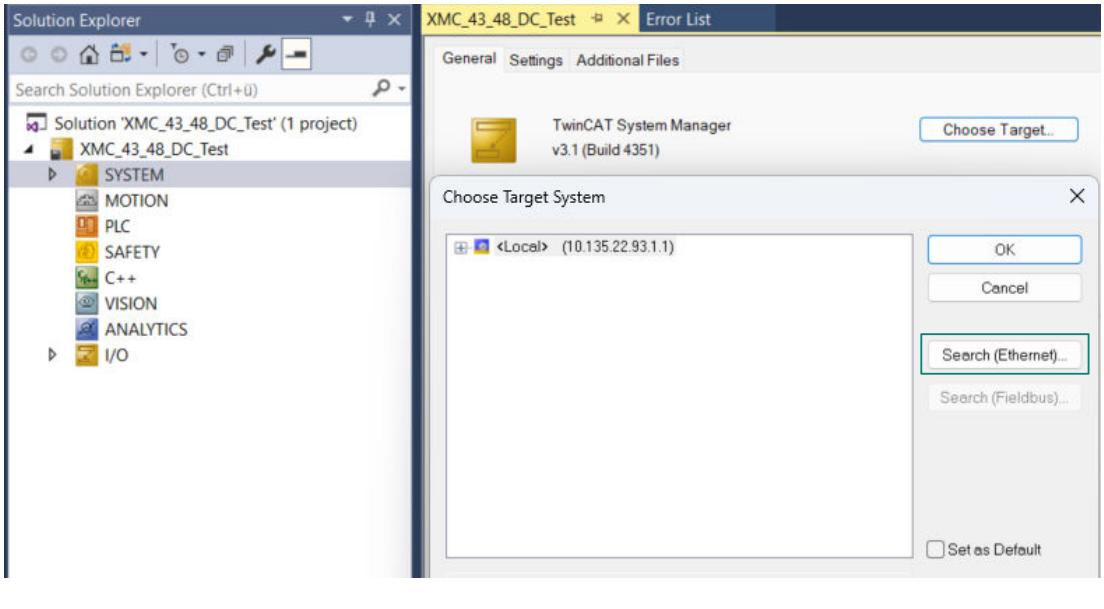


Figure 8 TwinCAT system search for master device

To add the target device, follow these steps:

a. From **Choose Target system** window, click **Search (Ethernet)...**.

Add Route Dialog box pops up

b. In **Add Route Dialog** window, enter the host name or IP information and click **Broadcast Search**. As a result, the host information is displayed in the window as shown in Figure 9

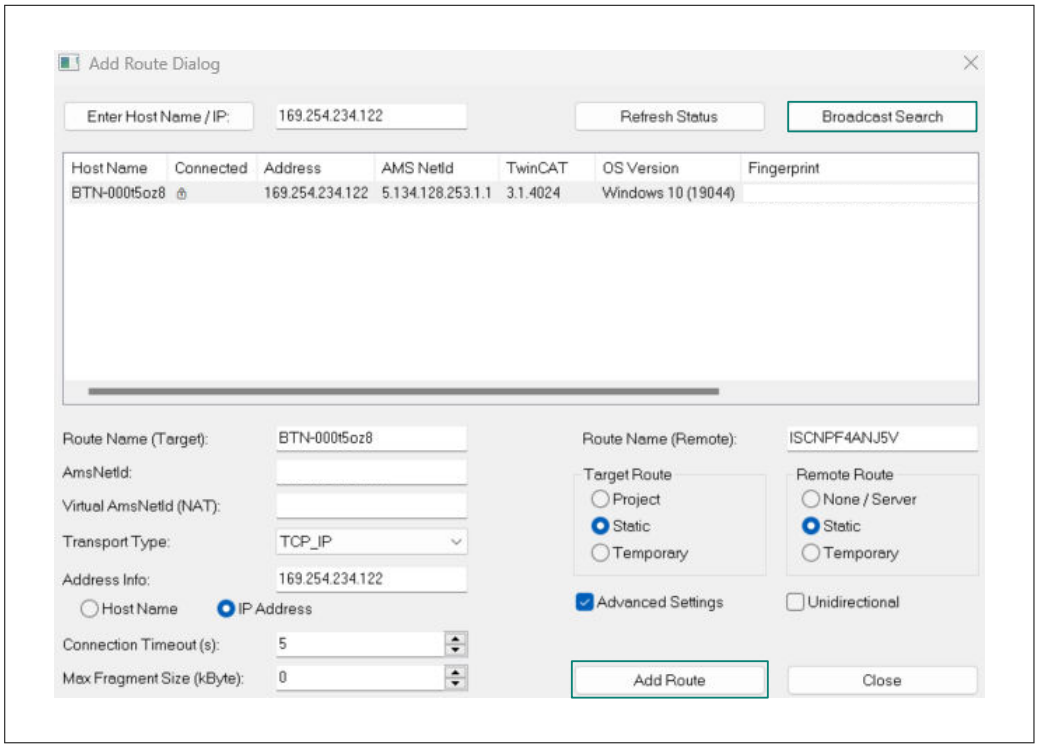


Figure 9 Adding the master device to the TwinCAT setup

c. Click the **Add Route** button. Enter the username and password provided by the master device vendor

9. After successfully adding the target device to the system, navigate to **SYSTEM** > **Choose Target** in the project window, you should then be able to view the option for target device, at which point select it and click **OK**

The target device name should now be visible, with a lock logo in the upper menu, adjacent to the project name. This indicates that a successful connection is established between the master device (CX2033) and TwinCAT 3

10. To scan the available EtherCAT ports of the master device, select **I/O** in the project window and then right click on “Devices” to initiate the scanning process, as shown in Figure 10

