
# User Manual

# CC-Link IE TSN Master SDK

***port*** **GmbH**

**Regensburger Str. 7**

**D-06132 Halle/Saale**

## Disclaimer

This manual represents the current state of the product. Please check with port.de for the latest version as the document may have a newer version since errors may be corrected or changes for a newer version of the product may be incorporated. Port.de assumes no responsibility for errors in this document. Qualified feedback is appreciated at service@port.de.

This document is the Intellectual Property of port.de and is intended to be used with the described product only. It may be forwarded and/or copied in the original and unmodified format. All rights reserved.

The product enables to use technologies such as PROFINET, EtherNet/IP and/or EtherCAT and others. These technologies are promoted by trade organizations, such as PNO (profibus.org), ODVA (odva.org) or ETG (ethercat.org). These trade organizations as well maintain the specification and care about legal issues.
We strongly recommend to become a member of these organisations. Most technologies are making use of patented or otherwise copyrighted technologies, approaches or other intellectual property. The membership usually automatically entitles the member for use of most of the technology-inherent copyrighted or otherwise protected Intellectual Property of the corresponding trade organization and most 3rd parties. Otherwise the user will need to obtain licenses for many patented technologies separately.

Further we suggest to you to subscribe to the corresponding Conformance Test Tool of these trade organizations. For instance the ODVA only accepts conformance test applications from companies who have a valid membership and have a valid subscription to the recent Conformance Test Tool. We as port are members in all corresponding organizations and are holding a subscription to these tools - however you as a customer need to have an own membership and an own subscription to the tool.

**All rights reserved**

The programs, boards and documentations supplied by port GmbH are created with due diligence, checked carefully and tested on several applications.
Nevertheless, port GmbH cannot guarantee and nor assume liability that the program, the hardware board or the documentation are error-free or appropriate to serve a specific customer purpose. In particular performance characteristics and technical data given in this document may not be interpreted to be guaranteed product features in any legal sense.

For consequential damages, every legal responsibility or liability is excluded.
port has the right to modify the products described or their documentation at any time without prior warning, as long as these changes are made for reasons of reliability or technical improvement.
All rights of this documentation are with port. Unless expressively granted - the transfer of rights to third parties or duplication of this document in any form, whole or in part, is subject to written approval by port. Copies of this document may however be made exclusively for the use of the user and his engineers. The user is thereby responsible that third parties do not obtain access to these copies.
The soft- and hardware designations used are mostly registered and are subject to copyright.


Regensburger Straße 7
D-06132 Halle
Tel. +49 345 - 777 55 0
Fax. +49 345 - 777 55 20
E-Mail service@port.de
www.port.de
www.port-automation.com

# Contents

# List of Figures

# List of Tables

# Changelog

| Version | Changes |
|---|---|
| 1.0 | Initial Release |
| 1.1 | Chapter 2.7<br>- Added functions goal_ccIIeTsnNmtUpload, goal_ccIIeTsnNmtDownload, goal_ccIIeTsnSlaveProcTypeRead<br>Chapter 2.8<br>- Added callbacks GOAL_CCL_CB_IP_ADDR_DUPL and GOAL_CCL_CB_READPROCTYPE_RES<br>Chapter 2.10<br>- Added callbacks GOAL_CCL_CO_CB_NMT_UPL_RES and GOAL_CCL_CO_CB_NMT_DNL_RES<br>Chapter 3.1.5<br>- Updated description for OpenIL 1.9<br>- Fixed misspellings<br>Chapter 4<br>- Added new application settings |

# 1 Introduction

The CC-Link IE TSN Master SDK is used to implement a Management Master station or a Control Master Station in a device. The SDK uses GOAL, port's Industrial Communication Framework. The SDK has the following directory structure.

| Path | Description |
|---|---|
| appl/goal_ccl_ie_tsn/01_master | Sample application for communication with port's Slave SDK |
| appl/goal_ccl_ie_tsn/03_master_ct | Sample application for Conformance Test |
| goal* | GOAL core, platform independent |
| plat | Platform specific files: architecture, board configuration, drivers |
| projects/goal_ccl_ie_tsn/01_master | Sample project for communication with port's Slave SDK |
| projects/goal_ccl_ie_tsn/03_master_ct | Sample project for Conformance Test |
| protos/ccl_ie_tsn | CC-Link IE TSN protocol stack |
| protos/goal_ts | (g)PTP protocol stack |
| protos/slmp | SLMP protocol stack |

Table 1.1: Directory structure of the Master SDK

# 2 Writing an Application for the Master SDK

## 2.1 Overview

An application for the Master SDK is a GOAL application. It consists of three functions that are called by GOAL: appl_init, appl_setup, appl_loop.
Additionally, the application can register a callback that is called by the CC-Link IE TSN Master stack to inform the application about events.

