AN14909

CODESYS Benchmarking on i.MX 8M Mini EVK/i.MX 943 EVK/i.MX 95 EVK

Rev. 1.0 — 28 January 2026 Application note

Document information

Information	Content
Keywords	AN14909, EtherCAT, CODESYS, PLC, i.MX 8M Mini, i.MX 943, i.MX 95, Benchmarking
Abstract	This application note describes the method for deploying and running CODESYS on i.MX series processors.

1 Introduction

As industrial automation systems demand greater flexibility, scalability, and real-time responsiveness, embedded platforms are playing an increasingly critical role in control architectures. The **i.MX series processors** of NXP, known for their high performance, low power consumption, and rich peripheral interfaces, have become a better choice in industrial control applications.

At the same time, **CODESYS** has emerged as a mature automation software platform based on the International Electrotechnical Commission (IEC) 61131-3 standard. It provides a powerful Programmable Logic Controller (PLC) programming environment and supports a wide range of industrial protocols, which implement the control systems on general-purpose hardware. Running CODESYS on the i.MX platform enables soft PLC functionality and allows efficient communication with field devices via industrial Ethernet protocols such as EtherCAT, which enables flexible and programmable control systems.

This application note introduces the deployment and operation of CODESYS on i.MX processors, which coveres key technical aspects such as system architecture, software environment setup, performance benchmark of Real Time Edge, and integration with EtherCAT Subdevices. Through this document, developers can gain a clear understanding of how to implement soft PLC solutions on embedded platforms and build next-generation industrial control systems.

2 Overview

This section introduces EthernetCAT, PLC, and CODESYS.

2.1 EtherCAT introduction

EtherCAT is an Ethernet-based field-bus system, invented by Beckhoff Automation. The protocol is standardized in IEC 61158 and is suitable for both hard and soft real-time computing requirements in automation technology. EtherCAT was developed to apply Ethernet for automation applications requiring short data update times (also called cycle times) with low communication jitter.

An EtherCAT network consists of a Maindevice and multiple Subdevices. The EtherCAT Maindevice sends an Ethernet encapsulated EtherCAT frame that passes through each node. Each EtherCAT Subdevice reads the data addressed to it on the fly and inserts its data in the frame as the frame is moving downstream. The only delay in the frame is due to the hardware propagation delay times. The last node in a segment detects an open port and sends the frame back to the Maindevice using Ethernet technology’s full duplex feature. The EtherCAT Maindevice is the only node within a segment allowed to actively send an EtherCAT packet; all other nodes merely forward packets downstream. This concept prevents unpredictable delays and guarantees real-time capabilities.

EtherCAT Subdevices use EtherCAT Slave Contoller (ESC) in the form of an Application Specific Integrated Circuits (ASIC), FGPA, or integrated in a standard microcontroller. The use of ESC enables processing frames on the fly and entirely in hardware, which makes network performance predictable.

The EtherCAT protocol is optimized for short cyclic process data. EtherCAT embeds its payload in a standard Ethernet frame. The EtherCAT frame is identified with `0x88A4` in the EtherType field. The EtherCAT frame contains the frame header and one or more datagrams.

Figure 1 shows the EtherCAT frame structure.

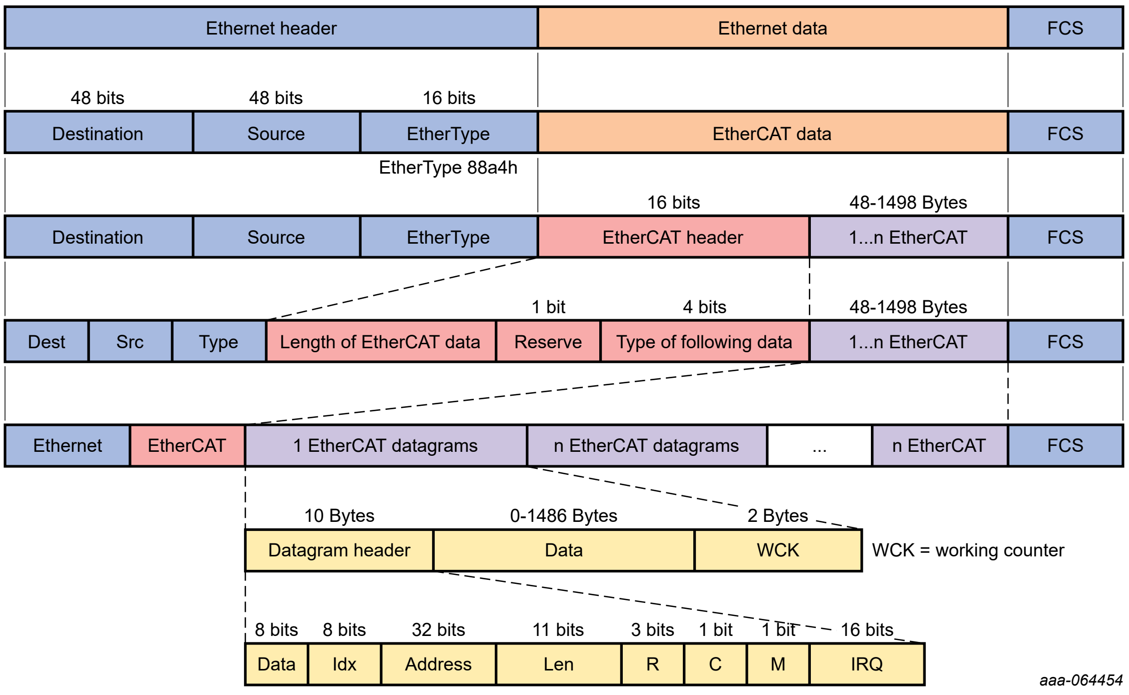


Figure 1. EtherCAT frame format

2.2 PLC introduction

PLC is a specialized industrial computer used to control machinery and processes in manufacturing and automation environments. It continuously monitors inputs from sensors and devices, processes this data based on a programmed logic, and then triggers outputs to control actuators like motors, valves, or lights. PLCs are robust and reliable, capable of operating in harsh conditions such as high temperatures, electrical noise, and vibration. They are programmed using languages like Ladder Logic or Structured Text, making them highly adaptable for various control tasks. PLCs play a crucial role in ensuring efficiency, safety, and precision in automated systems.

2.3 CODESYS introduction

Controller Development System (CODESYS) is a versatile and widely adopted software platform used for programming and engineering control systems, especially industrial automation. It provides an Integrated Development Environment (IDE) that allows engineers to create PLC applications using IEC 61131-3 standard programming languages such as Ladder Diagram, Structured Text, and Function Block Diagram. The IDE also supports visualization tools for monitoring and interacting with control processes. In addition to the development environment, CODESYS includes a runtime system that can be installed on various intelligent devices, including PLCs and embedded systems. For example, CODESYS Control for Linux ARM SL enables Arm-based devices running Linux to function as control systems, even in applications requiring hard real-time performance. This flexibility and hardware independence make CODESYS a powerful solution for modern automation needs.

3 Evaluation platform

This section introduces the evaluation platform.

3.1 Hardware

Performance metrics are captured using NXP evaluation platforms configured as test controllers. Each platform consists of a multicore Arm-based processor and LPDDR memory subsystem. The test environment includes a network of peripheral or endpoint devices connected through a standard communication interface, enabling consistent measurement of data throughput, latency, and CPU utilization across different SoCs.

All evaluation platforms operate under identical software and hardware configurations to ensure fair comparison. Each platform is benchmarked under the same workload, with the same memory interface width and speed classes used to evaluate performance scaling across device families.

Differences among platforms primarily include variations in processor architecture, core frequency, cache hierarchy, and external memory speed. These distinctions help characterize system performance in representative embedded use cases such as industrial control, edge processing, and communication workloads.

Table 1 summarizes the key hardware characteristics of the evaluated platforms.

Table 1. Key hardware characteristics of evaluated platforms

Features	i.MX 943 LPDDR5 EVK	i.MX 95 LPDDR5 EVK	i.MX 8M Mini EVK
Main CPU	4 × Arm Cortex A55	6 × Arm Cortex A55	4 × Arm Cortex A53
CPU Frequency	1.7 GHz	1.8 GHz	1.8 GHz
L1 Cache	32 kB I-cache 32 kB D-cache	32 kB I-cache 32 kB D-cache	32 kB I-cache 32 kB D-cache
L2 Cache	64 kB	64 kB	512 kB
L3 Cache	1 MB	512 kB	—
External Memory on EVK	LPDDR5 32-bit width	LPDDR5 32-bit width	LPDDR4 32-bit width
Memory Speed	Up to 4.2 GT/s	Up to 6.4 GT/s	Up to 3.0 GT/s

Taking the i.MX 8M Mini EVK as an example, the EtherCAT performance metrics are generated with running as EtherCAT controller using the CODESYS EtherCAT stack. Four Beckhoff digital output devices EL2008 and three Beckhoff digital output devices EL1018 are connected to the EtherCAT controller using a Beckhoff EtherCAT coupler EK1100.

Figure 2 shows the network architecture of EtherCAT control system used for Benchmark:

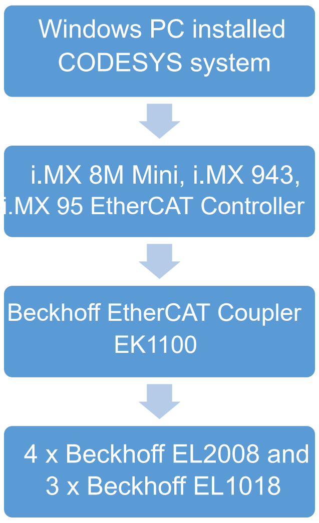

Windows PC installed CODESYS system
i.MX 8M Mini, i.MX 943, i.MX 95 EtherCAT Controller
Beckhoff EtherCAT Coupler EK1100
4 x Beckhoff EL2008 and 3 x Beckhoff EL1018

Figure 2. Network architecture of EtherCAT control system

Figure 2 shows the hardware connection to run EtherCAT Performance Metrics on i.MX 943 EVK.