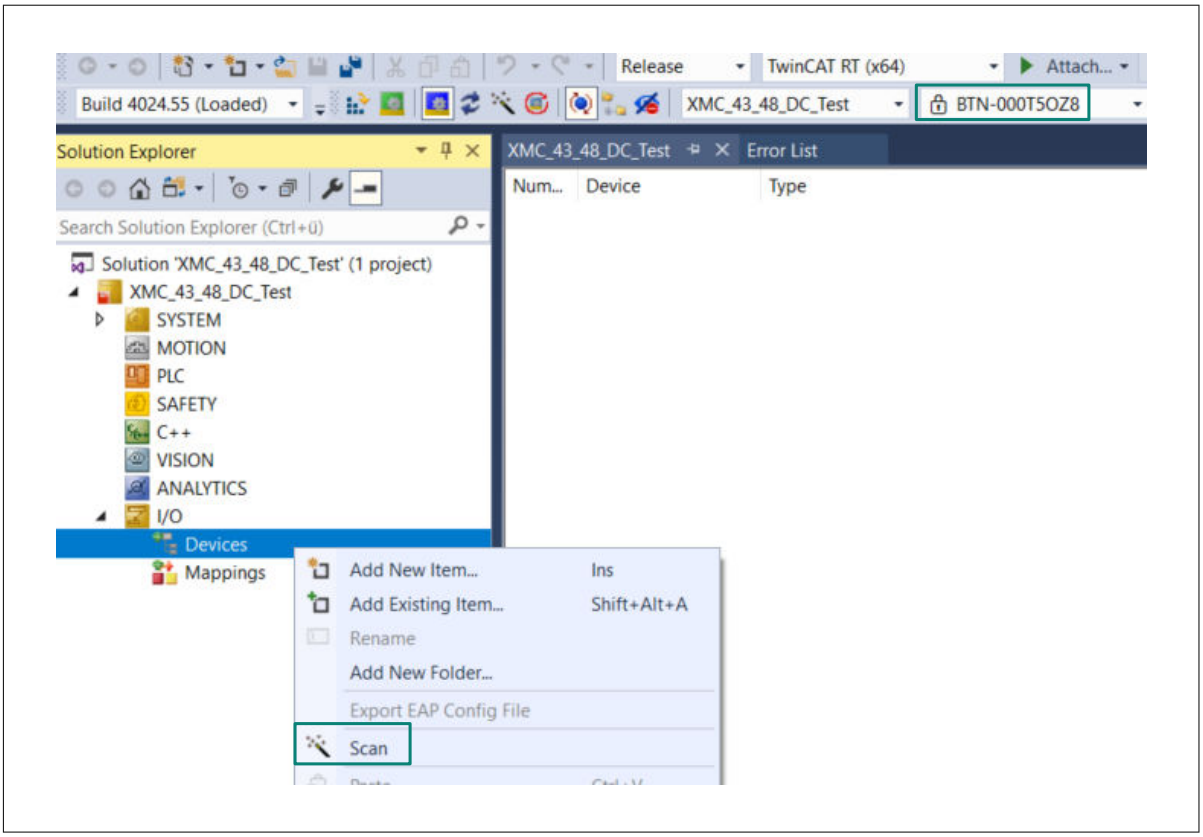


Figure 10 Adding the master device to the TwinCAT project

11. In the scan window, select the correct connection option that connects the master to the slave and click **OK**.

For example, for CX2033 master device, the connection option is EtherCAT port "X001", as shown in Figure 11

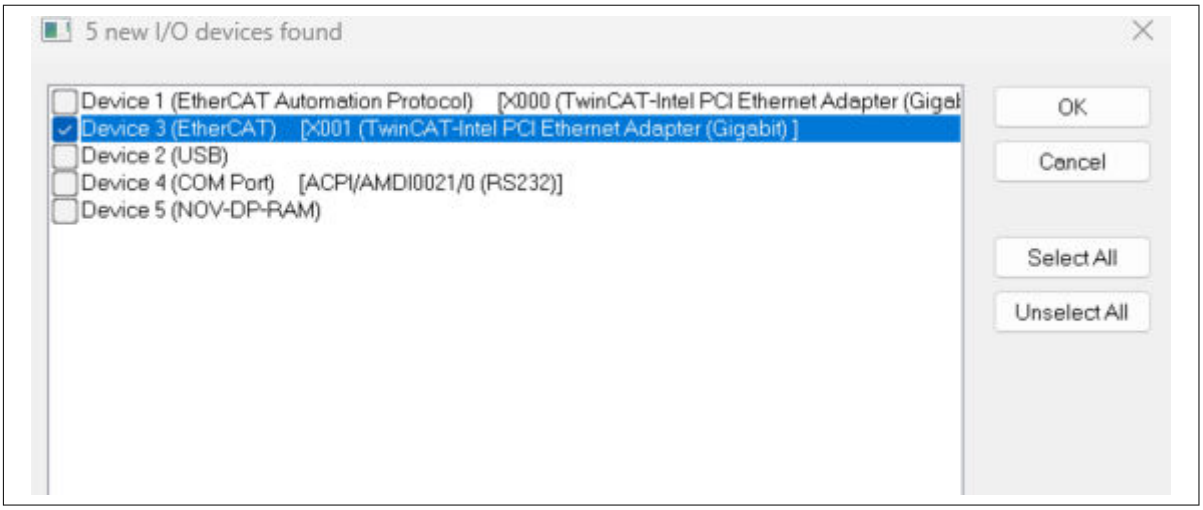


Figure 11 Scanning the connected I/O device(s)

12. The program will automatically prompt to scan for boxes to find the slave device. Select **Yes** and the XMC™ device should be added to the project window as “Box 1 (XMC_ESC)”. If multiple XMC™ devices are used, the program will scan and identify them sequentially as Box 1, Box 2, and so on.

Ensure sure that the input/output data of the slave device is accurately shown in the project window, as defined in the code example

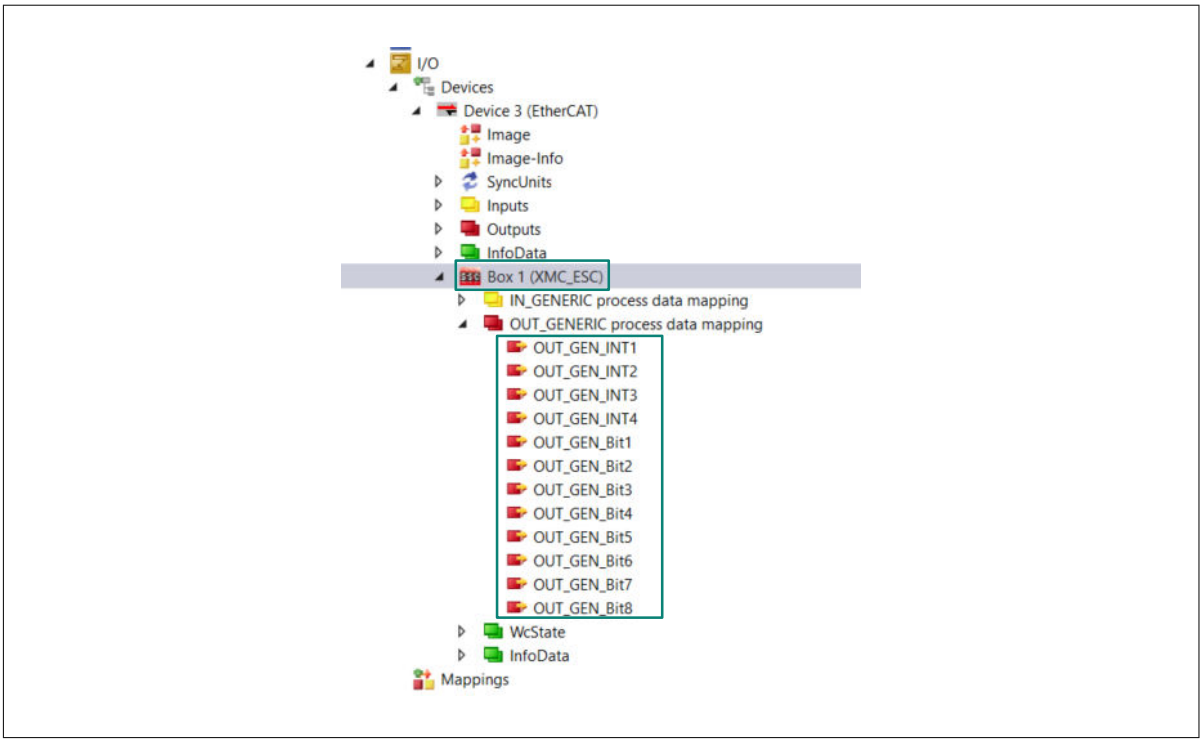


Figure 12 Checking the scanned slave device's input/output process data information

After setting up the TwinCAT environment with the connected master and slave devices, a testing phase is required to activate the DC mode and verify the results.