The function appl_init is used to register components in GOAL, e.g. the CC-Link IE TSN protocol stack.
The actual initialization of the application happens in appl_setup.

The application must include goal_includes.h and goal_ccl_ie_tsn_master.h.

## 2.2 Configuration of the CC-Link IE TSN protocol stack

The function appl_setup is called by GOAL during initialization. Within this function all functions listed in this chapter can be used to configure the bahavior of the Management Master. All Functions must be called before calling *goal_cclIeTsnNew*.

All functions return a status code indicating whether the operation succeeded or not.

| Function | Description |
|---|---|
| goal_cclIeTsnCfgManagementPrioritySet | Set the Management Priority of this master station |
| goal_cclIeTsnCfgNumGmRecordEntriesSet | Set the maximum number of Grandmaster Record entries |
| goal_cclIeTsnCfgNumSlavesSet | Set the maximum number of slaves handled by this station |
| goal_cclIeTsnCfgNumS2sSubpayloads | Set the maximum number of Subpayloads for Slave-2-Slave communication |
| goal_cclIeTsnCfgSlaveCheckIntervalSet | Set the check interval for pending slaves during RUN phase |
| goal_cclIeTsnCfgNodeTypeSet | Set the node Type of this station (Management Master or Control Master) |
| goal_cclIeTsnCfgCertificationClassSet | Set the Certification Class of this station |
| goal_cclIeTsnCfgDeviceVersionSet | Set the Device Version of this station |
| goal_cclIeTsnCfgDeviceVendorCodeSet | Set the Device Vendor Code of this station |
| goal_cclIeTsnCfgDeviceProductIdSet | Set the Product Id of this station |
| goal_cclIeTsnCfgDeviceExModelCodeSet | Set the Device Expansion Model Code of this station |
| goal_cclIeTsnCfgDeviceTypeIdSet | Set the Device Type Id of this station |
| goal_cclIeTsnCfglogSyncIntSet | Set the default logarithmic Sync Tx interval |
| goal_cclIeTsnCfglogAnnounceIntSet | Set the default logarithmic Announce Tx interval |
| goal_cclIeTsnCfglogPDelayIntSet | Set the default logarithmic PDelay Tx interval |

| | |
|---|---|
| goal_cclIeTsnCfgPdelayResTimeSet | Set the time between Pdelay_Req and Pdelay_Resp_Follow_Up |
| goal_cclIeTsnCfgDelaySetTimeSet | Set the time between Peer delay calculation and port role adjustment |
| goal_cclIeTsnCfgAnounceRelayTimeSet | Set the time for relaying Announce frames from Slave port to Master ports |
| goal_cclIeTsnCfgNumTxSubpayloadEntrieSet | Set the number of allowed Tx Subpayload information entries. The number defines how many Transmit Subpayloads can be handled by this station. |
| goal_cclIeTsnCfgNumRxSubpayloadEntrieSet | Set the number of allowed Rx Subpayload information entries. The number defines how many Receive Subpayloads can be handled by this station. |
| goal_cclIeTsnCfgNumCycTxHandlersSet | Set the number of cyclic transmission handlers. The number defines how many cyclic connections can be established at the same time. |
| goal_cclIeTsnCfgNumCycRxHandlersSet | Set the number of cyclic reception handlers. The number defines how many cyclic connections can be established at the same time. |
| goal_cclIeTsnCfgNumSlmpServerHandlesSet | Define how many received SLMP requests can be processed in parallel |
| goal_cclIeTsnCfgNumSlmpClientHandlesSet | Defines how many SLMP requests can be sent in parallel. |
| goal_cclIeTsnCfgNumSlmpDivDataHandlesSet | Define how many fragmented SLMP messages can be received in parallel. |
| goal_cclIeTsnCfgClearOnHoldEnable | Defines whether imported cyclic data is cleared or held if the the sender's application is stopped. |
| goal_cclIeTsnCfgPtpPrio1Set | Overwrite the PTP prio1 value for this station. |
| goal_cclIeTsnCfgLinkSpeedEnforce | Enforce a link speed for all ports |

Table 2.1: Configuration Functions of the CC-Link IE TSN protocol stack

## 2.3 Creating a new instance of the CC-Link IE TSN protocol stack

After the stack has been configured the function *goal_cclIeTsnNew* must be invoked to create a new instance of the protocol stack. It is also used to register a callback handler for processing events from the stack. The callback handler will be explained in detail in a later chapter.

```
    GOAL_STATUS_T res;                              /* result */
    static GOAL_CCL_HANDLE_T *pCcl = NULL;          /**< GOAL CCL handle */

    /* create instance of CC-Link IE TSN stack */
    res = goal_cclIeTsnNew(&pCcl, GOAL_CCL_INSTANCE_DEFAULT, appl_goalCclCb);
    if (GOAL_RES_ERR(res)) {
        goal_logErr("Failed to instantiate CC-Link IE TSN stack");
        return res;
    }
```

Figure 2.1: Creating a new instance of the CC-Link IE TSN master stack

This function creates a handle (in this example it is called pCcl) that must be used for all other function calls to reference the the stack instance.

