# SoM user manual

*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port.de for the latest version as the document may have a newer version since errors may be corrected or changes for a newer version of the product may be incorporated.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.


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

# Table of figures

# Changelog

| Version | Changes |
|---|---|
| **1.0** | Initial release |
| **1.1** | Added platform specific information for STM32 (9.2) |
| **1.2** | Added raspberry pi target (9.3)<br>Added RPC function for logging CC messages to AC (6.1.8) |
| **1.3** | Added examples with Renesas identification<br>Added example 10 pnio_process_alarm |
| **1.4** | Fixed missing references |

# 1 Introduction

The SoM module provides communication capabilities in the domain of industrial communication. The SoM acts as communication controller (CC), executing all external communication. Within this document this module is references as CC or CCM (communication controller module). The application controller (AC), which is a user defined controller with a set of software modules, controls the communication controller using a set of APIs. Communication between CC and AC uses the protocol “Core To Core”. This document describes APIs that are specific to the CCM module function.

# 2 Document structure

| Chapter | Content |
|---|---|
| **Introduction** | Introduction of this document |
| **Document Structure** | This chapter |
| **Device architecture** | Introduction to basic concepts and the architecture of the SoM module |
| **Application** | Guideline for application programming |
| **Communication stack** | Description of the communication interface of the module |
| **API** | Listing and explaination of the available APIs of the SoM module |
| **Examples** | Description of the examples |
| **Trouble Shooting** | List of common usage problems |
| **Targets** | Target specific information |

Table 1 Content of this document

# 3 Device architecture

## 3.1 Architecture

The module software contains a domain specific middleware (GOAL) with several software stacks that can be utilized to build applications in the domain of industrial communication. The following software stacks are provided:

**Device Detection** : A simple protocol for device management

**PROFINET with SNMP** : A communication stack for PROFINET communication

**EtherNet/IP** : A communication stack for EtherNet/IP communication

**Web Server** : A simple web server for application specific management and information provision

**TCP/IP Stack** : A TCP/IP stack which provides UDP and TCP channels
The device is utilized by an application controller. Both controllers communicate cyclicly, where the communication is dictated by the application controller. The application controller (AC) contains the application which orchistrates services of the module to build a customer application.

## 3.2 Interface

### 3.2.1 Hardware interface

The module provides a interface with following signals:

| Pin | Signal | Description |
|---|---|---|
| **1** | VCC33 | external DC |
| **2** | GND | ground |
| **3** | CS | SPI chip select, active low |
| **4** | RESET | external reset, active low |
| **5** | MISO | SPI MISO signal of the SPI interface |
| **6** | MOSI | SPI MOSI signal of the SPI interface |
| **7** | SCK | SPI SCL clock signal of the SPI interface |
| **8** | CATSYNC0 | EtherCAT synchronization function |
| **9** | CATSYNC1 | EtherCAT synchronization function |

Table 2 module hardware interface

### 3.2.2 Software Interface

The software interface of the module contains various layers, which provide communication channels according to the requirements of the communication data.
Over SPI a frame of up to 128 bytes of data is cyclically transfered, where the communication is initiated by the AC.

The SPI frame contains multiple segments of data, typically real time data from communication

stacks and non realtime data, which originates from rpc (remote procedure call). A detailed description oft he communication stack is given in chapter

### 3.2.3 Integration of communication layers / middleware

All layers of the required communication protocol and the AC applications are implemented using the GOAL middleware. This the AC application requires to be based on an implementation of this middleware. To utilize a certain feature set of the CC module (as fieldbus stack PROFINET), wrapper functionality is required that implements the RPC functions. These wrappers require GOAL, thus the target platform must support the GOAL middleware.

# 4 Application

## 4.1 Introduction

This document provides information about how to build an application for the SoM communication module.

## 4.2 Hardware setup

The module provides a management inferface, where initial configuration can be done. These properties can be stored permanently within the device. Following properties are provided to configure the SPI interface:

| Variable | Description | Default value |
|---|---|---|
| **SPI_TYPE** | SPI master/slave configuration | 1 = SLAVE |
| **SPI_MODE** | SPI timing mode | 0 = MODE0 |
| **SPI_UNITWIDTH** | SPI single transfer size | 0 = 8 BIT |
| **SPI_BITORDER** | SPI bit transfer direction | 0 = MSB |
| **SPI_TRANSFERSIZE** | SPI packet transfer size | 128 |

Table 3 SPI configuration

The module does only support SPI_TYPE slave.

The module does only support SPI_UNITWIDTH of 8 bit.

The module does only support SPI_TRANSFERSIZE of 128 byte.

The SPI mode can be configured according to Table 4.

The SPI bitorder can be configured according to Table 5.

| Value | SPI MODE |
|---|---|
| **0** | Mode 0 |
| **1** | Mode 1 |
| **2** | Mode 2 |
| **3** | Mode 3 |

Table 4 SPI Mode configuration

| Value | SPI BITORDER |
|---|---|
| **0** | MSB |
| **1** | LSB |

Table 5 SPI bitorder configuration

## 4.3 Basic application setup

A basic application consists of the optional function calls appl_init, appl_setup and appl_loop. It follows the design philosphy of the GOAL middleware.