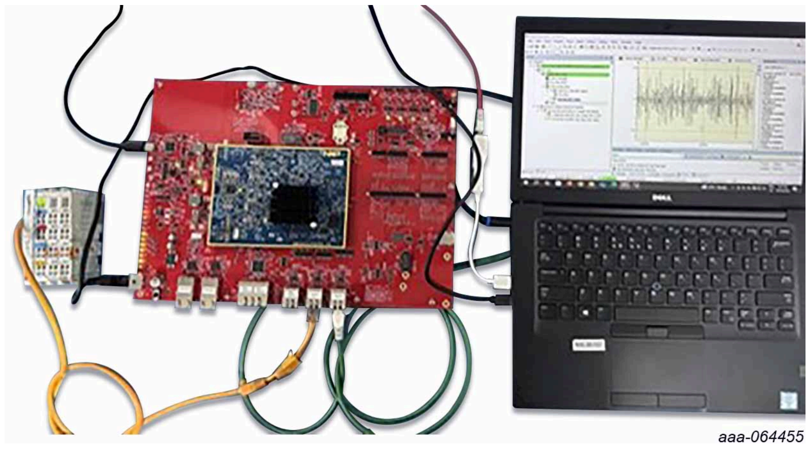


Figure 3. Hardware connection to run EtherCAT performance metrics on i.MX 943 EVK

3.2 Software

The i.MX 8M Mini EVK, i.MX 943 LPDDR5 EVK, and i.MX 95 LPDDR5 EVK are running Real-time Edge Software v3.2 booting with SD card.

Download and install the **CODESYS Development System V3 3.5.20.20 64 bit (.exe)** from CODESYS official website to your Windows computer.

Download the **CODESYS Control for Linux Arm SL 4.11.0.0** runtime package from CODESYS official website and decompress it using the commands below:

```
# unzip 'CODESYS Control for Linux ARM64 SL 4.11.0.0.package'
# cd Delivery/linuxarm64/
# dpkg-deb -R codesyscontrol_linuxarm64_4.11.0.0_arm64.deb software
# cd software/opt/codesys/bin/
```

The runtime binary *codesyscontrol.bin* is located in the *software/opt/codesys/bin/* directory.

In the Windows environment, double-click **CODESYS Control for Linux ARM64 SL 4.11.0.0. package** to install it onto your CODESYS.

For detailed hardware and software setup procedures for using the CODESYS stack to configure NXP i.MX EVKs as EtherCAT controllers, see Section 6.

4 Performance metrics

This section introduces the performance metrics.

4.1 Cyclictest performance metrics

Before evaluating the performance of each hardware platform running an EtherCAT application, it is advisable to collect baseline real-time performance data using cyclictest.

Cyclictest is a Linux command-line utility used to measure real-time latency in a system. It measures task scheduling latency under various system loads, offering valuable insight into the real-time determinism of a platform. It is part of the rt-tests. It creates one or more threads that sleep for a fixed interval. After each sleep, it measures the delay between the expected wake-up time and the actual wake-up time. These statistics serve as a reliable baseline for assessing how well the hardware can perform as an EtherCAT controller when running the CODESYS runtime stack.

This command measures real-time latency of the system under stress, which continues for six hours. The latency data is saved to a histogram file:

```
stress-ng -c 4 --cpu-method all & cyclictest -m -Sp98 -D6h -h400 -i200 -q >
 8mm_6h.hist &
```

Figure 4 to Figure 7 show the cyclictest latency plot running on the i.MX 943 LPDDR5, i.MX 95 LPDDR5, and i.MX 8M Mini EVK separately for six6 hours under `stress-ng`.