## 2.4 Runtime behavior of the CC-Link IE TSN Master stack

The functions in this chapter influence the runtime behaviour of the CC-Link IE TSN Master Station. The functions must be called after *goal_ccIeTsnNew* returned successfully and before *goal_ccIeTsnStart* is called. These functions can be directly called within appl_setup or at a later point in time. These functions represent settings that usually come from an Engineering Tool. Therefore, the settings can be applied after receiving the current configuration from the tool.

| Function | Description |
|---|---|
| goal_ccIeTsnDetectionAckEnforce | Enforce DetecktionAck transmission with every Detection frame |
| goal_ccIeTsnNetworkConfigSet | set Network properties:<br>- cycle time<br>- CC-Link IE Field Coexistence<br>- number of Generic PTP devices<br>- TxProhibit time<br>- timeslot number for cyclic communication<br>- number of allowed consecutive cyclic errors |
| goal_ccIeTimeSlotAdd | Add a time slot in ascending order from TSLT0 to TSLT7. |
| goal_ccIeTimeSlotEtherTypeAdd | Assign an Ethertype to a timeslot in ascending order from TSLT1 to TSLT7. |
| goal_ccIeTimeSlotMacAddrAdd | Assign a Destination MAC address to a specific timeslot. |
| goal_ccIeTimeSlotVlanAdd | Assign a VLAN Tag to a specific timeslot. |
| goal_ccIeTsnPortFilterSet | Set a port filter for each port. Use the GOAL_CCL_PORT_FILTER_* macros. |
| goal_ccIeTsnMulticastGroupAdd | Register a Multicast Group for reception. |
| goal_ccIeTsnRxAddrOverlapCheckEnable | Request Remote Stations to check their Rx Memory configurations for overlaps. |
| goal_ccIeTsnCycleCounterIgnoreEnable | Instruct all slaves to ignore the cycle counter in cyclic frames. |
| goal_ccIeTsnTimeoutsGet | Get the timeout values of the Management Master state machine. |
| goal_ccIeTsnTimeoutsSet | Set the timeout values of the Management Master state machine. |

| | |
|---|---|
| goal_cclIeTsnTimeSyncSet | Set the time synchronization settings:<br>- use 802.1AS or 1588v2<br>- domain number<br>- logarithmic Sync interval<br>- Sync Rx timeout factor<br>- logarithmic Announce interval<br>- announce Rx timeout<br>- logarithmic Pdelay interval<br>- DelayResp monitoring interval<br>- delay mechanism (E2E or P2P)<br>- number of tolerable Sync losses<br>- synchronization tolerance |
| goal_cclIeTsnNetworkNumberSet | Set the SLMP network number |
| goal_cclIeTsnMasterIdSet | Set the Id of the master station |
| goal_cclIeTsnCanOpenCallbackSet | Register a CANopen callback handler |
| goal_cclIeTsnStationNumSet | Set the station number of the device. |
| goal_cclIeTsnStationModeAdd | Register a Station mode for this device |
| goal_cclIeTsnLinkDevAdd | Add a Link device to a Station Mode |

Table 2.2: Runtime behavior functions of the CC-Link IE TSN Master stack

## 2.5 Slave Configuration

For each slave station in the network a slave handle must be added. Each slave is represented by a Slave Id. The functions must be called after *goal_cclIeTsnNew* returned successfully and before *goal_cclIeTsnStart* is called. These functions can be directly called within appl_setup or at a later point in time, i.e. after the expected slave configuration has been received from the Engineering Tool.