## 4.4 Default Features

By default, the SoM CC module will start a web server (on port 8080) for the firmware update feature and an instance of Device Detection for management using the management tool.
It is not possible by an AC application to disable the web server on the CC. However, the AC can start another instance of the web server.
It is possible for an AC application to limit the by default full management access through Device Detection in different ways. Please refer to the corresponding chapter in the documentation.

## 4.5 Features

### 4.5.1 Device Detection

#### *4.5.1.1 Initial state*

On the SoM communication module DD is enabled by default. This allows full access to all variables and logs on the CC module. DD bases on a simple UDP based protocol with no encryption whatssoever.

Thus, an application can and *should* limit the access to a feasable range. Limitation of features and access is possible on various levels.

#### *4.5.1.2 Initial disabling of features by the application*

This property is applied at application startup, when the AC application setup is executed.
With the call of **goal_ddNew()** from **appl_setup()** a bitmask can be passed whith bits representing single functions of DD. Please refer to the API documentation of DD for possible values.
If all bits are set, all features (hello, get, set, get_list, wink) are enabled.

#### *4.5.1.3 Initial disabling of features by CM variable*

The in chapter 4.5.1.2 described mechanism can also be set before application start by setting the corrensponding CM variable FEATURE_DISABLE (see chapter 4.9.1). Each request processed by the device detection module will utilize the value of this variable to determine, if the request is allowed to be processed.

Please note, that any call of goal_ddNew will overwrite the value of this variable, if a different initialization value is passed.

#### *4.5.1.4 Temporary enable features of the device detection*

This property is applied at an arbitraty time, for example when configured using a web site provided by the AC application.

```
GOAL_STATUS_T goal_ddSessionFeatureActivate(
    GOAL_DD_T *pHdlDd,                          /**< dd handle */
    uint32_t bitmaskFeatures                    /**< bitmask with feature enable bi
ts set */
);
```

- The parameter bitmask contains bits representing features being enabled
- Those bits overwrite the disable bits from CM for the current session

#### 4.5.1.5 Filtering and access rights

This property is applied at starup of the AC application.
Using filters it is possible to limit access to CM variables by groups (module ids) and variables (variable ids). Currently some filters are predefined for usage. These can be enabled using following function:

```
GOAL_STATUS_T goal_ddFilterAdd(
    GOAL_DD_T *pHdlDd,                          /**< dd handle */
    GOAL_DD_ACCESS_FILTER_SET_T setId           /**< set id */
);
```

This function works on a created instance of DD, thus requires passing of a valid DD handle. The setId can be used with following values:

| Set Id | Description |
|---|---|
| GOAL_DD_ACCESS_FILTER_SET_ALL | Complete access to all variables |
| GOAL_DD_ACCESS_FILTER_SET_BASIC | Filter for minimal function of the management tool |
| GOAL_DD_ACCESS_FILTER_SET_HIDDEN | Filter for hiding critial information |

Table 6 DD Filter IDs

Please refer to the API documentation of DD for detailed information.

### 4.5.2 PNIO

The communication module contains a full featured PROFINET slave stack. For application see the proper PROFINET documentation.

### 4.5.3 EtherNet/IP

The communication module contains a full featured EtherNet/IP slave stack. For application see the proper EtherNet/IP documentation.

### 4.5.4 HTTPD

The communication module contains a web server for basic management functionality. Please refer to the proper http documentation as part of the goal ma

### 4.5.5 Network channels

The communication module provides TCP and UDP communication channels. Please refer to the API documentation within this document for detailed description.

### 4.5.6 Generic Data Provider

For fieldbus communication applications a generic data provider is available, which contains gneralized information and led status for the application.

| Data Provider Information | PROFINET | EtherNet/IP |
|---|---|---|
| **GOAL_MCTC_DP_STATUS_FLG_CONN** | Connection | Connection |
| **GOAL_MCTC_DP_STATUS_FLG_ERR** | Error | Error |
| **GOAL_MCTC_DP_STATUS_FLG_VALID** | - | Process data valid |
| | | |
| **GOAL_MCTC_DP_LED_WINK** | DCP blink signaling | - |
| **GOAL_MCTC_DP_LED_RED1** | Error | Module Status red |
| **GOAL_MCTC_DP_LED_RED2** | Maintainance | Network Status red |
| **GOAL_MCTC_DP_LED_GREEN1** | Connection | Module Status green |
| **GOAL_MCTC_DP_LED_GREEN2** | DCP blink signaling | Network Status green |

Table 7 MCTC Status information

The LED status information are a recommendation regarding conformance/usability of a device implementation. The delivered examples show usage of this feature.

## 4.6 External Reset

The module provides an external reset input, where the AC can perform a reset of the SoM module.

**Caution**: Do not perform this reset during a firmware update of the CC. This will prevent proper update functionality. Therefore, it is recommended to utilize this reset only if the AC module is initially powered up (cold start).

## 4.7 RPC Synchronization reset

If either AC or CC restart, the peer module is required to restart too. Else a repeatedly RPC synchronization is not possible.