4 Testing the DC mode in TwinCAT 3

Follow these steps to test the DC mode in TwinCAT 3:

1. Run the TwinCAT application in normal Sync mode to ensure that the devices are in operating mode. After the successful scanning of the master and slave devices in the TwinCAT project, click **Toggle Free Run State** logo to activate the runtime, as shown Figure 13

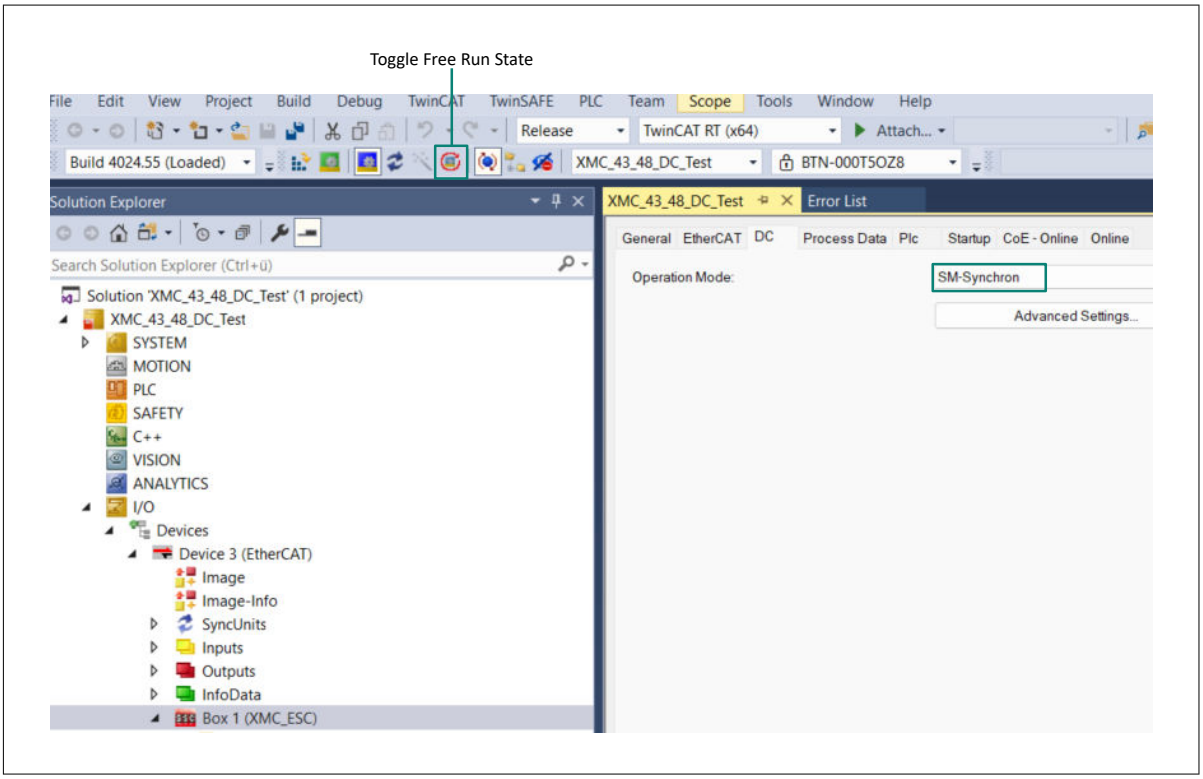


Figure 13 Toggle Free Run to activate the runtime in TwinCAT 3

2. In the **Solution explorer** window, navigate to **I/O** > **Devices**. Within the **Devices** folder,

a. Select **Box 1** (or **Box 2** for multiple slaves).

On the right pane, verify that the **DC** window displays the slave device in **SM-Synchron** mode by default, as shown in Figure 13

b. Select the master device, specifically the EtherCAT device.

On the right pane, verify that the **Online** window displays the device states as **OP** after toggling the free run mode, as shown in Figure 14

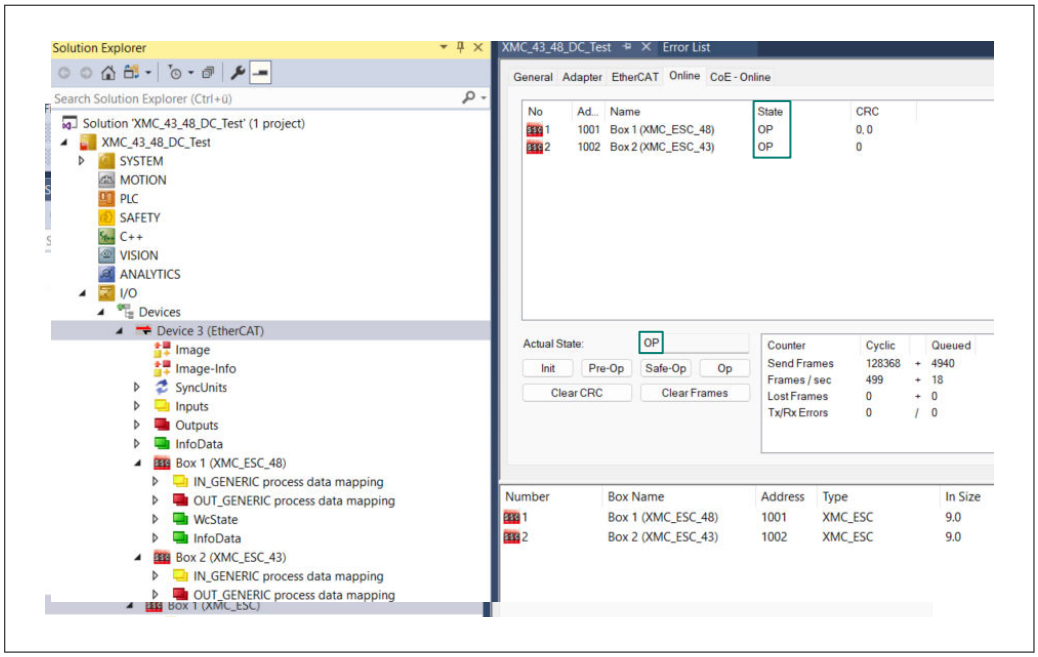


Figure 14 Master/slave device(s) running in OP state

Go to **EtherCAT** window and check that, in this setup, the SM-Synchron mode has a default cycle time of 4 ms