| **Function** | **Description** |
|---|---|
| goal_cclIeTsnSlaveAdd | Add a Slave station with an IPv4 address and the ID of the Controll Master.<br>The function returns the Slave ID (used as internal reference). |
| goal_cclIeTsnControlMasterAdd | Add a Control Master station with an IPv4 address and ist Master ID.<br>The function returns the station handle ID (used as internal reference). |
| goal_cclIeTsnSlaveStationModeSet | Set the expected station mode of a slave. |
| goal_cclIeTsnSlaveTsltMagnificationSet | Set the timeslot magnification value for a slave. |
| goal_cclIeTsnSlavePortFilterSet | Set the port filter for a slave. Each array entry represents a port. Use the GOAL_CCL_PORT_FILTER_* macros. |

| | |
|---|---|
| goal_ccIIeTsnSlaveCyclicConfigSet | Set the cyclic configuration for a slave:<br>- if slave is a reserved station<br>- if Control data can be split in multiple frames<br>- EMG groups slave belongs to<br>- GOF groups slave belongs to<br>- number of sub cycles<br>- subCycle where frame is either sent or received<br>- if cyclic frames must be sent from all ports |
| goal_ccIIeTsnSlaveInputAdd | This function is used to register an Input Link device for a slave, i.e. a Subpayload transmitted by the slave to the master.<br>It requires the receive address in the master's memory, the transmitt address in the slave's Link Device and the data length. |
| goal_ccIIeTsnSlaveOutputAdd | This function is used to register an Output Link device for a slave, i.e. a Subpayload received by the slave from the master.<br>It requires the transmit address in the master's memory, the receive address in the slave's Link Device and the data length. |
| goal_ccIIeTsnSlaveS2sTxSplAdd | Add a Tx Subpayload for Slave-to-Slave communication to a slave. The destination MAC address and IP address of the receiving slave must be specified. Furthermore the Subcycle settings must be specified. |
| goal_ccIIeTsnSlaveS2sRxSplAdd | Add a Rx Subpayload for Slave-to-Slave communication to a slave. |
| goal_ccIIeTsnSlaveS2sRxSrcAdd | Add a Rx Source information item for Slave-to-Slave communication to a slave. This function sets the IP address of the sending slave and the subcycle settings. |
| goal_ccIIeTsnSlaveRxMulticastGroupSet | Add a Slave to a Rx multicast group. The slave's Output Subpayloads will be sent with other Subpayloads via Multicast frames. |
| goal_ccIIeTsnSlaveTxMulticastGroupSet | Add a Slave to a Tx multicast group. The slave's Input Subpayloads will be received with other Subpayloads via Multicast frames. |
| goal_ccIIeSlaveTimeSlotMacAddrAdd | Assign a Destination MAC address to a timeslot for this slave. |
| goal_ccIIeSlaveTimeSlotVlanAdd | Assign a VLAN to a timeslot for this slave. |
| goal_ccIIeTsnSlaveUncontrolledMulticastGroupAdd | Add a slave to a Multicast Group that is not controlled by this station, i.e. multicast group for slave-to-slave communication |
| goal_ccIIeTsnSlaveAppNetSyncEnable | Request the slave to synchronize its application to the network cycle. |

| | |
|---|---|
| goal_cclIeTsnSlaveCyclicStopEnable | Enable or disable the Cyclic Stop request for a slave station. |
| goal_cclIeTsnSlaveCanOpenConfigSet | Configure a Slave for CANopen communication. Register startup commands, and TPDO and RPDO config objects. |

Table 2.3: Functions for Slave configuration

## 2.6 Starting the CC-Link IE TSN Master Stack

After finishing all settings of the runtime behavior and the expected slave configuration the stack must be started by calling *goal_cclIeTsnStart*.

```
res = goal_cclIeTsnStart(pCcl);
if (GOAL_RES_ERR(res)) {
    goal_logErr("Failed to start stack");
}
```

Figure 2.2: Start an instance of the CC-LINK IE TSN master stack

If this function succeeds the stack CC-Link IE TSN Master stack has been started and tries to detect all configured slaves. All functions described in previous chapters cannot be used anymore.

## 2.7 Functions used during Run phase

After sucessfulling starting the stack. The application can use the following functions to access cyclic data and to influence the stack's behaviour.

| Function | Description |
|---|---|
| goal_cclIeTsnCyclicStopSet | Enable or Disable Cyclic Stop for the Master Station. |
| goal_cclIeTsnInputGet | Read Input data from the Cyclic Memory Map.<br>Only Input Link devices can be used. Each Link Device has ist own memory map. |
| goal_cclIeTsnOutputSet | Write Output data to the Cyclic Memory Map<br>Only Output Link devices can be used. Each Link Device has its own memory map. |
| goal_cclIeTsnAppStopSet | Enable or Disable Application Stop mode for this station. |
| goal_cclIeTsnAppErrorStopSet | Enable or Disable Application Error Stop mode for this station. |
| goal_cclIeTsnEmergencyStopExec | Issue an Emergency Stop due to internal error.<br>Calling this function will cause all controlled devices in the network to shutdown.<br>After calling this function the application is expected to halt the device and stop processing cyclic data. |