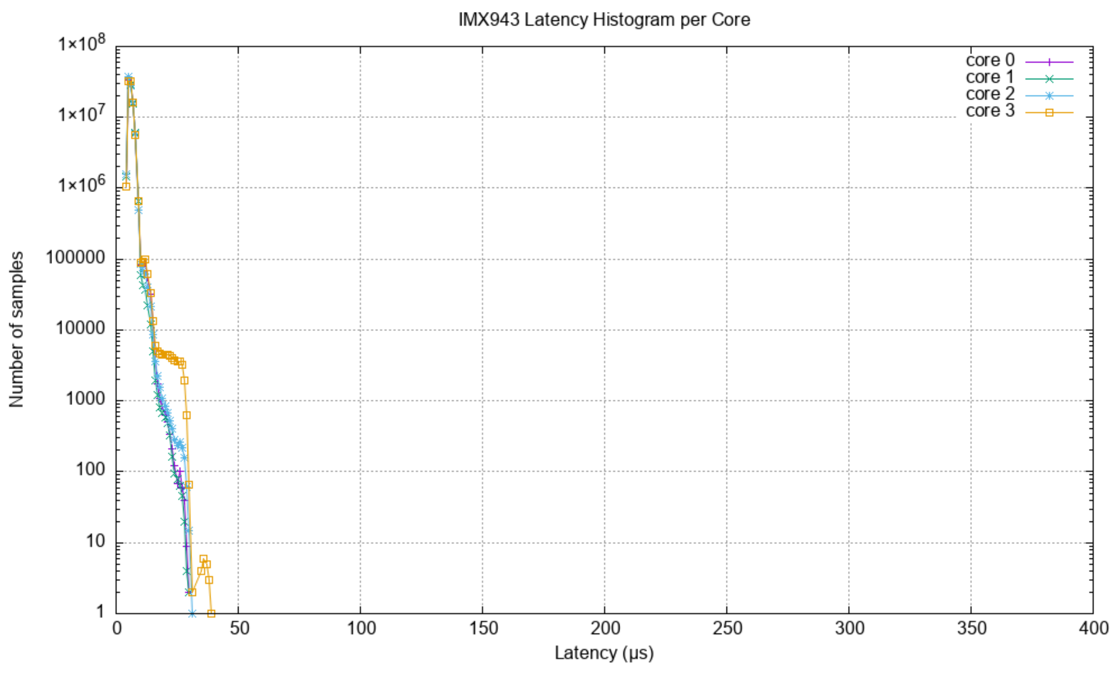


Figure 4. Latency plot for i.MX 943 running for six6 hours under stress-ng

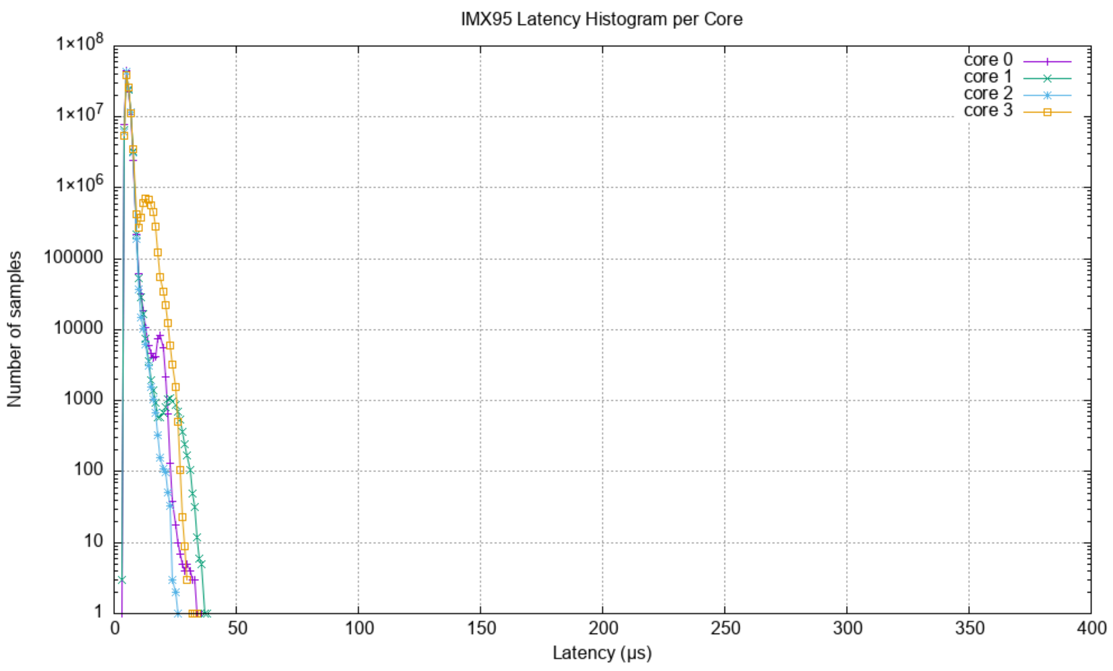


Figure 5. Latency plot for i.MX 95 running for six hours under stress-ng

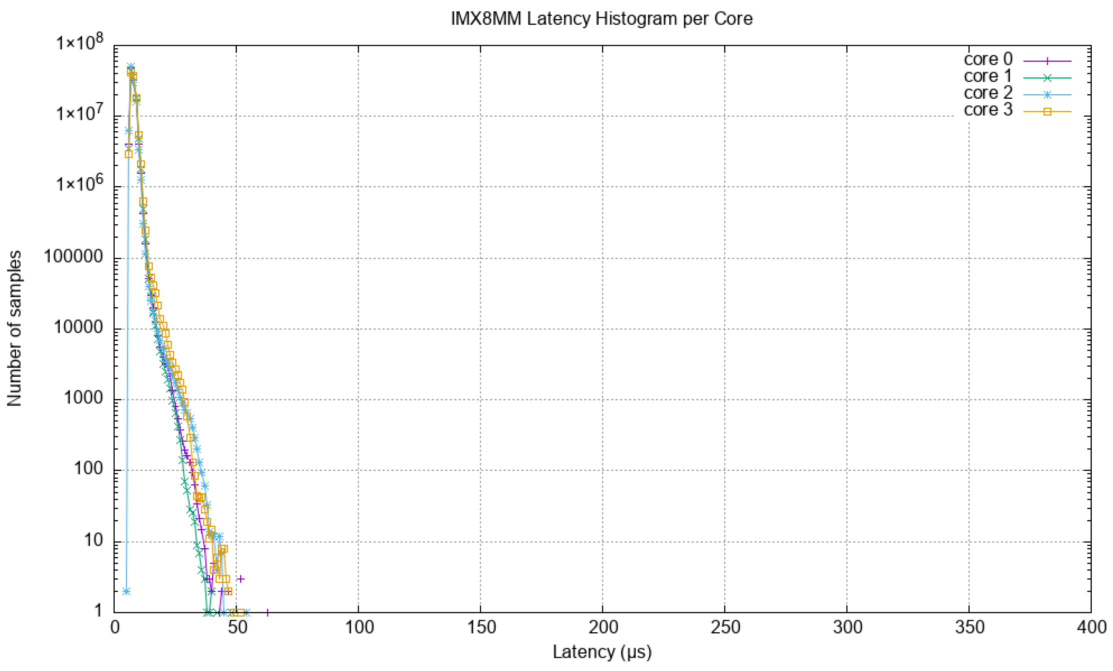


Figure 6. Latency plot for i.MX 8M Mini (1.2 GHz) running for six hours under stress-ng

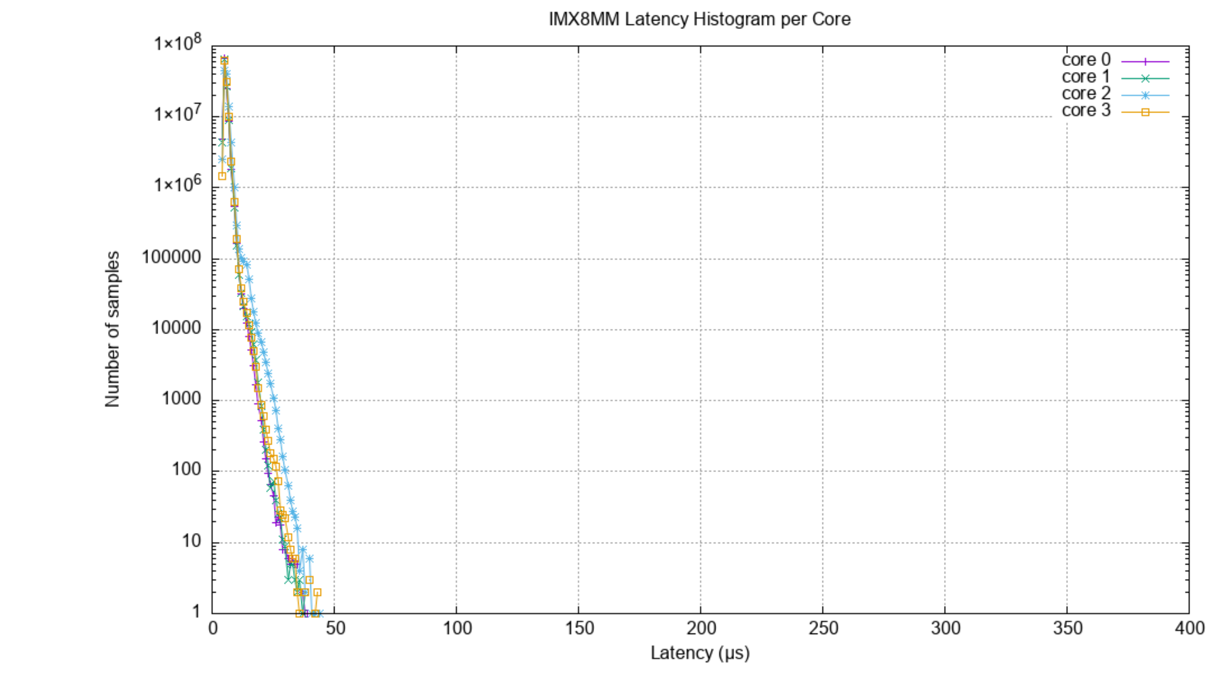


Figure 7. Latency plot for i.MX 8M Mini (1.6 GHz) running for six hours under stress-ng

Table 2 summarizes the maximum latency using cyclictest on i.MX 943, i.MX 95, and i.MX 8M Mini EVK. The results show that the i.MX 943 has the best latency performance among all the platforms, followed by i.MX 95 and i.MX 8M Mini. i.MX 95 has very close performance to i.MX 943, confirming strong real-time characteristics.

Table 2. Maximum latency using cyclictest

Hardware platform	i.MX 943 LPDDR5 EVK	i.MX 95 LPDDR5 EVK	i.MX 8M Mini EVK (1.2 GHz)	i.MX 8M Mini EVK (1.6 GHz)
Maximum latency	38 µs	39 µs	55 µs	44 µs

4.2 EtherCAT performance metrics

Filtered maximum cycle time is the highest cycle time recorded after removing outliers. It indicates the worst-case scheduling delay in the system. Filtered Maximum Jitter is the largest variation in cycle time after filtering. It reflects the timing stability and consistency of the system. Both are important metrics to reflect EtherCAT performance. The metrics results are measured by CODESYS. For detailed steps, see Section 6.5.3.

Table 3 shows the maximum cycle time and maximum jitter, which use i.MX 943 LPDDR5 EVK, i.MX95 LPDDR5 EVK, and i.MX 8M Mini EVK as EtherCAT controller, use CODESYS to read/write four Beckhoff EL2008 or three Beckhoff EL1018 Benchoff EtherCAT devices. The tests run for one hour.

Table 3. Filtered maximum cycle time and filtered maximum jitter

Hardware platform	ECAT network	Filtered maximum cycle time[1]	Filtered maximum jitter[1]
i.MX 943 LPDDR5 EVK	1 × Beckhoff EK1100, 4 × Beckhoff EL2008, and 3 × Beckhoff EL1018	91 µs	15 µs
i.MX 95 LPDDR5 EVK	1 × Beckhoff EK1100, 4 × Beckhoff EL2008, and 3 × Beckhoff EL1018	136 µs	12 µs
i.MX 8M Mini EVK (1.2 GHz)	1 × Beckhoff EK1100, 4 × Beckhoff EL2008, and 3 × Beckhoff EL1018	147 µs	12 µs

Table 3. Filtered maximum cycle time and filtered maximum jitter...*continued*

Hardware platform	ECAT network	Filtered maximum cycle time[1]	Filtered maximum jitter[1]
i.MX 943 LPDDR5 EVK	1 × Beckhoff EK1100, 1 × Beckhoff EL2008, and 1 × Beckhoff EL1018	90 µs	15 µs
i.MX 95 LPDDR5 EVK	1 × Beckhoff EK1100, 1 × Beckhoff EL2008, and 1 × Beckhoff EL1018	129 µs	12 µs
i.MX 8M Mini EVK (1.2 GHz)	1 × Beckhoff EK1100, 1 × Beckhoff EL2008, and 1 × Beckhoff EL1018	111 µs	10 µs
i.MX 8M Mini EVK (1.6 GHz)	1 × Beckhoff EK1100, 1 × Beckhoff EL2008, and 1 × Beckhoff EL1018	101 µs	10 µs

[1] The filtered maximum cycle time and filtered maximum jitter are recalculated based on the maximum cycle time and maximum jitter after clicking the **Reset** button in CODESYS.

The results show that:

- i.MX 943 has good performance, low jitter, and moderate cycle time.
- i.MX 95 has slightly higher cycle time but very low jitter, which means good stability.
- i.MX 8M Mini has the highest cycle time among them but jitter remains low.

4.3 CODESYS motor control benchmarking

Due to limited availability of Beckhoff IO device and CODESYS license constraints, commercial servo motors were used to evaluate EtherCAT performance across multiple control loop times (500 µs, 1 ms, and 2 ms) on NXP i.MX platforms, to show optimized native driver performance.

Table 4. i.MX 95 EtherCAT Benchmark results

Cycle time	Supported axes	CPU usage (%)	Max. jitter (µs)
2 ms	64	39.8	61
1 ms	58	88.3	152
500 µs	26	62.1	98

Notes for i.MX 95:

- With the optimized driver, up to 26 servos can be supported at 500 µs, and up to 58 servos at 1 ms.

Table 5. i.MX 943 EtherCAT Benchmark results

Cycle time	Supported axes	CPU usage (%)	Max. jitter (µs)
2 ms	64	24.9	48
1 ms	64	63.9	97
500 µs	36	87.6	149

Notes for i.MX 943:

- With the optimized driver, up to 36 servos can be supported at 500 µs.
- The reason why the i.MX 943 performs better than the i.MX 95, according to the comparison, is that it has a larger L3 cache.

Table 6. i.MX 8M Mini EtherCAT Benchmark results

Cycle time	Supported axes	CPU usage (%)	Max. jitter (µs)
2 ms	64	40.52	59
1 ms	58	94.48	190
500 µs	26	62.6	116

Notes for i.MX 8M Mini:

- With the optimized driver, up to 26 servos can be supported at 500 µs.