### 4.7.1 CC

By default, the CC module does not perform a reset implicitly. If the AC application restarts, a power cycle of the CC module (or a manual reset) is required.

It is possible to configure this behaviour using the CM variable interface (see chapter 4.9.10).

### 4.7.2 AC

The AC performs a reset of it's application on request of the CC, e.g. when the CC restarts. This can happen during a firmware update.

## 4.8 IP Settings

IP Settings can be done with the CM interface.

| Step | Action | Remark |
|---|---|---|
| **1** | Set CM Variable GOAL_ID_NET:IP | Configure IP address |
| **2** | Set CM Variable GOAL_ID_NET:NETMASK | Configure Netmask |
| **3** | Set CM Variable GOAL_ID_NET:GW | Configure Gateway |
| **4** | Set CM Variable GOAL_ID_NET:Valid to 1 | Set IP configuration to valid |
| **5** | Set CM Variable GOAL_ID_NET:DHCP_ENABLED to 0 | Disable DHCP |
| **6** | Set CM Variable GOAL_ID_NET:COMMIT to 1 | Apply IP settings |

Table 8 IP configuration with CM

To enable DHCP set the CM Variable GOAL_ID_NET:DHCP_ENABLED to 1 and perform a power cycle.

## 4.9 Management Interface CC

See goal_db, Page "Variables": The column "Long description" contains documentation of single variables.

The management interface has following functions:

- Responding to scan requests for devices
- Listing CM variables
- Reading CM variables
- Writing CM variables
- Setting IP address
- Wink command

Each of those functions can be permanently disabled by setting a bit in the CM variable GOAL_ID_DD:FEATURE_DISABLE.

### 4.9.1 Management Interface DD

Module Id = GOAL_ID_DD (34)

| Variable Name | Variable ID | Type | Max. Size | Long description |
|---|---|---|---|---|
| MODULENAME | 0 | GOAL_CM_STRING | 20 | Customer specific name of the module |
| CUSTOMERID | 1 | GOAL_CM_UINT32 | 4 | Customer Id |
| RESERVED | 2 | GOAL_CM_UINT8 | 1 | - |
| FEATURE_DISABLE | 3 | GOAL_CM_UINT32 | 4 | Each bit disables a function:<br>bit 0, disable "HELLO DETECTION"<br>bit 1, disable WINK<br>bit 2, disable GETLIST<br>bit 3, disable GET VALUE<br>bit 4, disable SET VALUE |

Table 9 DD management interface

### 4.9.2 PNIO

Module Id = GOAL_ID_PNIO (27)

- Internal variables used by the Profinet stack

### 4.9.3 LM

These variables provide an interface for the logging manager of the SoM module.

Module Id = GOAL_ID_LM (35)

| Variable Name | Variable ID | Type | Max. Size | Long description |
|---|---|---|---|---|
| VERSION | 0 | GOAL_CM_UINT8 | 1 | Version information for LM interface |
| READBUFFER | 1000 | GOAL_CM_GENERIC | 128 | Buffer for reading online logging from deivce |
| CNT | 1001 | GOAL_CM_UINT16 | 2 | Control word for online log access |
| EXLOG_READBUFFER | 1002 | GOAL_CM_GENERIC | 128 | Buffer for reading exception logging from deivce |
| EXLOG_CNT | 1003 | GOAL_CM_UINT16 | 2 | Control word for exception log access |
| EXLOG_SIZE | 1004 | GOAL_CM_UINT32 | 4 | Indicator for exception log size |
| EXLOG_USAGE | 1005 | GOAL_CM_UINT8 | 1 | Indicator for exception log usage |
| EXLOG_ERASE | 1006 | GOAL_CM_UINT8 | 1 | Command:<br>*, Erase Exception Log |

Table 10 LM Management interface

### 4.9.4 NET

Module Id = GOAL_ID_NET (12)

| Variable Name | Variable ID | Type | Max. Size | Long description |
|---|---|---|---|---|
| IP | 0 | GOAL_CM_IPV4 | 4 | IP address of first interface |
| NETMASK | 1 | GOAL_CM_IPV4 | 4 | NETMASK of first interface |
| GW | 2 | GOAL_CM_IPV4 | 4 | GATEWAY of first interface |
| VALID | 3 | GOAL_CM_UINT8 | 1 | Validity of IP address:<br>0, Stored IP address is not valid, interface settings originate from network stack of system<br>1, Stored IP address is valid, will be applied to interface at start of device |
| DHCP_ENABLED | 4 | GOAL_CM_UINT8 | 1 | DHCP enable:<br>0, DHCP disabled<br>1, DHCP enabled |
| DHCP_STATE | 5 | GOAL_CM_UINT8 | 1 | DHCP state:<br>0, DHCP initialized<br>1, DHCP server selecting<br>2, DHCP requesting configuration<br>3, DHCP ip address bound<br>4, DHCP renewing configuration<br>5, DHCP rebinding ip address to interface |
| DNS0 | 6 | GOAL_CM_IPV4 | 4 | First DNS server of first interface |
| DNS1 | 7 | GOAL_CM_IPV4 | 4 | Second DNS server if first interface |
| HOSTNAME | 8 | GOAL_CM_STRING | 20 | Hostname of first interface |
| COMMIT | 1000 | GOAL_CM_UINT8 | 1 | Command:<br>*, Apply IP settings |