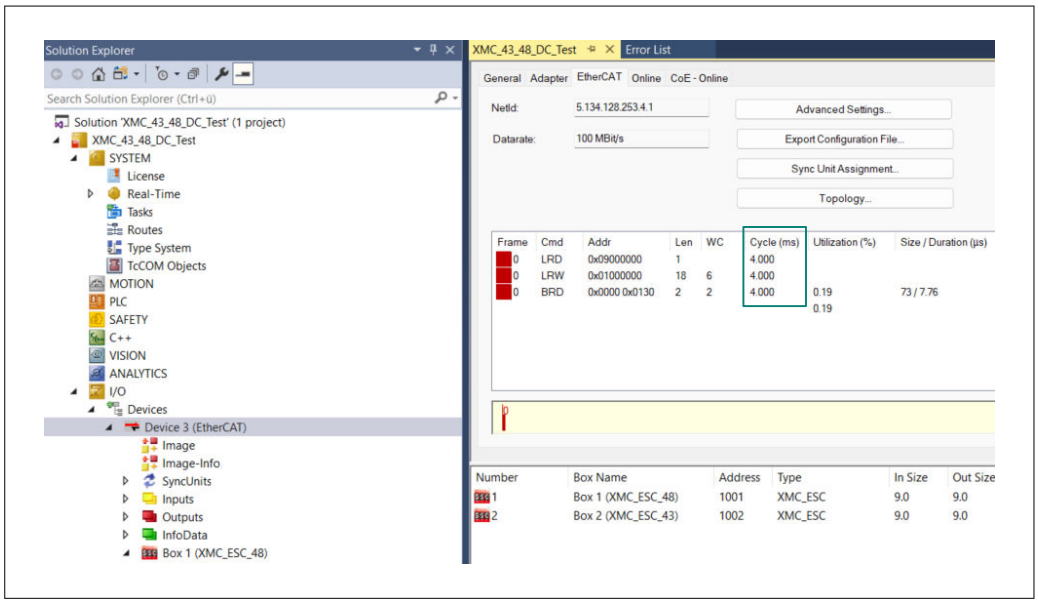


Figure 15 SM-Synchron mode default cycle time

3. Do the following configurations to enable and test the DC mode:

a. In the **Solution explorer** window, navigate to **"Your project name"** > **SYSTEM** > **Real-Time**. On the right pane, adjust the **Base Time** to 125 µs, as shown in Figure 16

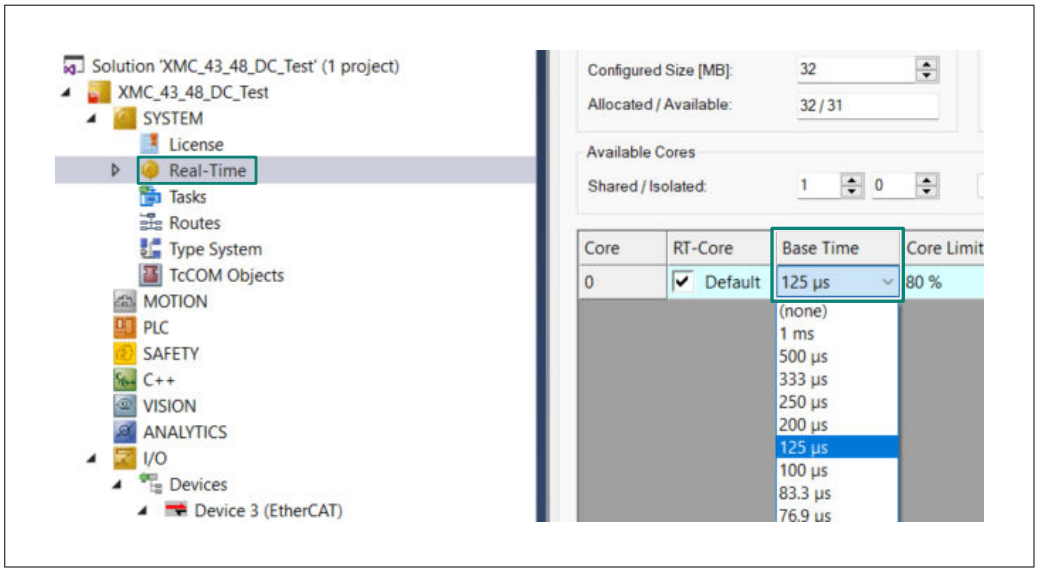


Figure 16 Adjusting the system base time for DC mode

b. In the **Solution explorer** window, navigate to **"Your project name"** > **SYSTEM** > **Tasks**.

Right click on the **Tasks**, and select **Add new item**.

The **Insert Task** window pops up. In this window, select **TwinCAT Task with Image** option, as shown in Figure 17.

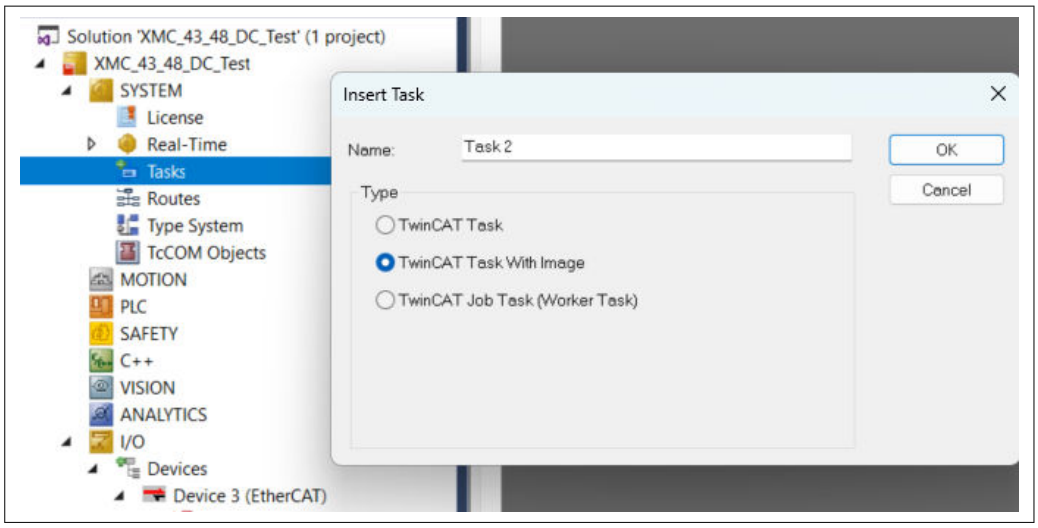


Figure 17 Creating TwinCAT task with Image

Select the new task, which is created under the **Tasks**. On the right pane, in the **Task** window, adjust the **Cycle ticks** according to the desired DC mode cycle time.