| | |
|---|---|
| goal_ccIIeTsnPowerSupplyErrorStopExec | Issue an Emergency Stop due to Power supply error.<br>Calling this function will cause all controlled devices in the network to shutdown.<br>After calling this function the application is expected to halt the device and stop processing cyclic data. |
| goal_ccIIeTsnOutputDevGet | Get data from an Output Link Device (data received via Master-to-Master communication). |
| goal_ccIIeTsnInputDevSet | Set data of an Input Link Device (data transmitted for Master-to-Master communication). |
| goal_ccIIeTsnSdoWrite | Start a SDO Write operation for a slave. |
| goal_ccIIeTsnSdoRead | Start a SDO Read operation for a slave. |
| goal_ccIIeTsnNmtUpload | Get the NMT state of a CANopen Slave. |
| goal_ccIIeTsnNmtDownload | Set the NMT state of a CANopen Slave. |
| goal_ccIIeTsnSlaveProcTypeRead | Request processor Type information from a slave |

Table 2.4: Functions used during Run phase

## 2.8 Application Callback

During initialization the application can register a callback handler with the function *goal_ccIIeTsnNew*.

The callback handler uses the following arguments:

| Argument data type | Description |
|---|---|
| GOAL_CCL_HANDLE_T * | CC-Link IE TSN stack instance reference |
| GOAL_CCL_CB_ID_T | callback ID indicating callback type |
| GOAL_CCL_CD_DATA_T * | callback data, actual meaning depends on callback ID |

Table 2.5: Arguements of the Application Callback Handler

Some callback ID also evaluate the return value of the handler to decide how to proceed.

| Callback ID | Description | Callback data | Return value |
|---|---|---|---|
| GOAL_CCL_CB_DETECTION_MISMATCH | mismatch in detected slaves and configuration from engineering tool | NULL | don't care |
| GOAL_CCL_CB_NETCFG_MISMATCH | mismatch in network configuration | pNetConfigMismatchId (Id of Slave with mismatching network configuration) | GOAL_OK: continue operation<br>other: go to Error state |

| | | | |
|---|---|---|---|
| GOAL_CCL_CB_MASTERCFG_MISMATCH | mismatch in configuration of a Control Master | NULL | don't care |
| GOAL_CCL_CB_ERROR_STATE | device entered Error state | NULL | don't care |
| GOAL_CCL_CB_UNKNOWN_SLAVE | detected a slave that was not set by the engineering tool | pUnknownSlaveIpAddr (IP Address of unknown Slave) | don't care |
| GOAL_CCL_CB_SLAVE_UNDETECTED | slave was not detected in the network | pUndetectedSlaveId (ID of Slave Handle) | GOAL_OK: continue operation other: go to Error state |
| GOAL_CCL_CB_SLAVE_WRONG_IP_ADDR | slave has wrong IP address (not in the same subnet) | pWrongIpAddrSlaveId (ID of Slave Handle) | GOAL_OK: continue operation other: go to Error state |
| GOAL_CCL_CB_SLAVE_IP_ADDR_DUPL | slave has a duplicate IP address | pDuplicateIpAddrSlaveId (ID of Slave Handle) | GOAL_OK: continue operation other: go to Error state |
| GOAL_CCL_CB_SLMP_ERROR | SLMP error received from station | pSlmpErrorInfo->slaveId (ID of Slave Handle) pSlmpErrorInfo->cmd (SLMP command) pSlmpErrorInfo->subCmd (SLMP subcommand) pSlmpErrorInfo->endCode (end code indicating error) | don't care |
| GOAL_CCL_CB_CM_UNCONFIGURED | Control Master is not configured | pUnconfiguredMasterId (ID of Control Master handle) | GOAL_OK: continue operation other: go to Error state |
| GOAL_CCL_CB_SLAVE_UNCONTROLLED | Slave not controlled by its Control Master | pUncontrolledSlaveId (ID of Slave Handle) | GOAL_OK: continue operation other: go to Error state |