Table 11 NET Management interface

### 4.9.5 BOOT

Module Id = GOAL_ID_BOOT (37)

| Variable Name | Variable ID | Type | Max. Size | Long description |
|---|---|---|---|---|
| SIGNATURE | 0 | GOAL_CM_GENERIC | 16 | Signature of booted image |
| BLVERSION | 1 | GOAL_CM_STRING | 16 | Bootloader Version |
| FWVERSION | 2 | GOAL_CM_STRING | 16 | Firmware Version |

| | | | | |
|---|---|---|---|---|
| RESET_CAUSE | 1000 | GOAL_CM_UINT8 | 1 | Reset cause:<br>0, Unspecified<br>1, Firmware Update Requested<br>2, Watchdog<br>3, Firmware Commit Required<br>4, Reserved |
| IMAGE_NUMBER | 1001 | GOAL_CM_UINT8 | 1 | Booted image number |
| IMAGE_COUNTER | 1002 | GOAL_CM_UINT8 | 1 | Booted image age counter |

Table 12 BOOTLOADER Management interface

### 4.9.6 CM

Module Id = GOAL_ID_CM (2)

| Variable Name | Variable ID | Type | Max. Size | Long description |
|---|---|---|---|---|
| VERSION | 0 | GOAL_CM_UINT32 | 4 | Version information for CM interface |
| SAVE | 1000 | GOAL_CM_UINT8 | 1 | Command:<br>*, Save CM to Flash |

Table 13 CM Management interface

### 4.9.7 ETH

Module Id = GOAL_ID_ETH (4)

| Variable Name | Variable ID | Type | Max. Size | Long description |
|---|---|---|---|---|
| MAC | 0 | GOAL_CM_GENERIC | 6 | |
| LINK | 1000 | GOAL_CM_UINT32 | 4 | Link status mask of interfaces |
| SPEED | 1001 | GOAL_CM_UINT32 | 4 | Port speed mask of interfaces |
| DUPLEX | 1002 | GOAL_CM_UINT32 | 4 | Port Duplex mask of interfaces |
| PORTCNT | 1003 | GOAL_CM_UINT32 | 4 | Number of interfaces |

Table 14 ETH Management interface

### 4.9.8 EIP

Module Id = GOAL_ID_EIP (23)

- Internal variables used by the EtherNet/IP stack

### 4.9.9 HTTPD

Module Id = GOAL_ID_HTTP (25)

| Variable Name | Variable ID | Type | Max. Size | Long description |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| HTTP_CHANNELS_MAX | 0 | GOAL_CM_UINT16 | 2 | Determines the number of possible connections to the HTTP server |
| HTTPS_CHANNELS_MAX | 1 | GOAL_CM_UINT16 | 2 | Determines the number of possible connections to the HTTPS server |
| USERLEVEL0 | 2 | GOAL_CM_STRING | 32 | Authentication data for level 0 |
| USERLEVEL1 | 3 | GOAL_CM_STRING | 32 | Authentication data for level 1 |
| USERLEVEL2 | 4 | GOAL_CM_STRING | 32 | Authentication data for level 2 |
| USERLEVEL3 | 5 | GOAL_CM_STRING | 32 | Authentication data for level 3 |

Table 15 HTTP Management interface

### 4.9.10 CCM

Interface for Management Tool for informative and configuration purpose.
Module Id = GOAL_ID_CCM (72)

| Variable Name | Variable ID | Type | Max. Size | Long description |
|---|---|---|---|---|
| SPI_TYPE | 0 | GOAL_CM_UINT8 | 1 | SPI Type (currently only slave supported):<br>0, SPI Master<br>1, SPI Slave |
| SPI_MODE | 1 | GOAL_CM_UINT8 | 1 | SPI Mode:<br>0, CPOL=0; CPHA=0<br>1, CPOL=0; CPHA=1<br>2, CPOL=1; CPHA=0<br>3, CPOL=1; CPHA=1 |
| SPI_SPEED | 2 | GOAL_CM_UINT8 | 1 | SPI Speed in Master Mode |
| SPI_UNITWIDTH | 3 | GOAL_CM_UINT8 | 1 | Bitsize of one single transfer unit:<br>0, 8 Bit<br>1, 16 Bit<br>2, 32 Bit |
| SPI_BITORDER | 4 | GOAL_CM_UINT8 | 1 | Bitorder of SPI transfers:<br>0, MSB first<br>1, LSB first |
| SPI_TRANSFERSIZE | 5 | GOAL_CM_UINT16 | 2 | Minimum transfer size of single transmission frame |

| | | | | |
|---|---|---|---|---|
| COMM_FAULT_ERROR_STATE | 6 | GOAL_CM_UINT8 | 1 | Fault action to execute when communication to AC was lost during a cyclic connection:<br>0, Enter fault state (disable connection)<br>1, Keep running (keep connection) |
| COMM_SYNC_RESET | 7 | GOAL_CM_UINT8 | 1 | Behaviour when a sync reset request was received from AC:<br>0, Do nothing<br>1, Perform reset of CC controller |
| FW_UPDATE_COMMIT_DISABLE | 8 | GOAL_CM_UINT8 | 1 | Optional disable of the additional commit step during firmware update:<br>0, Firmware update requires commit step<br>1, Firmware update doesn't require a commit step |
| UPTIME | 1000 | GOAL_CM_UINT32 | 4 | Number of seconds since start of device |

Table 16 CCM Management interface

## 4.10 Firmware Update

### 4.10.1 Update the communication controller

Firmware update of the communiation controller is possible in the field. It is done using the Management tool. This tool uses the http protocol to transfer a firmware package to the device.