5 Optimization

To improve the performance, perform the steps described below, and different SoC needs different steps.

5.1 Isolate CPU core for real-time tasks

Real-time tasks such as `EtherCAT_Task` can experience performance degradation due to interference from other threads scheduled by the Linux kernel. To mitigate this, it is advisable to isolate the CPU core designated for real-time processing from the kernel scheduler.

By default, EtherCAT_Task is bound to CPU1. To isolate this core, execute the following command in the U-Boot environment:

```
uboot => setenv mmcargs 'setenv bootargs ${jh_clk} ${mcore_clk} console=
${console} root=${mmcroot} isolcpus=1'
```

This command appends the `isolcpus=1` parameter to the kernel boot arguments, instructing the kernel to exclude CPU1 from its scheduling domain. This ensures that CPU1 is reserved exclusively for real-time tasks, reducing latency and improving determinism.

5.2 Load appropriate device tree for EtherCAT

By default, the system uses a standard device tree configuration that initializes EtherCAT through a predefined driver path. However, to enable a customized EtherCAT implementation, such as one provided by a real-time extension or a vendor-specific stack, a modified device tree is required. This custom DTB redirects the initialization flow to an alternative driver path, effectively overriding the default EtherCAT driver behavior.

In the U-Boot prompt, set the DTB file as follows:

```
# For i.MX8MM
uboot => setenv fdtfile imx8mm-evk-ecat.dtb
```

This command sets the device tree file to *imx8mm-evk-ecat.dtb*, which includes the necessary modifications to reroute EtherCAT initialization. As a result, the system bypasses the default driver and instead loads the customized EtherCAT driver during kernel boot.

5.3 Set CPU frequency to performance mode

To maintain consistent high performance on CPU1, it is essential to disable dynamic frequency scaling. This can be achieved by setting the CPU frequency governor to performance mode, which locks the CPU at its maximum frequency.

Execute the following command after the system boots:

```
echo performance > /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor
```

This configuration prevents the CPU from scaling down its frequency. It reduces latency and ensures that real-time tasks have access to the maximum processing power.

5.4 Disable DHCP client (udhcpc)

During the system startup, the udhcpc (BusyBox DHCP client) may automatically attach to the Ethernet interface.

Since EtherCAT requires direct hardware access to the Ethernet port, any running network service (like `udhcpc`) can interfere with EtherCAT communication and reduce performance.

To release the Ethernet interface before starting EtherCAT, stop the DHCP client:

```
kill -9 udhcpc_PID
```

Replace `udhcpc_PID` with the actual process ID of the running `udhcpc` instance. For example:

```
ps | grep udhcpc
kill -9 3468
```

6 Real-time Edge PLC image setup and CODESYS execution guide

This section describes how to build a Real-time Edge PLC image and how to set a CODESYS test project as an example to drive motors.

6.1 Overview

Based on the real-time Edge Yocto project, a new Yocto distro named *nxp-real-time-edge-plc* is added, which is specific to the PLC use case.

For more information about i.MX Yocto project, see the *i.MX Yocto Project User's Guide* (document UG10164).

For additional information, see the *Real-time Edge Yocto Project User Guide* (document RTEDGEYOCTOUG) at REALTIME EDGE Documentation.

6.2 Features

Features are as follows:

- **Optimized native driver**
  Typical industrial software, for example CODESYS or IGH EtherCAT MainDevice stack, works on user space and communicates using Linux standard network interface. To reduce the latency when Ethernet raw packets pass from user space through the Linux standard network, the network driver is optimized to avoid memory reallocation and copying, task rescheduling. The latency of the critical path is reduced for packet transmitting and receiving.
- **Light root filesystem**
  A lighter root filesystem saves CPU cycles that the PLC user program uses.
- **Tested platforms**
  – imx8mm-lpddr4-evk
  – imx95-19x19-lpddr5-evk

– imx943-19x19-lpddr5-evk

6.3 Build the image

This section provides detailed information along with the procedure for building an image.

6.3.1 Build configurations

A new yocto distro named *nxp-real-time-edge-plc* is added for the PLC use case, and below platforms are tested:

- imx8mm-lpddr4-evk
- imx95-19x19-lpddr5-evk
- imx943-19x19-lpddr5-evk

Real-time Edge provides the script *real-time-edge-setup-env.sh* to simply the setup for both i.MX and Layerscape boards. To use the script, the name of the specific machine to be built for and the desired distro must be specified. The script sets up a directory and the configuration files for the specified machine and distro.

The syntax for the *real-time-edge-setup-env.sh* script is shown below:

```
$ DISTRO=nxp-real-time-edge-plc MACHINE=<machine name> source real-time-
edgesetup-env.sh -b <build dir>
```

Where:

- **MACHINE=<machine name>** is the name of the supported platforms.
- **-b <build dir>** specifies the name of the build directory created by the *real-time-edge-setup-env.sh* script.

After the script runs, the working directory is the one just created by the script, specified with the *-b* option. A *conf* folder is created containing the files *bblayers.conf* and *local.conf*.

The *local.conf* file contains the machine and distro specifications. An example is shown below:

```
MACHINE ??= 'imx8mm-lpddr4-evk'
DISTRO ?= 'nxp-real-time-edge-plc'
ACCEPT_FSL_EULA = "1"
```

The MACHINE configuration can be changed by editing this file, if necessary.

6.3.2 Build scenarios

The following are build setup scenarios for various configurations. Set up the manifest and populate the Yocto project layer sources using the commands below:

```
$ mkdir yocto-real-time-edge
$ cd yocto-real-time-edge
$ repo init -u https://github.com/nxp-real-time-edge-sw/yoctoreal-time-edge.git/
yocto-real-time-edge.git \
-b real-time-edge-walnascar \
-m real-time-edge-3.2.0.xml
$ repo sync
```

The following sections give some specific examples.

- Real-time Edge PLC image on i.MX 8M Mini EVK

```
$ DISTRO=nxp-real-time-edge-plc MACHINE=imx8mm-lpddr4-evk source real-timeedge-
```

```
setup-env.sh -b build-imx-real-time-edge-plc
$ bitbake nxp-image-real-time-edge-plc
```

- Real-time Edge PLC image on i.MX 95 EVK

```
$ DISTRO=nxp-real-time-edge-plc MACHINE=imx95-19x19-lpddr5-evk source realtime-
edge-setup-env.sh -b build-imx-real-time-edge-plc
$ bitbake nxp-image-real-time-edge-plc
```

- Real-time Edge PLC image on i.MX 943 EVK

```
$ DISTRO=nxp-real-time-edge-plc MACHINE=imx943-19x19-lpddr5-evk source
 realtime-
edge-setup-env.sh -b build-imx-real-time-edge-plc
$ bitbake nxp-image-real-time-edge-plc
```

6.3.3 Image deployment

All filesystem images are deployed to the *<build directory>/tmp/deploy/images* directory. Each image building creates a U-Boot image, a kernel image, and an image type based on the *IMAGE_FSTYPES* variable defined in the machine configuration file. Most machine configurations provide an SD card image (*.wic*) and a rootfs image (*.tar*). The SD card image contains a partitioned image (with U-Boot, kernel, rootfs, and other such files) suitable for booting the corresponding hardware.

Copy image to SD card:

The SD card image file, (*<image_name>.wic*) contains a partitioned image (with U-Boot, kernel, rootfs, and other files) suitable for booting the corresponding hardware. To copy the image to an SD card, run the following commands:

```
$ zstd -d <image_name>.wic.zst
$ sudo dd if=<image_name>.wic of=/dev/sd<disk> bs=1M conv=fsync
```

6.4 Board environment setup

This section describes how to set the board environment.

6.4.1 i.MX 8M Mini EVK board

- Hardware Requirements
  - Network cables
  - Mini/micro USB cable
  - Network port converter with type-c interface (i.MX 8M Mini has only one network port, so it needs another network port connected with CODESYS IDE)
  - Beckhoff IO device
  - Personal Computer on which the CODESYS has been installed
- Prepare the example
  - Connect the IO Devices to the network port of the board
  - Plug the Network port converter into the USB port (labeled PORT1) of the board, and then connect the network port converter to the PC
  - Connect a USB cable between the host PC and the OpenSDA USB port on the target board, and open a serial terminal with the following settings:
    - 115,200 baud rate
    - Eight data bits

- No parity
- One stop bit
- No flow control

– Figure 8 describes the i.MX 8M Mini board connection.