| | | | |
|---|---|---|---|
| GOAL_CCL_CB_WRONG_GRANDMASTER | Slave has not the chosen Grandmaster | pWrongGrandmasterSlaveId (ID of Slave Handle) | don't care |
| GOAL_CCL_CB_RSV_TRANSIENT_DONE | reserved transient transmission done | NULL | don't care |
| GOAL_CCL_CB_RESERVED_STATION_ON | device entered Reserved Station mode | NULL | don't care |
| GOAL_CCL_CB_RESERVED_STATION_OFF | device left Reserved Station mode | NULL | don't care |
| GOAL_CCL_CB_CYCLIC_STOP_ON | cyclic communication stopped | NULL | don't care |
| GOAL_CCL_CB_CYCLIC_STOP_OFF | cyclic communication restarted | NULL | don't care |
| GOAL_CCL_CB_OWN_STATION_EMG_STOP | device received Emergency Stop request | pEmgGroup (EMG group causing stop) | don't care |
| GOAL_CCL_CB_CYCLIC_ERROR_ON | other station causes cyclic error | pCyclicErrIpAddr (IP address of Slave) | don't care |
| GOAL_CCL_CB_CYCLIC_ERROR_OFF | other station fixed cyclic error | pCyclicErrIpAddr (IP address of Slave) | don't care |
| GOAL_CCL_CB_OTHER_STATION_EMG_STOP | Emergency stop for an Emergency Group | pEmgGroup (EMG group that will be stopped) | don't care |
| GOAL_CCL_CB_OTHER_STATION_GOF_STOP | Emergency stop for a GOF Group | pGofGroup (GOF group that will be stopped) | don't care |
| GOAL_CCL_CB_SLAVE_INVALID_CFG | expected configuration of slave is not valid | pInvalidCfgSlaveId (ID of Slave Handle) | GOAL_OK: continue operation other: go to Error state |
| GOAL_CCL_CB_SLAVE_INVALID_DATA_ON | received invalid control data from slave | pInvalidCycDataSlaveId (ID of Slave Handle) | don't care |
| GOAL_CCL_CB_SLAVE_INVALID_DATA_OFF | control data from slave is valid again | pInvalidCycDataSlaveId (ID of Slave Handle) | don't care |
| GOAL_CCL_CB_CYC_COM_ENABLED | station started to send and receive process data | NULL | don't care |
| GOAL_CCL_CB_CYC_COM_DISABLED | station stopped to send and receive process data | NULL | don't care |
| GOAL_CCL_CB_IP_ADDR_DUPL | station's IP address is also used by another station | NULL | don't care |
| GOAL_CCL_CB_READPROCTYPE_RES | received a ReadProcType response | pReadProcTypeRes (ReadProcType response data) | don't care |

Table 2.6: Handling of Application Callback IDs

```
static GOAL_STATUS_T appl_goalCclCb(
    GOAL_CCL_HANDLE_T *pCclm,                   /**< GOAL CCL handle */
    GOAL_CCL_CB_ID_T cbId,                      /**< callback ID */
    GOAL_CCL_CD_DATA_T *pCbData                 /**< callback data */
)
{
    GOAL_STATUS_T res = GOAL_OK;                /* result */

    switch (cbId) {
        /* ... */

        case GOAL_CCL_CB_SLAVE_WRONG_IP_ADDR:
            goal_logInfo("slave 0x%04x has unexpected IP address",
                         *(pCbData->pWrongIpAddrSlaveId));
            /* abort initialization */
            res = GOAL_ERROR;
            break;

        /* ... */
    }

    return res;
}
```

Figure 2.3: Sample implementation of the application callback

## 2.9 Configuration of CANopen Slaves

CANopen slaves are configured with the same functions as Link Device slaves. Additionally, there is the function *goal_cclIeTsnSlaveCanOpenConfigSet* to set CANopen specific properties of the slave.

```
static uint16_t slave2RpdoMapObj = 0x1601;      /**< Slave2: RPDO Mapping object */
static uint16_t slave2TpdoMapObj = 0x1A01;      /**< Slave2: TPDO Mapping object */
static GOAL_CCL_CO_STARTUP_T slave2StartUpCmds[] = {
    {GOAL_TRUE, 0x1C00, 1, sizeof(uint16_t), (uint8_t *) &slave2RpdoMapObj},
    {GOAL_TRUE, 0x1C01, 1, sizeof(uint16_t), (uint8_t *) &slave2TpdoMapObj},
};
static uint16_t slave2TpdoCfgTbl[] = {
    0x1C01,
};
static uint16_t slave2RpdoCfgTbl[] = {
    0x1C00,
};

res = goal_cclIeTsnSlaveCanOpenConfigSet(pCcl, slaveId,
                                         slave2StartUpCmds, ARRAY_ELEMENTS(slave2StartUpCmds),
                                         slave2TpdoCfgTbl, ARRAY_ELEMENTS(slave2TpdoCfgTbl),
                                         slave2RpdoCfgTbl, ARRAY_ELEMENTS(slave2RpdoCfgTbl));
if (GOAL_RES_ERR(res)) {
    goal_logErr("Failed to configure CANopen properties of slave %u", slaveId);
}
```

Figure 2.4: Configuration of CANopen slave

An entry in the startup command table has the following fields:

```
typedef struct {
    GOAL_BOOL_T wrFlag;                 /**< write or read access */
    uint16_t index;                     /**< object index */
    uint16_t subIndex;                  /**< object subindex */
    uint16_t dataLen;                   /**< size of data to be written or read */
    uint8_t *pData;                     /**< write data buffer */
} GOAL_CCL_CO_STARTUP_T;
```

Figure 2.5: CANopen startup command table entry

The other two tables contain lists of TPDO Config Objects and RPDO Config objects that must be enabled for cyclic communication.
In the RUN phase the application can use the functions goal_cclIeTsnSdoWrite and goal_cclIeTsnSdoRead to start SDO write or read operations.