#### *4.10.1.1 Firmware package*

Firmware image is bundled within a package. This package contains a signed firmware, which secures only acceptance of firmware from the device originator. Thus, it is possible for the device to check authenticity of the firmware image.

#### *4.10.1.2 Control interface*

By default, a firmware update is possible at any time. The communication module provides an interface to enable and disable the firmware update process. An application should use this interface as there are situations where a firmware update should not be accepted. This should be deactivated during an active cyclic connection to the PLC.

For usage of this interface see chapter 6.1.

#### *4.10.1.3 Firmware update sequence*

Firmware update is a two-step process. At first the firmware is uploaded to the http server of the CC module. Following checks are done during this check:

1. This upload uses authentication with authentication level 0, where credentials are checked if configured for the HTTPD module. Those variables are configurable through the RPC interface of the HTTPD service. By default, these credentials are empty. Thus, any attempt to authenticate is accepted.

2. The firmware update must be allowed by the AC. By default, this is the case. However, the AC can disable this function using the RPC interface.

If at the end of the transfer a valid firmware is detected, the module restarts and enters the bootloader. This software module checks the signature of the firmware and writes the correctly signed firmware to the memory of the device. As a fallback, the previously run firmware is kept.

After restart of the module a successful communication to the management tool required to permanently enable the updated firmware. If this is possible, the module will restart again, and the bootloader will mark the new firmware as the current firmware. If this fails, the bootloader will revert to the original firmware with the next power cycle.

#### *4.10.1.4 Keep update functionality while disabling DD*

An application integrator might want to disable the DD feature to not expose any internal information of the device. Disabling is possible. However, to allow a firmware update using the management tool, the following steps need to be implemented:

1. Disabling DD by default

   When calling the API function goal_ddNew, a bitmask can be passed, that determines the active DD features. If the predefined value GOAL_DD_FEAAT_NO is used, DD is disabled completely.
   Internally this value is stored to the CM variable FEATURE_DISABLE, which state also can be saved permanently to flash.

2. Introducing a switch to termporarily enable DD features

If a firmware update shall be processed, minimal DD features need to be set temporarily. These are:

- HELLO REQUEST
- SET CONFIG
- GET CONFIG
- SET IP

This can be done using the API function goal_ddSessionFeatureActivate() with the following arguments:

```
goal_ddSessionFeatureActivate(pHdlDd, GOAL_DD_FEAT_GETCONFIG |
                                      GOAL_DD_FEAT_SETCONFIG |
                                      GOAL_DD_FEAT_SETIP);
```

The temporary switch, to enable the management interface, can be implemented using the web server. See example 01_pnio_io_mirror for usage.

As a result, the device will be invisible for the management tool. For firmware update, the required interface is temporarily activated, thus allowing to update the communication module firmware.

### 4.10.2 Update possibilities for application controller

By default, there is no firmware update available for the application controller. However, the module provides mechanisms for implementing such a feature within the customer application. Following, two possible solutions are shown:

#### *4.10.2.1 AC Firmware update over HTTP*

The application controller can provide a web site or tool-based firmware update over HTTP

transport, utilitung the provided RPC interface of the HTTPD service. As a starting point, the HTTPD example *goal_http/02_post*, respectively ccm example *20015013_SoM/01_pnio_io_mirror* can be used.

For latter, data for firmware update transported using a HTTP POST request will be processed in the callback function ***httpDataCbPost***, for example as shown in the following sample code:

```
/*****************************************************************************/
/** goal http data callback
 *
 */
static GOAL_STATUS_T httpDataCbPost(
    GOAL_HTTP_APPLCB_DATA_T *pCbInfo            /**< pointer to callback info struc
t */
)
{
    GOAL_STATUS_T res = GOAL_OK;                /* result */
    static uint32_t uploadDataLen = 0;          /* recent uploaded data length */

    if (hdlUplHtml == pCbInfo->hdlRes) {
        /* check request method */
        switch (pCbInfo->reqType)
        {
        case GOAL_HTTP_FW_POST_START:
            /* reset data length */
            uploadDataLen = 0;
            GOAL_HTTP_RETURN_OK_204(pCbInfo);
            break;

        case GOAL_HTTP_FW_POST_DATA:

            /* process data */
            GOAL_MEMCPY(pDst, pCbInfo->cs.pData, pCbInfo->cs.lenData);
            uploadDataLen += pCbInfo->cs.lenData;

            GOAL_HTTP_RETURN_OK_204(pCbInfo);
            break;

        case GOAL_HTTP_FW_POST_END:
            GOAL_HTTP_RETURN_OK_204(pCbInfo);
            break;

        case GOAL_HTTP_FW_REQ_DONE_OK:
        case GOAL_HTTP_FW_REQ_DONE_ERR:
            res = GOAL_OK;
            break;

        default:
            /* return error */
            GOAL_HTTP_RETURN_ERR_403(pCbInfo);
            break;
        }
    }
    else {
        /* return error */
        GOAL_HTTP_RETURN_ERR_404(pCbInfo);
    }
    return res;
}
```