For example, in Figure 18, **Cycle ticks** is set to 2, which sets the DC mode cycle time to 250 µs. The lowest supported XMC4300/4800 DC mode cycle time is 125 µs

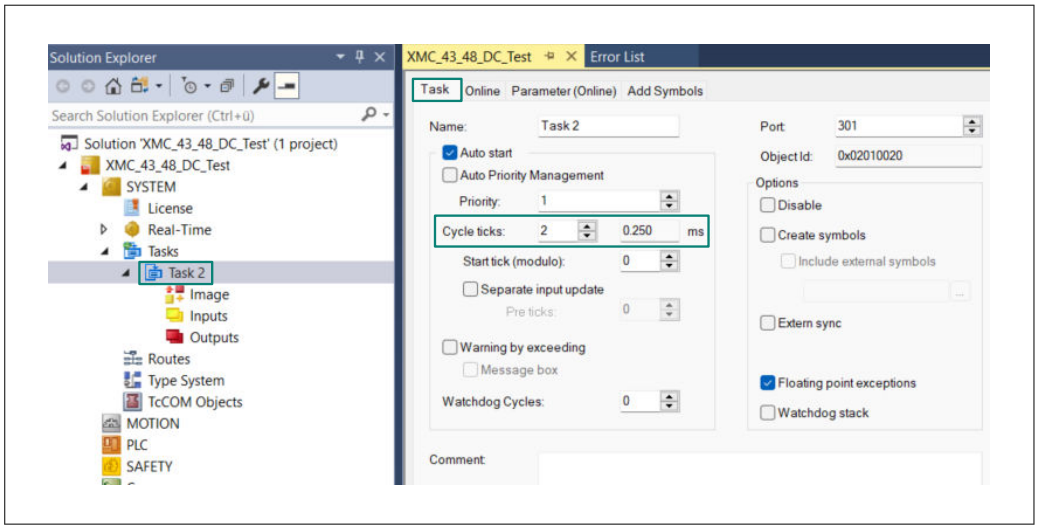


Figure 18 Setting the cycle ticks and cycle time for DC mode

c. Under the created task, right click on **Inputs** and select **Add new item**.

The **Insert Variable** window pops up. In this window, select the required **Data Type**(for example, BIT, as shown in Figure 19. Then, click **OK**

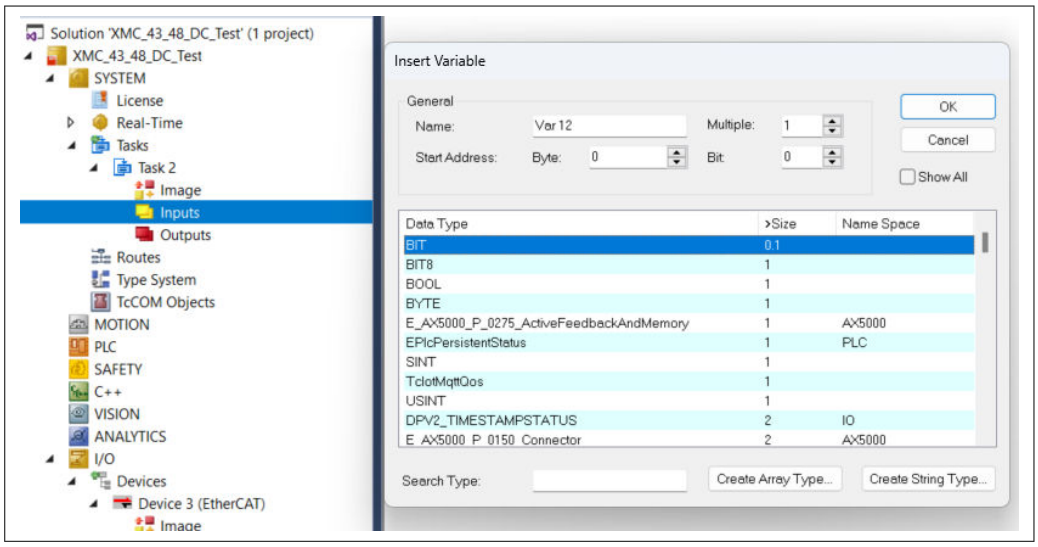


Figure 19 Creating the Task variable input

In the project window, select new variable, which created under **Inputs**. On the right pane, in the **Variable** window, select **Linked to** option.

In the **Attach Variable Var 12 (input)** popped up window, choose **IN_GEN_Bit1** (or another input bit) from Box 1, as shown in Figure 20

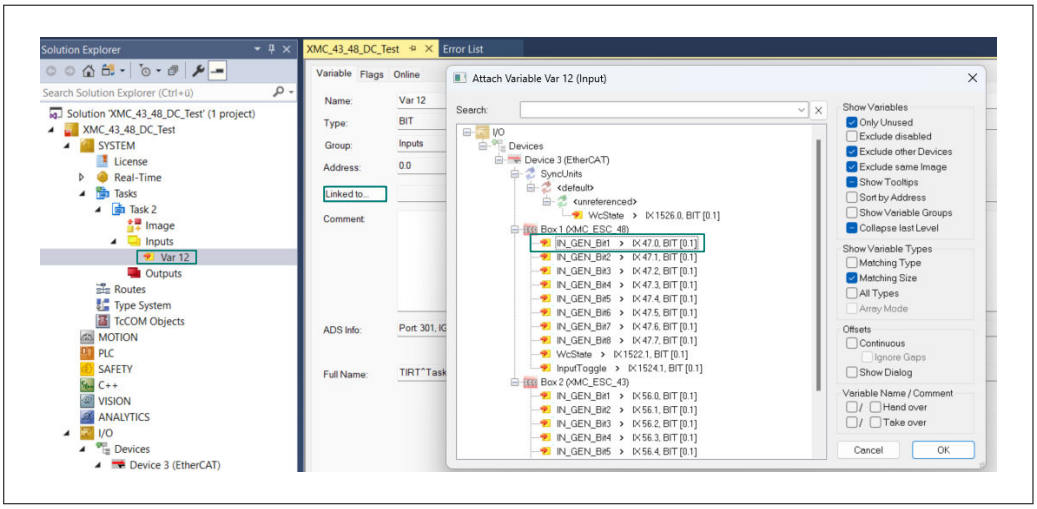


Figure 20 Linking the created task variable to slave input

d. If multiple slave devices are connected, repeat the previous steps for creating another input variable and link it to **IN_GEN_Bit1** (or another input bit) from Box 2, as shown in Figure 21