## 2.10 CANopen Data Callback

The application can use the function *goal_cclIeTsnCanOpenCallbackSet* to register a callback handler for CANopen object access.

```
        res = goal_cclIeTsnCanOpenCallbackSet(pCcl, appl_goalCclCanOpenCb);
        if (GOAL_RES_ERR(res)) {
            goal_logErr("Failed to register CANopen callback");
        }
```

Figure 2.6: Register a CANopen callback handler

The handler is called by the protocol stack every time a SDO Read or Write response was received, including those of the startup commands.
The callback handler uses the callback IDs GOAL_CCL_CO_CB_SDO_READ_RES or GOAL_CCL_CO_CB_SDO_WRITE_RES to indicate if the callback data refers to a read response or a write response.
The callback IDs GOAL_CCL_CO_CB_NMT_UPL_RES and GOAL_CCL_CO_CB_NMT_DNL_RES are used to indicate results of an NMT Upload or Download request.

| Member | Data type | Description |
|---|---|---|
| slaveId | uint16_t | slave id |
| endCode | uint16_t | SLMP end code (status of operation) |
| index | uint16_t | object index (only valid during startup or if endCode == 0x0000) |
| subIndex | uint8_t | object subindex (only valid during startup or if endCode == 0x0000) |
| dataLen | uint16_t | object data length (only valid for read access and if endCode == 0x0000) |
| pData | uint8_t * | object data (only valid for read access and if endCode == 0x0000) or current NMT state of responder (as a uint8_t variable) |

Table 2.7: Members of callback data of CANopen data callback

# 3 Supported Platforms

Since the CC-Link IE TSN Protocol stack runs on GOAL It can run on any platform supported by GOAL. However in order to fullfill the requirements of a Class B device special hardware support is needed.
The hardware must support timestamping of Ethernet frames as defined by IEEE 1588v2 or IEEE 802.1AS.
Additionally it must support time aware queuing of Ethernet framesas defined by IEEE 802.1Qbv.

## 3.1 NXP LS1028A

The NXP LS1028A is a SoC that fullfills the hardware requirements for a Class B device.
There is an evaluation board called LS1028ARDB.

### 3.1.1 Building the firmware

The LS1028ARDB uses OpenIL, a Linux distribution for industrial automation with Realtime support. The distribution can be built with Buildroot.

- git clone https://github.com/openil/openil.git
- cd openil
- git checkout OpenIL-v1.9-202009
- make nxp_ls1028ardb-64b_defconfig
- make 2>&1 | tee build.log

### 3.1.2 Flashing the firmware

Once the firmware has been built it must be copied to the SD-Card of the LS1028ARDB.

- insert SD-Card into Linux PC
  - card is listed as /dev/sdX [e.g. /dev/sdc]
  - see log via dmesg
- in directory openil:
  - sudo dd if=output/images/sdcard.img of=/dev/sdc bs=1024
  - **ATTENTION: choosing the wrong output device will overwrite sections of the PC's HDD/SDD causing data loss**
- insert SD-Card into LS1028ARD and start device

### 3.1.3 Debug Interface

A serial console is available on UART1.
Use the following parameters: 115,200 baud/s, 8 data bits, no parity, 1 stop bit.

### 3.1.4 Configuration of the LS1028ARDB

It is possible to configure the number of ports used for CC-Link IE TSN. By default each port of the integrated TSN Switch is independent, i.e. there is no forwarding between these ports.
The interfaces representing these ports are called swp0 to swp3. In order to enable forwarding between two or more ports the following script could be executed:

```
#!/bin/sh

# create a bridge device
ip link add name switch type bridge
ip link set switch up
# add 1st port
ip link set swp0 master switch
ip link set swp0 up
# add 2nd port
ip link set swp1 master switch
ip link set swp1 up
# uncomment to add 3rd port
#ip link set swp2 master switch
#ip link set swp2 up
# uncomment to add 4th port
#ip link set swp3 master switch
#ip link set swp3 up

# add a route for this interface (subnet address might need to adjusted)
ip route add 192.168.3.0/24 dev switch
```

This will create a new interface called “switch”.