#### *4.10.2.2 AC Firmware update over TCP*

If HTTP is not favoured for transport of the firmware update, a plain TCP socket can be used for data transfer. This transport can be used the stream of data it provides, or any additional protocol can be used above this transport. See RPC interface of the networking tcp module.

Respectively, following callback function will handle received data:

```
/***************************************************************************/
/** TCP Server Callback
 *
 * Process TCP data stream segments
 */
static void tcpCallback(
    GOAL_MA_CHAN_TCP_T *pMaTcpHdl,              /**< MA handle */
    GOAL_NET_CB_TYPE_T cbType,                  /**< callback type */
    struct GOAL_NET_CHAN_T *pChan,              /**< channel descriptor */
    struct GOAL_BUFFER_T *pBuf                  /**< GOAL buffer */
)
{
    GOAL_NET_ADDR_T remote;                     /* remote address */
    GOAL_STATUS_T res;                          /* result */

    if (cbType == GOAL_NET_CB_NEW_SOCKET) {
        goal_logInfo("new TCP listener");
    }
    else if (cbType == GOAL_NET_CB_NEW_DATA) {

        /* process data */

        goal_logDbg("Data received on tcp socket %p", (void *) pChan);
    }
}
```

# 5 Communication stack

## 5.1 SPI Data Exchange

The communication with the SoM module uses the well-known Serial Peripheral Interface (SPI). A transfer must always be triggered by the application core (AC) and must at least be once during the heartbeat timeout. The implementation in the SoM updates the SPI data after each transfer to make sure it doesn't interrupt a running transfer.

### 5.1.1 Clock domains and communication cycle

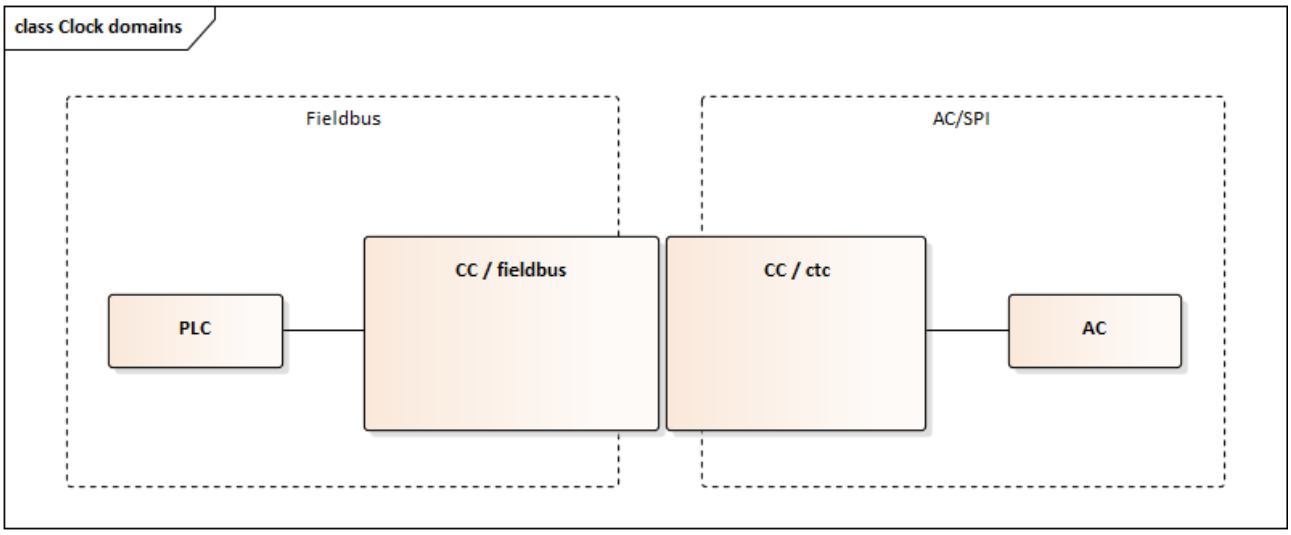


Figure 1 ctc clock domains

Operation of the device includes two clock domains, which run independently of each other. First clock domain is the fieldbus side. Commonly the PLC interacts with the device in one clock domain, where the PLC controls the timing of the output data. Second clock domain is drived by the AC through initiation of the SPI cycle, which reads the output data from the device and updates input data for the next fieldbus cycle. Therefore, a specific timing of the process data is set up as shown in Figure 2 Communication Cycle.