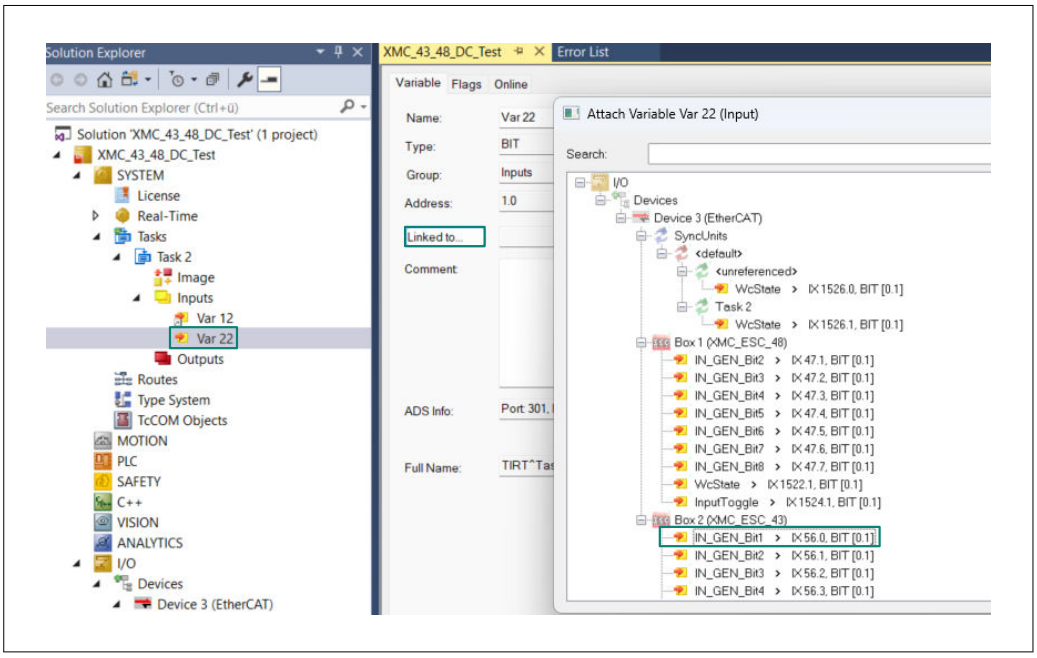


Figure 21 Linking the created task variable to the second slave

4. In the **Solution explorer** window, navigate to **I/O** > **Devices** > **Device folder**. Within the device folder,

a. Select **Box 1**.

On the right pane, in the **DC** window, change the opration mode from **SM-Synchron** to **DC-Synchron**, as shown in Figure 22

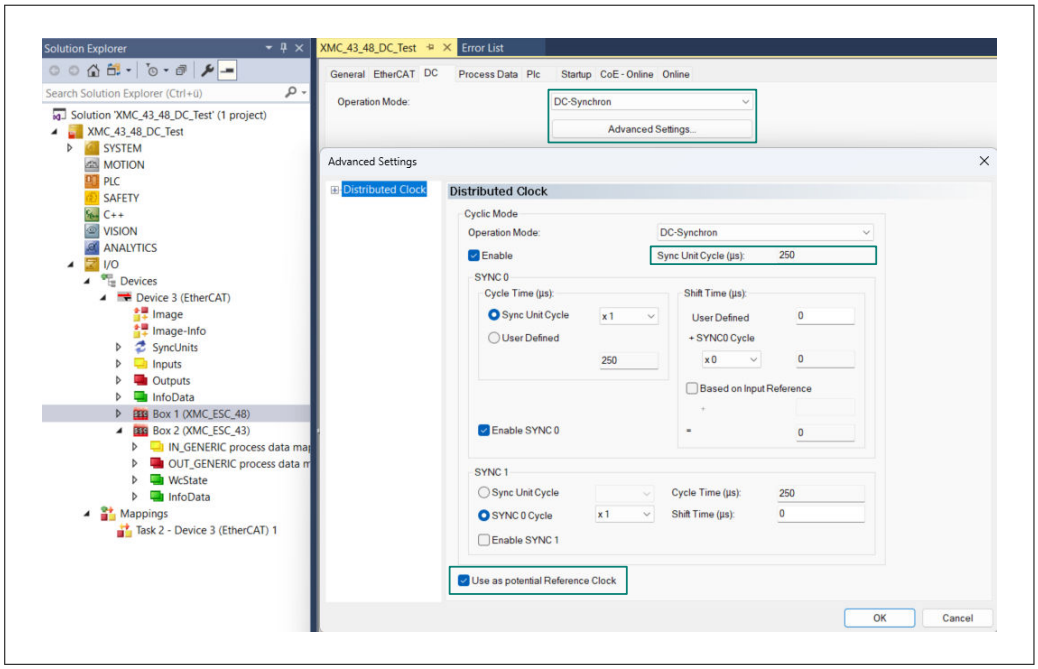


Figure 22 Change the slave configuration for DC mode

b. If multiple slave devices are connected, do the same change for each slave. From the first slave connected to the master, go to **Advanced Settings** option in the **DC** window, and set the slave as a reference clock, as shown in Figure 22

c. No additional manual settings are required, as all other settings for DC mode, including Sync 0 cycle time and shift time, will be automatically adjusted according to the created task

5. After creating the task variable and selecting the DC mode, all updated settings must be activated by clicking the **Activate Configuration** button. Click this button and then select **OK** from the subsequent window that appears, as shown in Figure 23