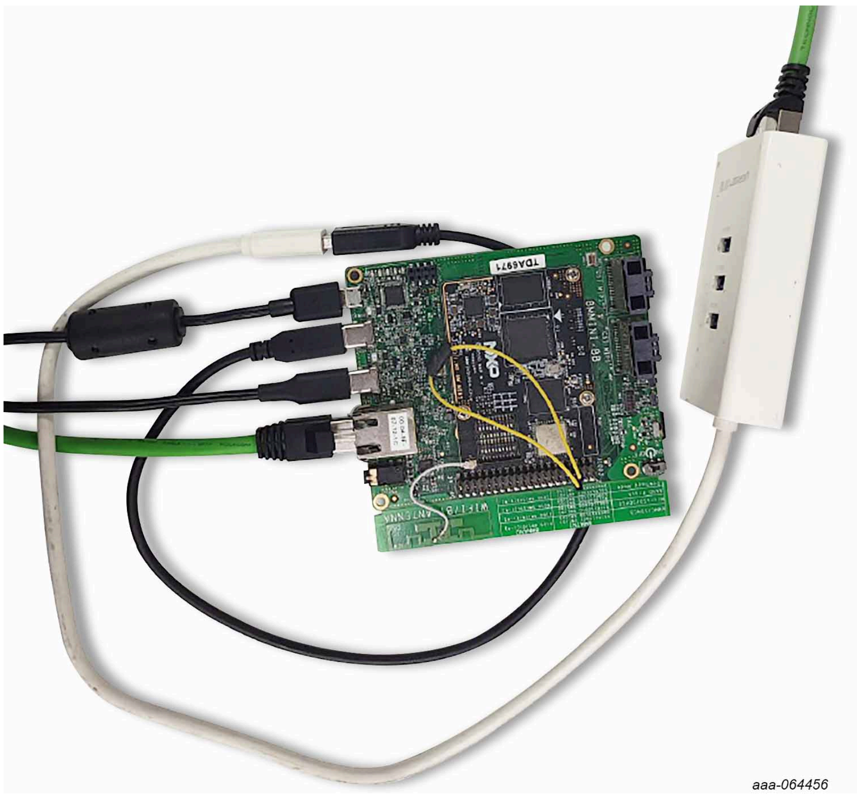


Figure 8. i.MX 8M Mini board connection

6.4.2 i.MX 95 EVK board

- Hardware Requirements
  - Network cables
  - Mini/micro USB cable
  - Beckhoff IO device
  - Personal Computer on which the CODESYS has been installed
- Prepare the example
  - i.MX95-15x15-lpddr4x-evk:
    - Connect the PC to the network port labeled **J28** on the board
    - Connect the IO devices to the network port labeled **J11** on the board
  - i.MX95-19x19-lpddr5-evk:
    - Connect the PC to the network port labeled “**J9**” on the board
    - Connect the IO devices to the network port labeled “**J8”** on the board
  - General Configuration:
    - Connect a USB cable between the host PC and the OpenSDA USB port on the target board, and open a serial terminal with the following settings:
      - 115,200 baud rate
      - Eight data bits

- No parity
- One stop bit
- No flow control

6.4.3 i.MX 943 EVK board

- Hardware Requirements
  - Network cables
  - Mini/micro USB cable
  - Beckhoff IO device
  - Personal Computer on which the CODESYS has been installed
- Prepare the example
  - Connect the PC to the network port labeled **J27** on the board
  - Connect the IO devices to the network port labeled **J26** on the board
  - Connect a USB cable between the host PC and the OpenSDA USB port on the target board, and open a serial terminal with the following settings:
    - 115,200 baud rate
    - Eight data bits
    - No parity
    - One stop bit
    - No flow control

6.5 Running CODESYS on the boards

This section describes how to run CODESYS on the boards.

6.5.1 Running CODESYS on i.MX 8M Mini

To isolate CPU1 (by default, the `EtherCAT_Task` thread created by the Realtime of CODESYS is bound into CPU1), execute the following command.

```
uboot ==> setenv mmcargs 'setenv bootargs ${jh_clk} ${mcore_clk} console=
${console} root=${mmcroot} isolcpus=1'
```

To update the *dtb* file and boot into the kernel, execute the following command.

```
uboot ==> setenv fdtfile imx8mm-evk-ecat.dtb
uboot ==> run bootcmd
```

tTo use the highest frequency, set the CPU frequency governor to *performance* for the **CPU1** core.

```
echo performance > /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor
```

Stop the **udhcpc** service whose interface is connected to the EtherCAT servos.

```
# ethercat_net=`ls /sys/bus/platform/devices/30be0000.ethernet/net` && echo
$ethercat_net
# ps
// find “PID root 3468 S udhcpc -R -b -p /var/run/udhcpc.eth0.pid -I
 $ethercat_net”
# kill -9 PID
```

Enable the EtherCAT port.

```
# ethercat_port=`ls /sys/bus/platform/devices/30be0000.ethernet/net` && echo
 $ethercat_port
# ifconfig $ethercat_port up
```

Else assign the IP address manually.

```
// assume the name of the USB network port is eth1
# ifconfig eth1 192.168.1.xx
```

To copy **codesyscontrol.bin** into the board and start it, use the `scp` command:

```
# ./codesyscontrol.bin > CODESYS.log &
```

Note:

- *To obtain codesyscontrol.bin, see Section 6.6.1.2.*
- *CODESYS logs can be viewed in the CODESYS.log file.*

6.5.2 Running CODESYS on i.MX 95

To isolate CPU1 (by default, the `EtherCAT_Task` thread created by the Realtime of CODESYS is bound into CPU1), execute the following command.

```
uboot ==> setenv mmcargs 'setenv bootargs ${jh_clk} ${mcore_clk} console=
${console} root=${mmcroot} isolcpus=1'
uboot ==> run bootcmd
```

Stop the **udhcpc** service whose interface is connected to the EtherCAT servos.

```
# ps
// find “PID root 3468 S udhcpc -R -b -p /var/run/udhcpc.eth0.pid -i eth0”
# kill -9 PID
```

Enable the EtherCAT port.

```
# imx95-15x15-lpddr4x-evk(J8):
# echo -n 0001:00:00.0 > /sys/bus/pci/drivers/fsl_enetc4/unbind
# echo -n fsl_ecat_enetc4 > /sys/bus/pci/devices/0001:00:00.0/driver_override
# echo -n 0001:00:00.0 > /sys/bus/pci/drivers/fsl_ecat_enetc4/bind
# ifconfig eth0 up
# imx95-19x19-lpddr5-evk(J11):
# echo -n 0002:00:00.0 > /sys/bus/pci/drivers/fsl_enetc4/unbind
# echo -n fsl_ecat_enetc4 > /sys/bus/pci/devices/0002:00:00.0/driver_override
# echo -n 0002:00:00.0 > /sys/bus/pci/drivers/fsl_ecat_enetc4/bind
# ifconfig eth0 up
```

Else assign IP address manually (J9).

```
# ifconfig eth1 192.168.1.xx
```

Set the CPU frequency governor to *performance* for **CPU1** core to use the highest frequency.

```
# echo performance > /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor
```

Use the `scp` command to copy **codesyscontrol.bin** into the board and start it:

```
# ./codesyscontrol.bin > CODESYS.log &
```

Note:

- *To obtain the codesyscontrol.bin, see Section 6.6.1.2.*
- *CODESYS logs can be viewed in the CODESYS.log file.*

6.5.3 Running CODESYS on i.MX 943

To isolate CPU1 (by default, the `EtherCAT_Task` thread created by the Realtime of CODESYS is bound into CPU1), execute the following command.

```
uboot ==> setenv mmcargs 'setenv bootargs ${jh_clk} ${mcore_clk} console=
${console} root=${mmcroot} isolcpus=1'
uboot ==> run bootcmd
```

Stop the **udhcpc** service whose interface is connected to the EtherCAT servos.

```
# ps
// find “PID root 3468 S udhcpc -R -b -p /var/run/udhcpc.eth0.pid -i eth0”
# kill -9 PID
```

Enable the EtherCAT port (J26).

```
# echo -n 0001:01:08.0 > /sys/bus/pci/drivers/fsl_enetc4/unbind
# echo -n fsl_ecat_enetc4 > /sys/bus/pci/devices/0001:01:08.0/driver_override
# echo -n 0001:01:08.0 > /sys/bus/pci/drivers/fsl_ecat_enetc4/bind
# ifconfig eth1 up
```

Else assign IP address manually (J27).

```
# ifconfig eth2 192.168.1.xx
```

Set the CPU frequency governor to *performance* for **CPU1** core to use the highest frequency.

```
# echo performance > /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor
```

Use the **scp** command to copy **codesyscontrol.bin** into the board and start it:

```
# ./codesyscontrol.bin > CODESYS.log &
```

Note:

- *To obtain the codesyscontrol.bin, see Section 6.6.1.2.*
- *CODESYS logs can be viewed in the CODESYS.log file.*

6.6 Setting up CODESYS project

CODESYS is a powerful commercial PLC software programming tool. It supports IEC61131-3 standard IL, ST, FBD, LD, CFC, SFC, and six kinds of PLC programming languages. Users can choose different language editing subroutines in the same project, function module, and so on.

6.6.1 CODESYS project setup

This section describes how to set up the CODESYS project.

6.6.1.1 Downloading CODESYS software and runtime binary

Download and install the *CODESYS Development System V3 3.5.20.20 64 bit (.exe)* from CODESYS official website to your Windows.

Download CODESYS Control for Linux Arm SL 4.11.0.0 runtime package from CODESYS official website and decompress it using the commands below:

```
# unzip 'CODESYS Control for Linux ARM64 SL 4.11.0.0.package'
# cd Delivery/linuxarm64/
# dpkg-deb -R codesyscontrol_linuxarm64_4.11.0.0 _arm64.deb software
# cd software/opt/codesys/bin/
```

The runtime binary *codesyscontrol.bin* is located in the *software/opt/codesys/bin/* directory.

In the Windows environment, double click the **CODESYS Control for Linux ARM64 SL 4.11.0.0.** package to install it onto your CODESYS.