Following data transports occur:

1. New output data from PLC
2. Processing of output data in CC module and preloading of SPI transfer buffer
3. A finished SPI transfer initiated by the AC executes data exchange between AC and CC
4. Preloading of new input data for the next SPI transfer
5. A finished sequential SPI transfer executes data exchange between AC and CC, thus providing new input data for the next fieldbus transfer

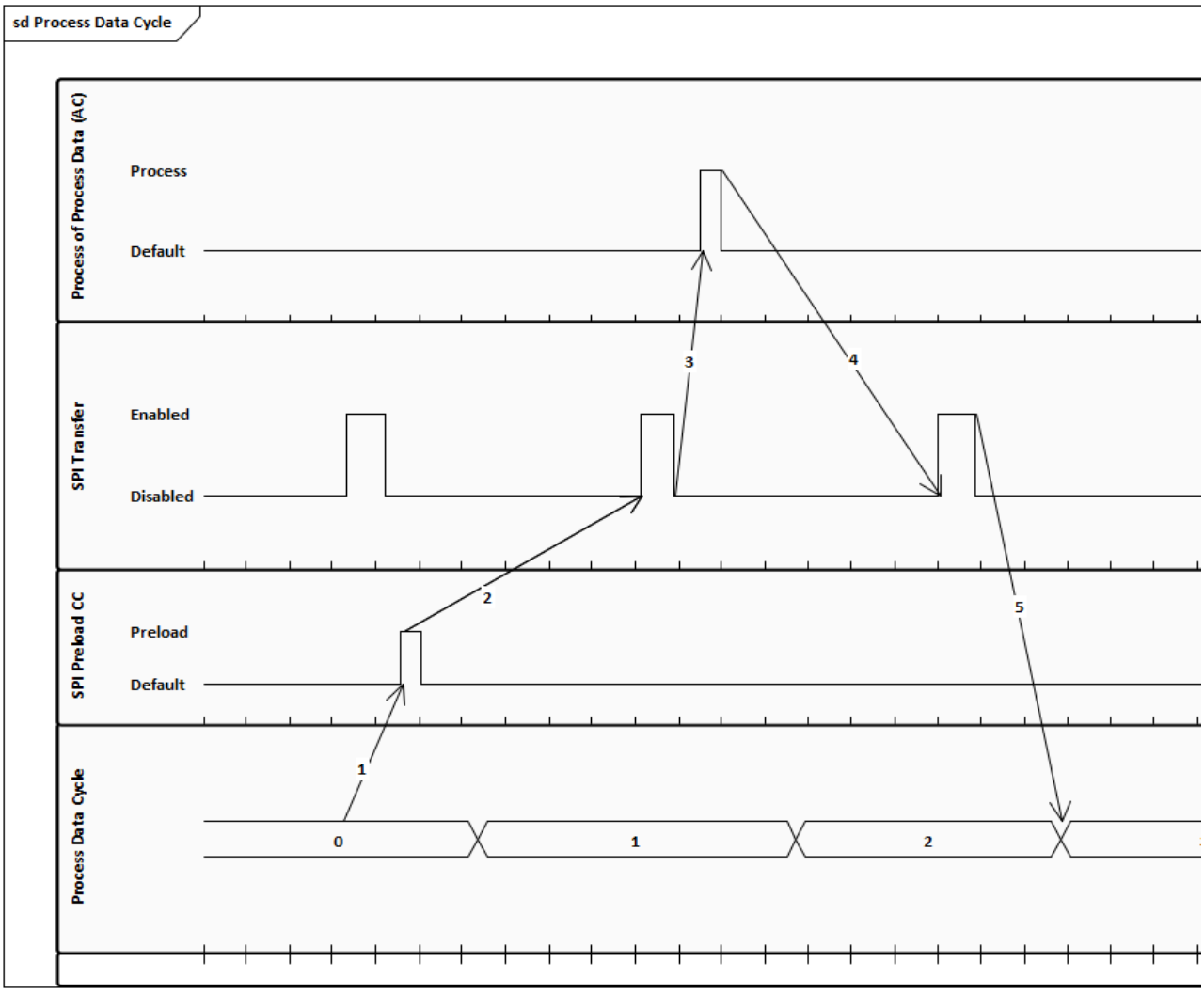


Figure 2 Communication Cycle

### 5.1.2 Technical data

- Transfer length:
  - Cyclic only: 72 Bytes
  - Cyclic + RPC data: 128 Bytes
- Baud-rate: min ... max
- Delay between SPI transfers: 0.5 ms ... Heartbeat Timeout (1 second)
- Minimal round-trip time: 4 ... 6 ms (depending on the used protocol and setup

### 5.1.3 SPI Timing

Following simplified diagram shows basic SPI timing which must be considered by the application controller.