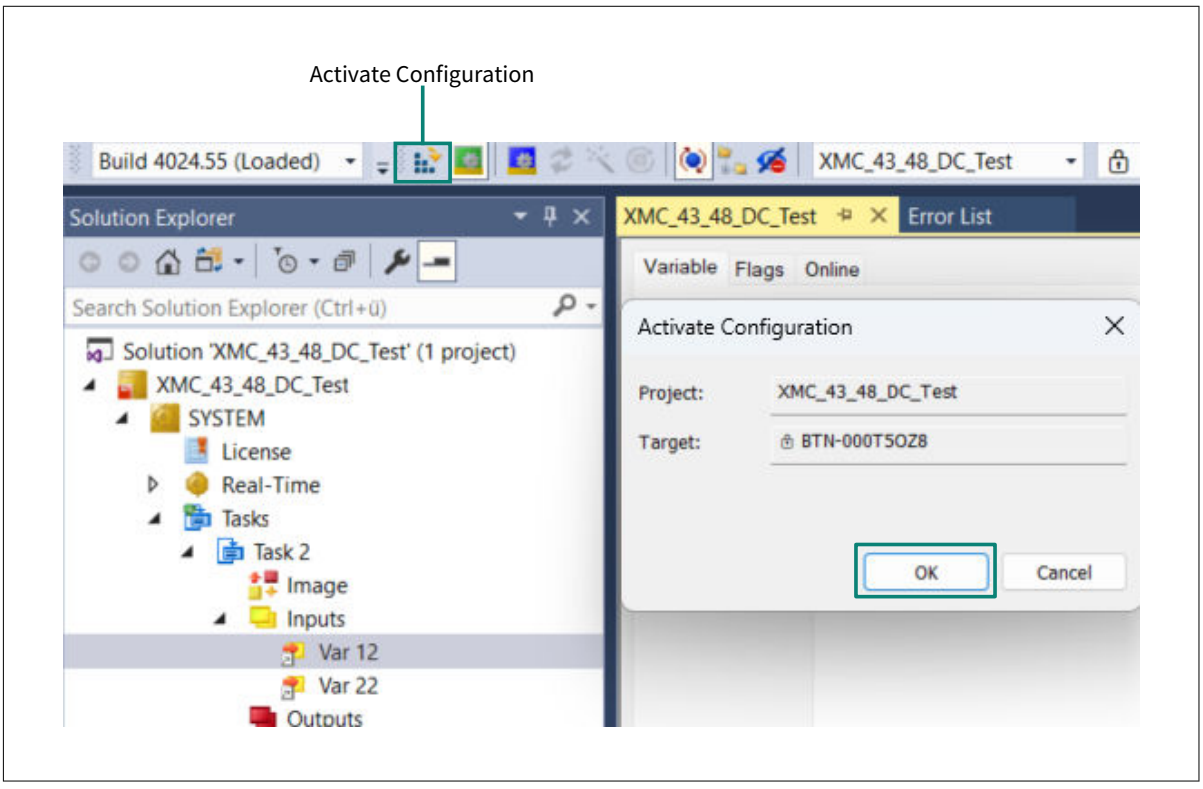


Figure 23 Activating the DC mode configurations

6. If the TwinCAT program prompts for a system restart in Run Mode, click **OK** to proceed. Following the configuration update, the DC mode should be activated, which can be verified from the master device's EtherCAT window, as shown in Figure 24

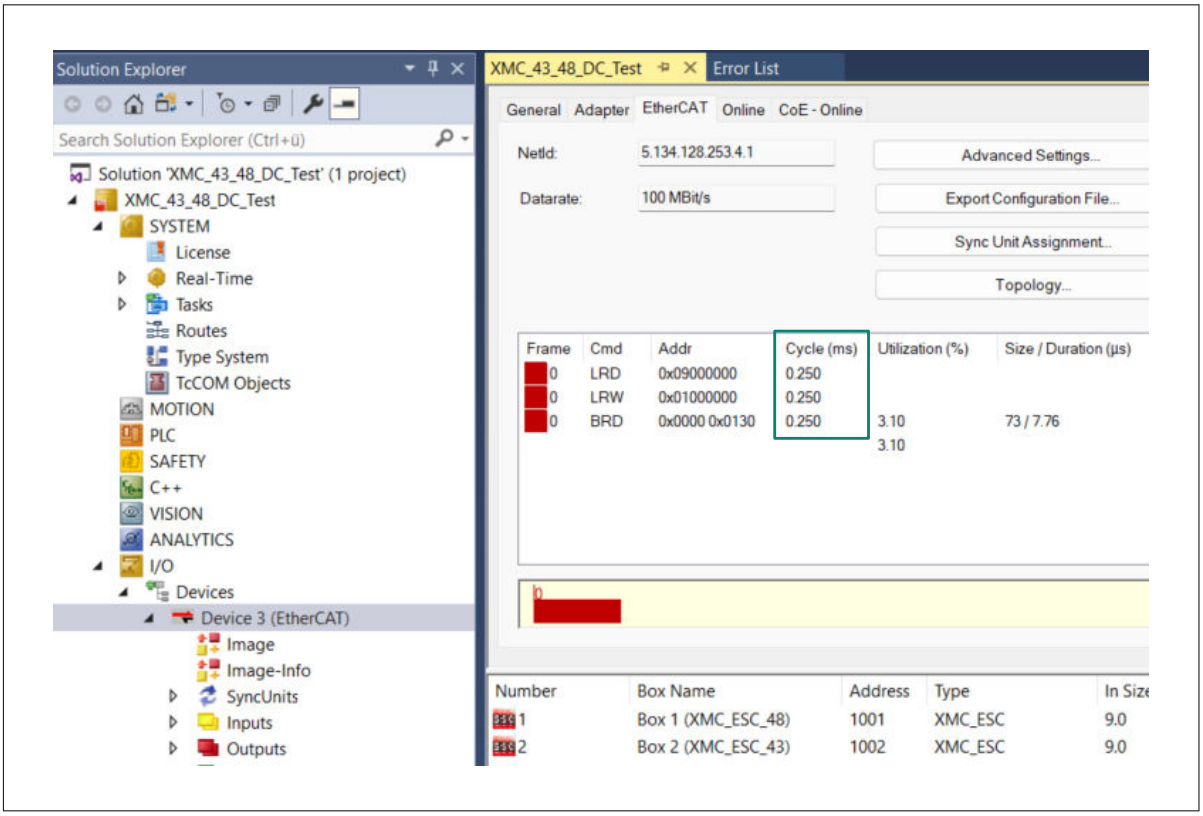


Figure 24 Verifying the DC mode cycle time

7. In DC mode, the process input/output data of the connected slave device can also be tested. The BUTTON1 and BUTTON2 on the XMC4800 Relax EtherCAT Kit are mapped to the IN_GEN_Bit1 and IN_GEN_Bit2 objects, respectively. Pressing and holding either BUTTON1 or BUTTON2 updates the corresponding button states on the host, which can be observed Figure 25.

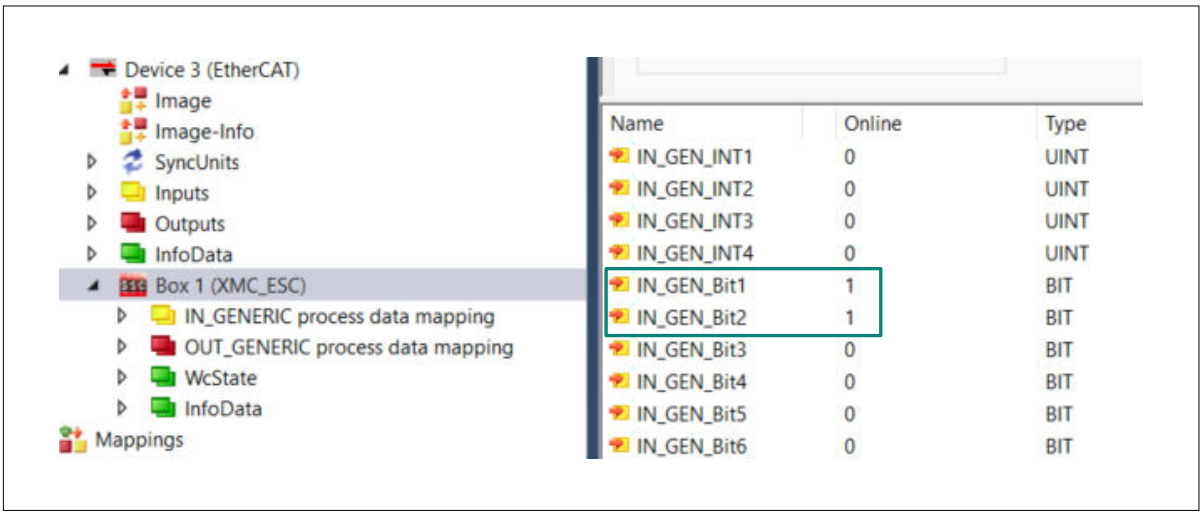


Figure 25 Button states on XMC4800 Relax Kit

Similarly, BUTTON1 on the XMC4300 Relax EtherCAT Kit is also mapped to IN_GEN_Bit1, and its status can be verified in TwinCAT by pressing and holding the button

8. Right-click on the OUT_GEN_Bit1 of the slave node (XMC4300 or XMC4800) and select "Online Write '1'" from the context menu. On the XMC4800 kit, this action will toggle LED1 (P1.14) to the "On" state, while

selecting "Online Write '0'" will toggle it to the "Off" state. The same logic applies to all LEDs, from LED1 to LED8, located on the upper board.