Note:

Download the 64-bit version for the platforms:

- *i.MX 8M Mini EVK*
- *i.MX 95 EVK*
- *i.MX 943 EVK*

6.6.1.2 Starting CODESYS runtime

Use the **scp** command to copy **codesyscontrol.bin** into the board and start it:

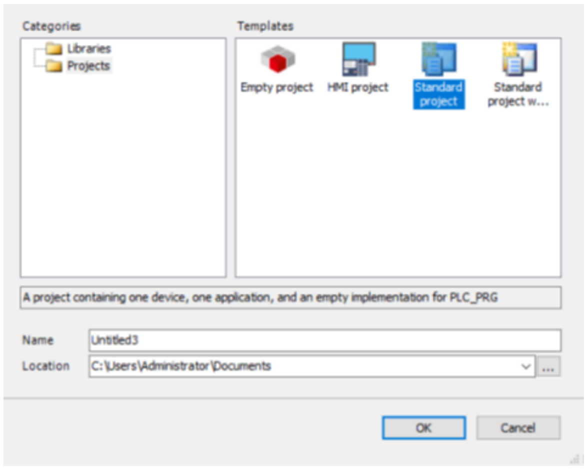
```
# ./codesyscontrol.bin > CODESYS.log &
```

CODESYS logs can be viewed in the *CODESYS.log* file.

6.6.1.3 Creating CODESYS project

This section describes the steps to create the CODESYS project in the machine on which the CODESYS tool has been installed.

1. Open **CODESYS** and build a new project using the steps below:
   a. Click **File** -> **New project**.
   b. Select **Standard project** as the template.
   c. Rename this project.
   d. Click **OK**.



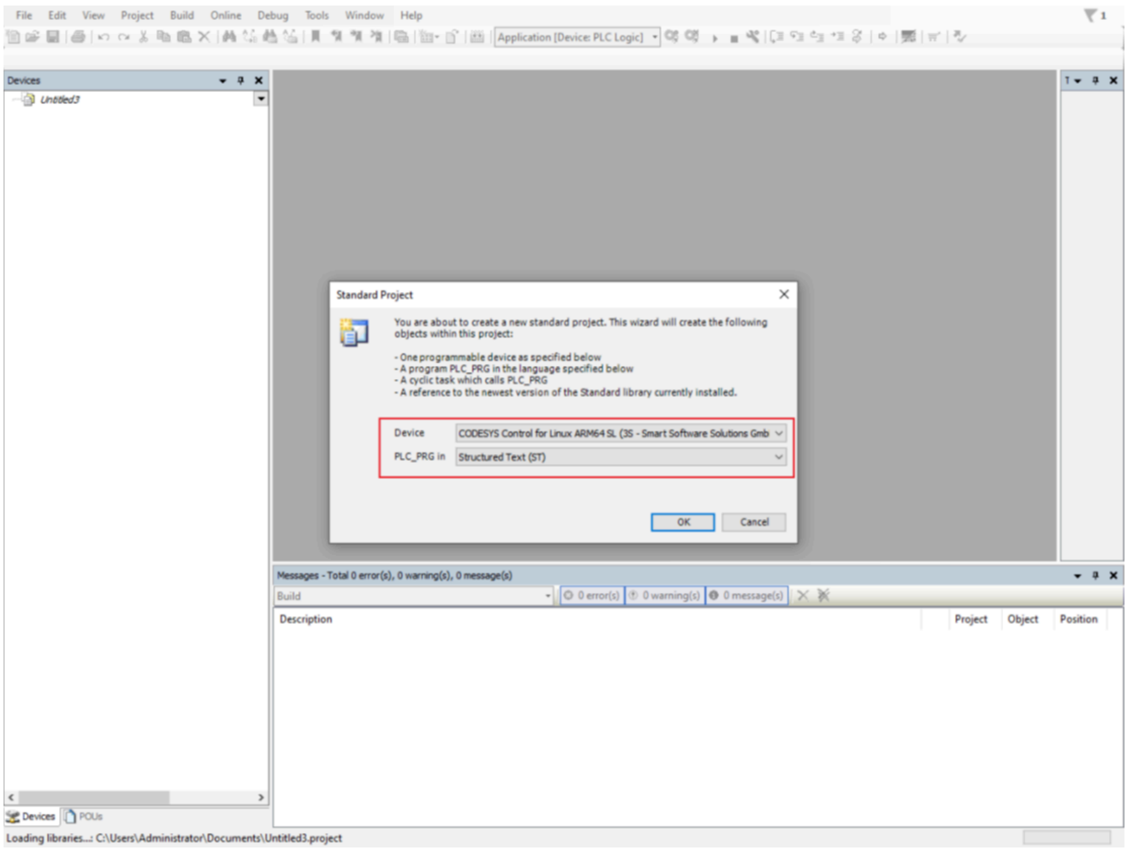
A project containing one device, one application, and an empty implementation for PLC_PRG

Name: Untitled3

Location: C:\Users\Administrator\Documents

Figure 9. Select Standard project as template

2. Set Device and PLC_PRG.
   a. Select **CODESYS Control for Linux ARM64 SL** or **CODESYS Control for Linux ARM SL** as the Device.
   b. Select **Structured Text(ST)** as the PLC_PRG.
   c. Click **OK**.


File Edit View Project Build Online Debug Tools Window Help
Application [Device: PLC Logic]
Devices
Untitled3
Project
Object
Position
Devices
POUs
OK
Cancel

Standard Project

You are about to create a new standard project. This wizard will create the following objects within this project:

- One programmable device as specified below
- A program PLC_PRG in the language specified below
- A cyclic task which calls PLC_PRG
- A reference to the newest version of the Standard library currently installed.

Device: CODESYS Control for Linux ARM64 SL (3S - Smart Software Solutions Gmb

PLC_PRG in: Structured Text (ST)

Messages - Total 0 error(s), 0 warning(s), 0 message(s)

Build | 0 error(s) | 0 warning(s) | 0 message(s)

Description

Loading libraries...: C:\Users\Administrator\Documents\Untitled3.project

Figure 10. Selecting the device and PLC_PRG

6.6.1.4 Adding EtherCAT MainDevice and SubDevice

To add the EtherCAT MainDevice and SubDevice, use the steps below.

1. Set the EtherCAT MainDevice:
   a. Right-click on the device -> **Add Device**.
   b. Select **EtherCAT Master**.
   c. Click **Add Device**.

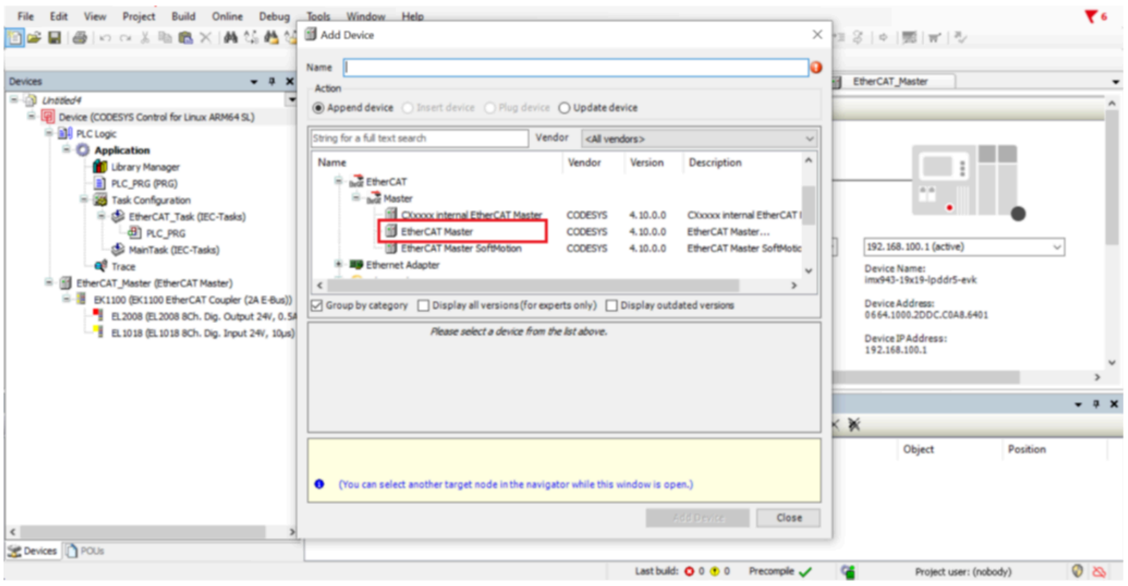

Add Device
Name
Action
Append device
Insert device
Plug device
Update device
Vendor <All vendors>
Name Vendor Version Description
EtherCAT
Master
CXxxxx internal EtherCAT Master CODESYS 4.10.0.0
EtherCAT Master CODESYS 4.10.0.0 EtherCAT Master...
EtherCAT Master SoftMotion CODESYS 4.10.0.0
Ethernet Adapter
Group by category
Display all versions (for experts only)
Display outdated versions
Please select a device from the list above.
(You can select another target node in the navigator while this window is open.)
Add Device
Close
Device (CODESYS Control for Linux ARM64 SL)
EtherCAT_Master (EtherCAT Master)
EK1100 (EK1100 EtherCAT Coupler (2A E-Bus))
EL2008 (EL2008 8Ch. Dig. Output 24V, 0.5A)
EL1018 (EL1018 8Ch. Dig. Input 24V, 10µs)
192.168.100.1 (active)
Device Name: imx943-19x19-lpddr5-evk
Device Address: 0664.1000.2DDC.C0A8.6401
Device IP Address: 192.168.100.1

Figure 11. Selecting EtherCAT MainDevice

2. Within the **Devices** window, right click **EtherCAT_Master (EtherCAT Master)**, click **Scan for Devices...** and then **Copy All Devices to Project**.