By default the OpenIL image runs netopeer2, a NETCONF server. This server interfers with the Realtime behaviour of the GOAL process. Therefore the server must be removed from the initialization scripts:

```
rm /etc/init.d/S91netopeer2-server
```

### 3.1.5 Building the CC-Link IE TSN Master Application

The application can be built with aarch64-linux-gnu-gcc.
Navigate to the project folder of a sample application, e.g. “*projects/goal_ccl_ie_tsn/03_master_ct/gcc*”.

- Select the target platform (only required once)
  - make select
  - enter the number of “linux_nxp_ls1028a”
- build sample application
  - make

The binary is located at
“*projects/goal_ccl_ie_tsn/03_master_ct/gcc*/linux_nxp_ls1028a/goal_ linux_nxp_ls1028a.bin”.
The file must be copied to the LS1028ARDB, e.g. via scp.

On the LS1028ARDB:

- reduce kernel log messages to only critical ones
  - `echo 1 > /proc/sys/kernel/printk`
- disable memory overcommitting
  - `echo 2 > /proc/sys/vm/overcommit_memory`
- make application ecutable
  - `chmod +x goal_ linux_nxp_ls1028a.bin`
- start application (adjust Ethernet port if necessary)
  - `./goal_ linux_nxp_ls1028a.bin -i <IFACE>`

**Note:**
<IFACE> is the ethernet interface that GOAL should use. This is either the bridge device “switch” or a standalone port, e.g. “swp0”.

### 3.1.6 Start CC-Link IE TSN Master application automatically

In order to start the Master SDK automatically after boot up, a script must be created in the directory */etc/init.d*, e.g. S99GOAL.

The script should have the following content:

```
#!/bin/sh
#
# CC-Link IE TSN Master SDK
#
GOAL=/root/goal_linux_nxp_ls1028a.bin
start() {
       echo 1 > /proc/sys/kernel/printk
       echo 2 > /proc/sys/vm/overcommit_memory
       printf "Starting CC-Link IE TSN Master SDK: "
       ${GOAL} -i swp0 &
       echo "OK"
}
stop() {
       printf "Stopping CC-Link IE TSN Master SDK: "
       killall $(basename ${GOAL})
       echo "OK"
}
restart() {
       stop
       start
}

case "$1" in
  start)
       start
```

```
        ;;
  stop)
        stop
        ;;
  restart|reload)
        restart
        ;;
  *)
        echo "Usage: $0 {start|stop|restart}"
        exit 1
esac
```

Please make sure that the script is executable:

```
chmod +x /etc/init.d/S99GOAL
```

# 4 Conformance Test

In order to perform the conformance test, several configurations must be tested.
Configuration of the Master Station the network and the expected slaves is done by API functions as described in chapter 2.
There is a sample application that contains the required configurations. In can be found in *appl/goal_ccl_ie_tsn/03_master_ct.*

There are configuration macros to enable or disable features. If a new configuration is required, the macros must be set to the appropriate values. The application must be recompiled (see chapter 3.1.5) and copied to the LS1028ARDB (see chapter 3.1.6).

| Configuration Macro | Description |
|---|---|
| GOAL_APPL_SLAVE0_ENABLED | 1: enable Slave 0<br>0: disbale Slave 0<br>(NZ2GN2B1-32DTE, 192.168.3.1) |
| GOAL_APPL_SLAVE1_ENABLED | 1: enable Slave 1<br>0: disbale Slave 1<br>(RJ71GN11-T2, 192.168.3.2) |
| GOAL_APPL_SLAVE2_ENABLED | 1: enable Slave 0<br>0: disbale Slave 0<br>(MR-J5-10G, 192.168.3.10) |
| GOAL_APPL_SLAVE2_ALT_MAPPING | 1: usa an alternate PDO mapping for Slave 2<br>0: use default PDO mapping for Slave 2 |
| GOAL_APPL_CO_OBJ_TEST | 1: execute CANopen object test for Slave 2<br>0: do not execute test |
| GOAL_APPL_SLMP_TEST | 1: execute SLMP Client test<br>0: do not execute test |
| GOAL_APPL_TIMESYNC_1588 | 1: use IEEE 1588v2 for time synchronization<br>0: use IEEE 802.1AS for time synchronization |
| GOAL_APPL_MULTICAST | 1: use multicast frames for cyclic communication<br>0: use unicast frames for cyclic communication |
| GOAL_APPL_TIME_SYNC_SLAVE | 1: device is not the Grandmaster<br>0: device is the Grandmaster |
| GOAL_APPL_LINKSPEED_100 | 1: enforce a Link speed of 100 Mbit/s<br>0: enforce a Link speed of 1000 Mbit/s |

Table 4.1: Configuration Macros of Conformance Test sample application