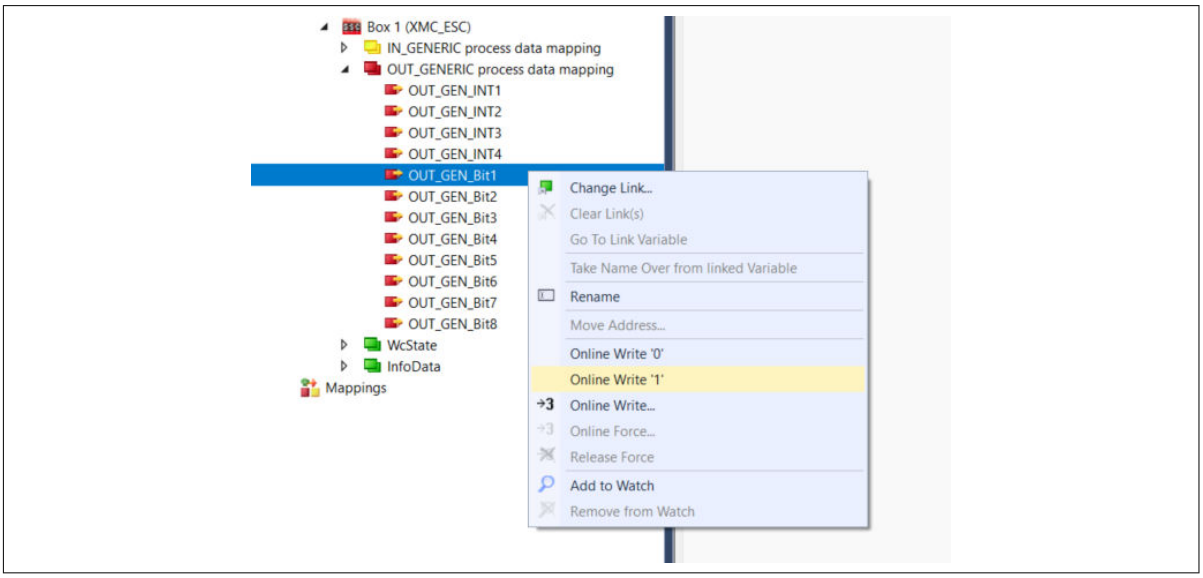


Figure 26 Online write to change the LED1 state from Low to High on XMC4800 Relax Kit

Similarly, online write 1 to the OUT_GEN_Bit1 and OUT_GEN_Bit2 changes the LED1 and LED2 states on the XMC4300 Relax Kit from Low to High

9. Another test that can be performed on the XMC4800 kit is to control the brightness of LED2 (P5.8). This is achieved by adjusting the duty cycle of the PWM signal that drives LED2. To do this, right-click on OUT_GEN_INT1 and select **Online Write**. In the **Set Value Dialog** window, enter a value within the range of 0 to 65535.

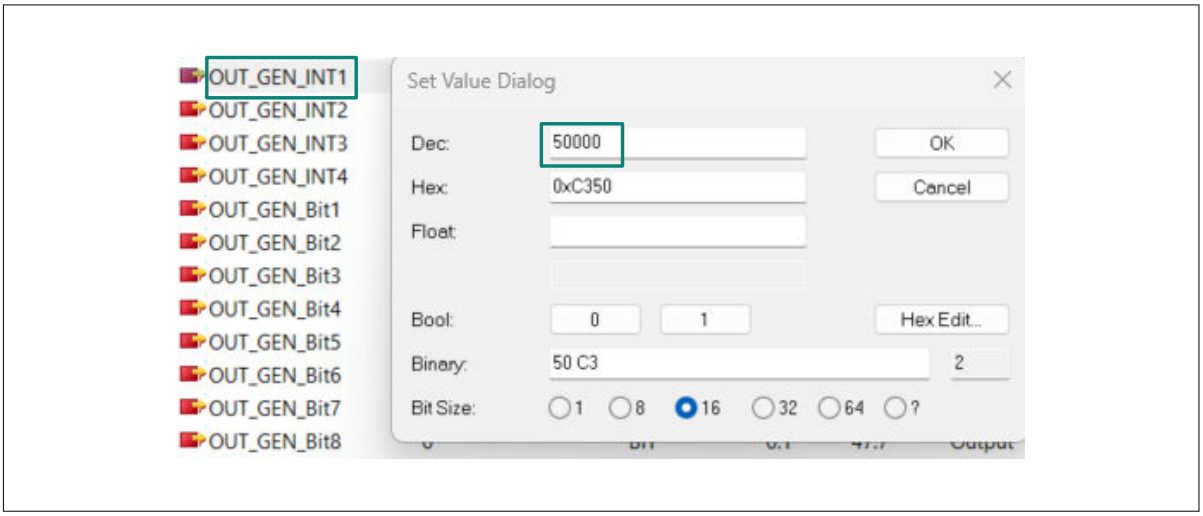


Figure 27 Setting values to control the brightness of LED2 on the XMC4800 Kit

***Note**: The lower the value, the brighter LED2 will be. Conversely, to turn off LED2, set the value to 65535 (the maximum value) and observe the resulting effect.*

References

User guides

[1] Infineon Technologies AG: *XMC4300 Relax EtherCAT Kit for XMC4000 family*; Available online
[2] Infineon Technologies AG: *Evaluation Board for XMC4000 family - XMC4700/XMC4800 Relax Kit Series-V1*; Available online

Datasheets

[3] Infineon Technologies AG: *XMC4000 Family, XMC4300*; Available online
[4] Infineon Technologies AG: *XMC4000 Family, XMC4700/XMC4800*; Available online

Reference manuals

[5] Infineon Technologies AG: *XMC4000 Family, XMC4300*; Available online
[6] Infineon Technologies AG: *XMC4000 Family, XMC4800*; Available online

Code examples

[7] Infineon Technologies AG: *XMC4300 EtherCAT SSC*; Available online
[8] Infineon Technologies AG: *XMC4800 EtherCAT SSC*; Available online

Additional information

[9] Beckhoff: *EtherCAT Distributed Clocks - default settings*; Available online

Revision history

Document revision	Date	Description of changes
**	2025-09-05	Initial release

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2025-09-05

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2025 Infineon Technologies AG

All Rights Reserved.

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-nzr1755773894650

Important notice

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.