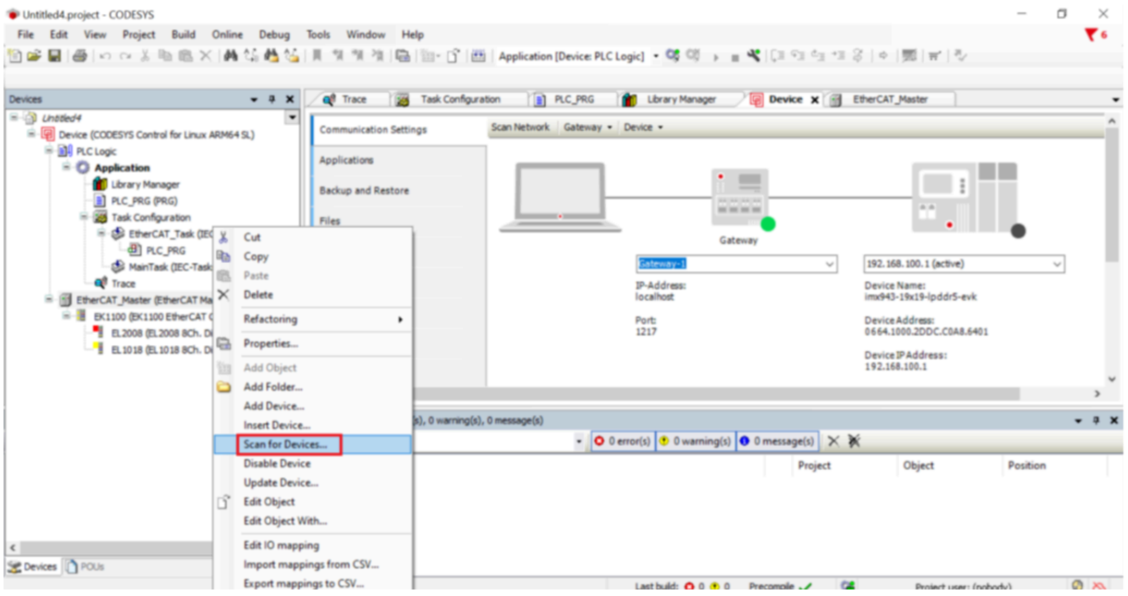


Figure 12. Scan for devices (1)

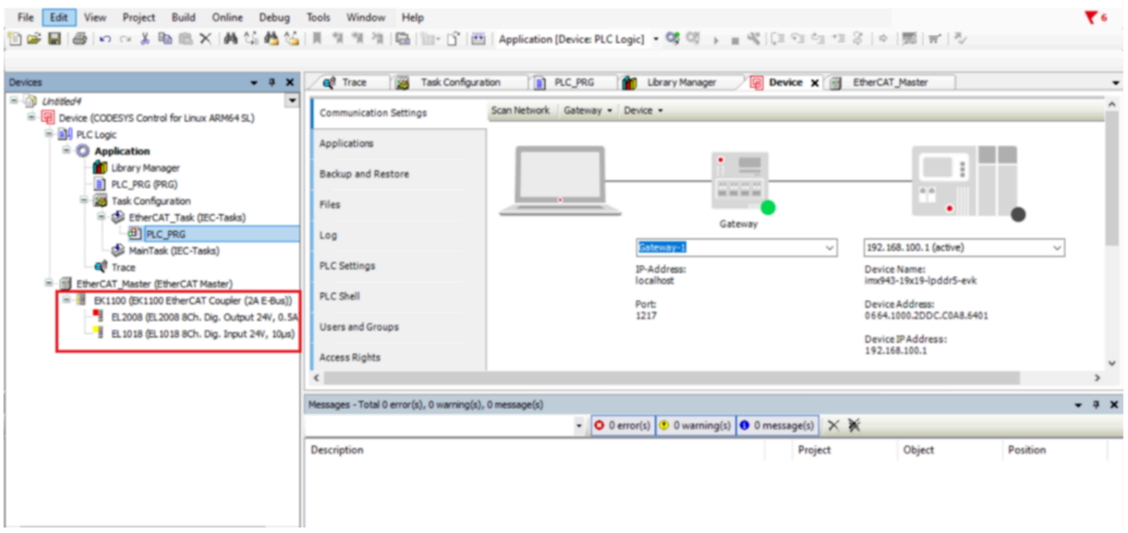

Devices
Untitled4
Device (CODESYS Control for Linux ARM64 SL)
PLC Logic
Application
Library Manager
PLC_PRG (PRG)
Task Configuration
EtherCAT_Task (IEC-Tasks)
PLC_PRG
MainTask (IEC-Tasks)
Trace
EtherCAT_Master (EtherCAT Master)
EK1100 (EK1100 EtherCAT Coupler (2A E-Bus))
EL2008 (EL2008 8Ch. Dig. Output 24V, 0.5A
EL1018 (EL1018 8Ch. Dig. Input 24V, 10µs)
Communication Settings
Applications
Backup and Restore
Files
Log
PLC Settings
PLC Shell
Users and Groups
Access Rights
Scan Network
Gateway
Device
Gateway-1
IP-Address: localhost
Port: 1217
192.168.100.1 (active)
Device Name: imx943-19x19-lpddr5-evk
Device Address: 0664.1000.2DDC.C0A8.6401
Device IP Address: 192.168.100.1
Messages - Total 0 error(s), 0 warning(s), 0 message(s)
Description
Project
Object
Position

Figure 13. Scan for devices (2)

Note:

If the EtherCAT devices do not appear under the list of scanned devices:

- *Try to power cycle the EtherCAT devices*
- *Try to refresh the boards*

6.6.2 Graphically view cycle time and jitter statistics in real-time

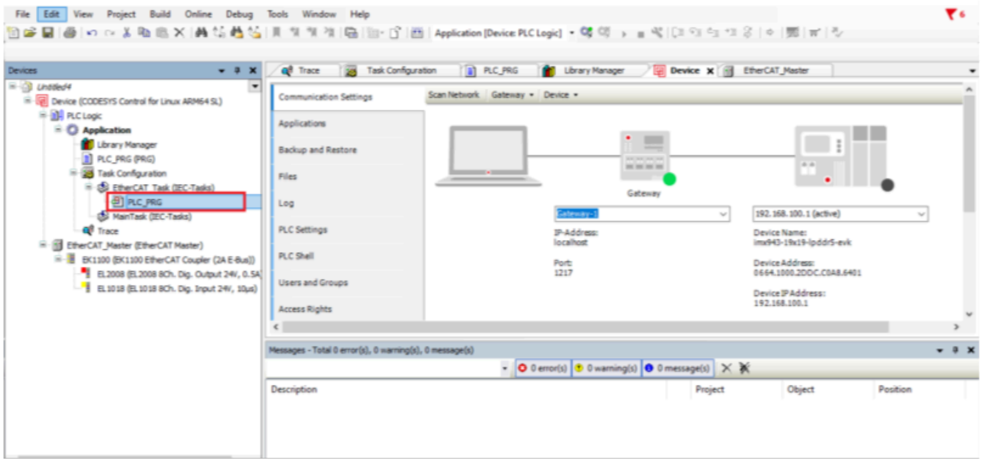

Devices
Untitled4
Device (CODESYS Control for Linux ARM64 SL)
PLC Logic
Application
Library Manager
PLC_PRG (PRG)
Task Configuration
EtherCAT_Task (IEC-Tasks)
PLC_PRG
MainTask (IEC-Tasks)
Trace
EtherCAT_Master (EtherCAT Master)
EK1100 (EK1100 EtherCAT Coupler (2A E-Bus))
EL2008 (EL2008 8Ch. Dig. Output 24V, 0.5A
EL1018 (EL1018 8Ch. Dig. Input 24V, 10µs)
Communication Settings
Scan Network
Gateway
Device
Gateway-1
192.168.100.1 (active)

Figure 14. Ethercat Task

To graphically view cycle time and jitter statistics in real-time, use the steps below:

1. Add the **CmplecTask** library in the library manager.
2. In your **PLC_PRG**, insert the following in the **Var** section:

```
PROGRAM PLC_PRG
  VAR
         tTask : Task_Info2; (* Task Info *)
         aIecTasks : ARRAY[1..20] OF Task_Info2; (* All Task Info *)
         hCurrentTask : RTS_IEC_HANDLE := SysTypes.RTS_INVALID_HANDLE;
         Result : RTS_IEC_RESULT; (* Result Code *)
```

```
          pTaskInfo : POINTER TO Task_Info2;
          hIecTask : RTS_IEC_HANDLE;
          pResult : POINTER TO RTS_IEC_RESULT;

   END_VAR
```

3. Add the following code in the **body** of the `PLC_PRG`:

```
...
(* Retrieve information about the current task *)
IF hCurrentTask = SysTypes.RTS_INVALID_HANDLE THEN
  hCurrentTask := IecTaskGetCurrent(pResult:=ADR(Result));
         pTaskInfo := IecTaskGetInfo3(hIecTask:=hCurrentTask,
      pResult:=ADR(Result));
END_IF
...
```

6.6.2.1 Adding Trace object to your CODESYS project

To add a Trace object to your CODESYS project, use the steps below:

1. Right click the **Application** on the left-hand panel.
2. Select **Add Object** > **Trace**.

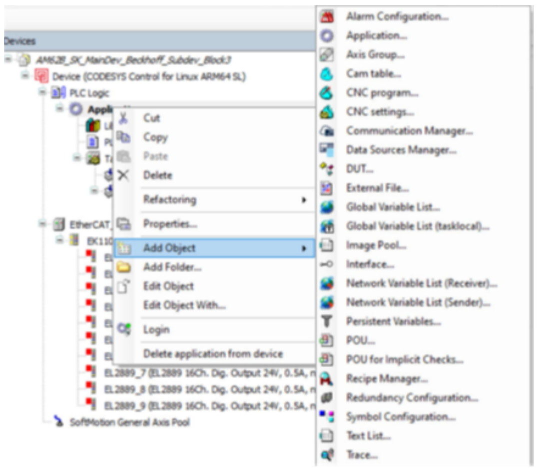
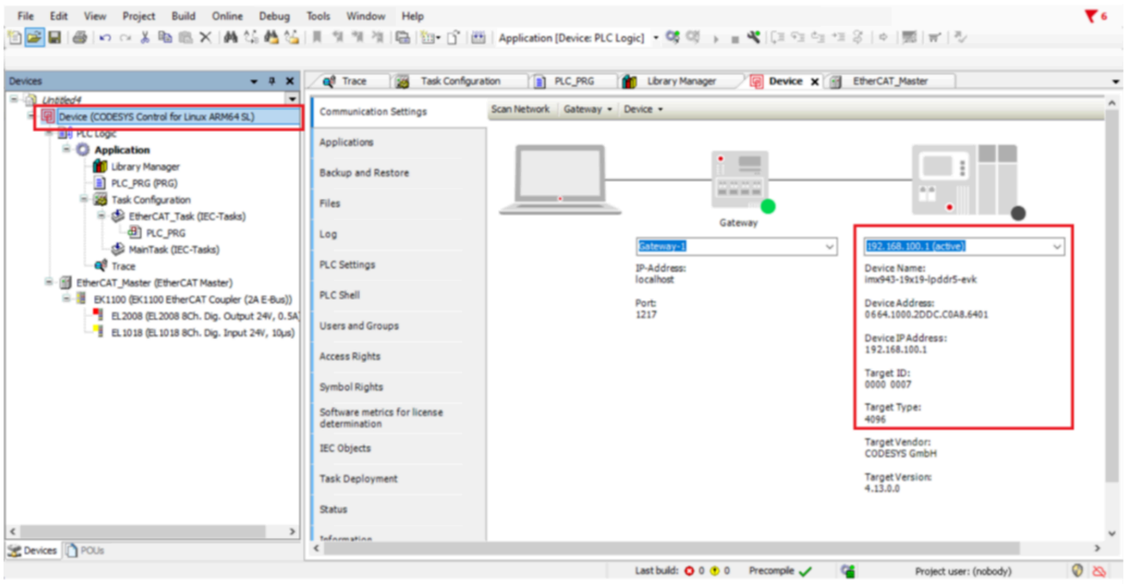


Figure 15. Add Object

3. Under **Task for Trace Recording**, select the task containing **PLC_PRG**.
4. Select **pTaskInfo->cycle time** and **pTaskInfo->jitter** to trace in the **Trace** object
5. Double click the **Trace** object under **Application** in the left-hand panel.
   a. Right click anywhere on the Trace object graph and select **Add Variable**.
   b. Next to the **Variable** entry, click the three dots.

6.6.2.2 Running CODESYS project

In the **Codesys_Control** window, fill in the IP address of the board in the respective textbox. Then, press **Enter** to connect to the board, as shown in Figure 16.



Gateway-1

IP-Address:
localhost

Port:
1217

192.168.100.1 (active)

Device Name:
imx943-19x19-lpddr5-evk

Device Address:
0664.1000.2DDC.C0A8.6401

Device IP Address:
192.168.100.1

Target ID:
0000 0007

Target Type:
4096

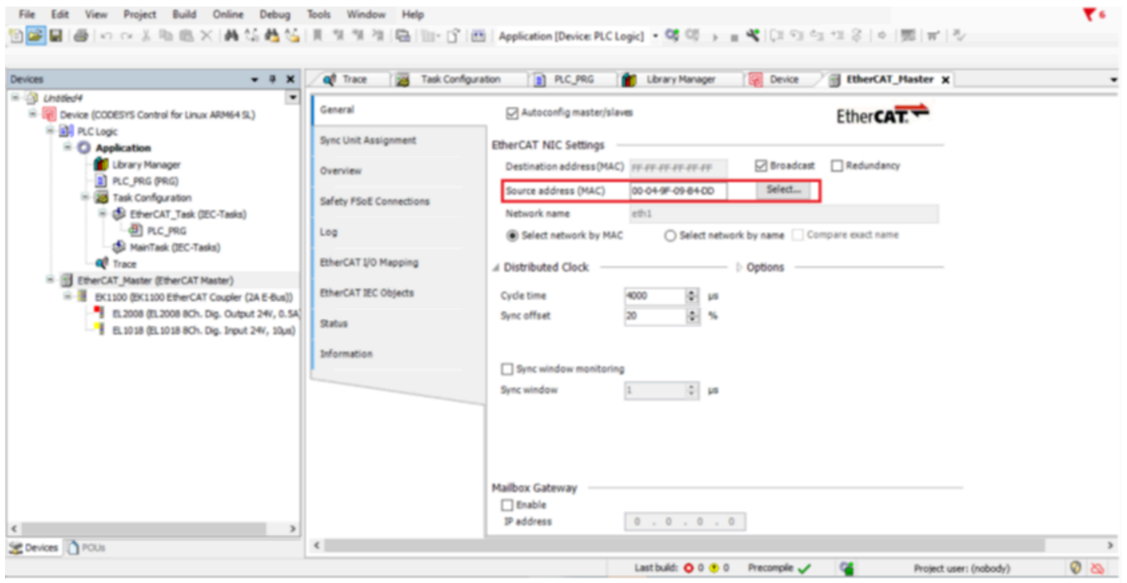
Target Vendor:
CODESYS GmbH

Target Version:
4.13.0.0

Figure 16. Fill IP Address

To run the CODESYS project, use the steps below:

1. Select the network port on the board that is used to communicate with the IO Device.



Autoconfig master/slaves

EtherCAT NIC Settings

Destination address (MAC) FF-FF-FF-FF-FF-FF   Broadcast [✓]   Redundancy [ ]

Source address (MAC) 00-04-9F-09-B4-DD   Select...

Network name eth1

[✓ Select network by MAC]   Select network by name   Compare exact name

Distributed Clock

Cycle time 4000 µs

Sync offset 20 %

Sync window monitoring [ ]

Sync window 1 µs

Mailbox Gateway

Enable [ ]

IP address 0 . 0 . 0 . 0

Figure 17. Select Network Port

2. Click the **Login** and **Start** buttons to run the PLC program:

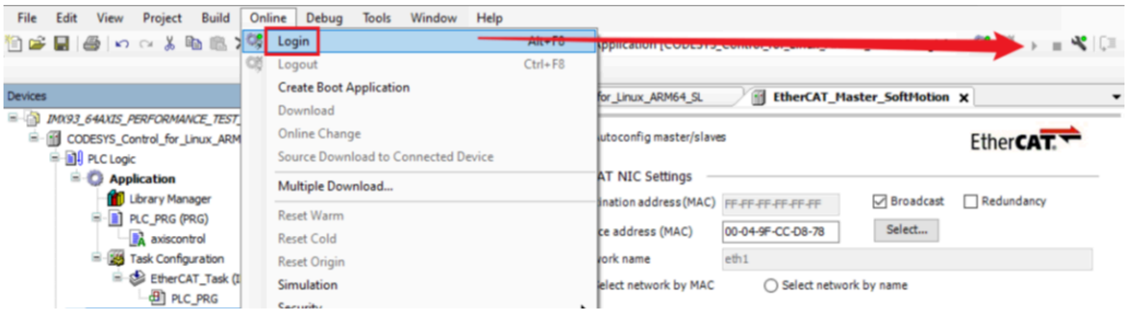
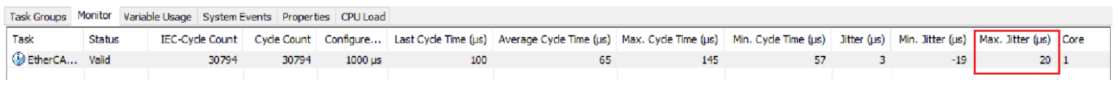


Figure 18. Click Login and Start

Maximum cycle time and maximum jitter are measured by CODESYS. Click **Task Configuration** -> **Monitor** to open the tab:



Task	Status	IEC-Cycle Count	Cycle Count	Configure...	Last Cycle Time (µs)	Average Cycle Time (µs)	Max. Cycle Time (µs)	Min. Cycle Time (µs)	Jitter (µs)	Min. Jitter (µs)	Max. Jitter (µs)	Core
EtherCA...	Valid	30794	30794	1000 µs	100	65	145	57	3	-19	20	1

Figure 19. Maximum jitter values

7 Summary

This application note describes the method for deploying and running CODESYS on i.MX series processors. It covers key technical aspects including system architecture design, software environment setup, Real-time performance optimization, and integration with EtherCAT Subdevices. The document helps developers quickly MainDevice the implementation path of soft PLCs on embedded platforms and provides technical guidance for building efficient and scalable industrial control systems.

Testing was performed using the CODESYS EtherCAT MainDevice Stack running on Real-time Edge Linux, with all key system-level optimizations applied. It includes CPU core isolation, performance governor mode, udhcpc disablement, and an optimized native Ethernet driver for low-latency packet handling.

Benchmark results on the i.MX 8M Mini, i.MX 943, and i.MX 95 platforms indicate that specific tasks and the size of the L2/L3 cache affects the overall performance of the CODESYS project. However, by Real-time edge plc image and optimizing the native driver, CPU usage and system jitter were reduced, thus improving overall performance.

8 Note about the source code in the document

The example code shown in this document has the following copyright and BSD-3-Clause license:

9 Revision history

Table 7 summarizes the revisions to this document.

Table 7. Revision history

Document ID	Release date	Description
AN14909 v.1.0	28 January 2026	Initial public release

Legal information

Tables

Figures

Contents

Date of release: 28 January 2026
Document identifier: AN14909