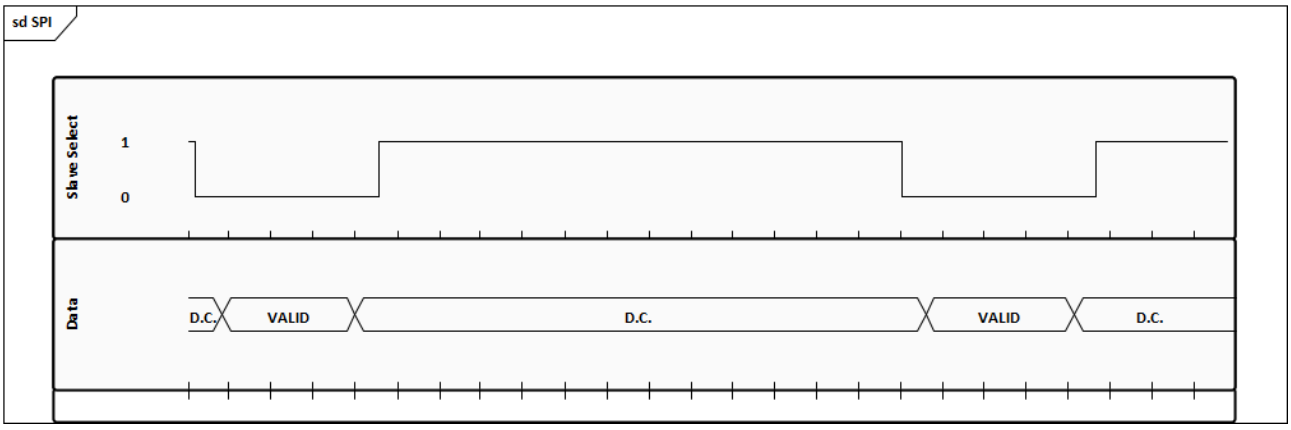


Figure 3 Basic SPI timing

#### 5.1.3.1 SPI Speed

The communication module supports SPI speed in the range of 29.3 kHz and 10MHz. Default SPI speed is 2MHz.

#### 5.1.3.2 SPI Setup Time

In Figure 3 Basic SPI timing, the communication scheme between AC and CC is shown. Following time needs to be considered by the Application Controller during communication.

*SPI Setup Time .. Time between activation of the module using the Slave Select signal and first data*

The SPI Setup Time must at least be 10ns, before the module accepts data over SPI.

#### 5.1.3.3 SPI Cycle Time

This time is the distance between two consecutive SPI transfers. Between two transfers a minimum time of 250 us must be kept to secure proper processing in the module.

### 5.1.4 SPI Frame Structure

The SPI frame must contain the structure from Table 1: SPI Frame Structure to get accepted by the SoM module.

| **Bytes 0..1** | **Byte 2** | **Byte 3** | **Bytes 4 .. 76** | **Bytes 77 .. 127** |
|---|---|---|---|---|
| **Fletcher-16 Checksum with Offset 0x0007 (little endian)** | Sequence | Data Length | Cyclic Data | RPC Data |

Table 17 SPI Frame Structure

The same layout is send back by the device containing its local sequence counter. The sequence counter is tracked and if it doesn’t change during the heartbeat timeout the communication gets

stopped until the sequence is updated again.

#### 5.1.4.1 Fletcher-16 Checksum (16 Bit)

To calculate the Fletcher-16 checksum the Wikipedia entry https://en.wikipedia.org/wiki/Fletcher%27s_checksum#Example_calculation_of_the_Fletcher-16_checksum can be used. Start index is byte 4 and end is at 127. After the calculation the value 0x0007 needs to be added to not have false positives if the whole area is set to zeros. In the frame the value has a width of 16 bit and needs to have the little endian encoding.

#### 5.1.4.2 Sequence Counter (8 Bit)

The sequence counter must be incremented on each sent out frame to be recognized as new data. It can start at any 8 bit value.

#### 5.1.4.3 Data Length (8 Bit)

If only cyclic data is transferred the data length can be set from 0 to 73 bytes. When using RPC over SPI the data length needs to have a fixed value of 124.

## 5.2 Remote Procedure Call

The RPC protocol used by the Micro Core-To-Core (MCTC) implementation of the SoM module is transferred in byte 77 – 127. RPC transfers are always acknowledged to minimize data loss on erroneous transfers as good as possible. Also the RPC calls can be larger than the available 50 bytes as the SoM module internally stores each received RPC frame in a ring-buffer and waits until a partitioned transfer is completed before proceeding with the request.

### 5.2.1 RPC Frame Structure

The RPC frame must contain the structure from Table 1: RPC Frame Structure to be accepted by the SoM module.

| **Bytes 0..1** | **Byte 2** | **Byte 3** | **Byte 4** | **Byte 5** | **Bytes 6 .. 49** |
|---|---|---|---|---|---|
| **Fletcher-16 Checksum with Offset 0x0007 (little endian)** | Local Sequence | Remote Sequence Acknowledge | Data Length | Flags | Data |

Table 18 RPC Frame Structure

Each time a new local sequence is sent the SoM will respond with the corresponding acknowledge

in the second transferred frame. If no acknowledge was received the AC can retransmit its frame to re-request the acknowledge.

#### 5.2.1.1 Fletcher-16 Checksum (16 Bit)

To calculate the Fletcher-16 checksum the Wikipedia entry https://en.wikipedia.org/wiki/Fletcher%27s_checksum#Example_calculation_of_the_Fletcher-16_checksum can be used. Start index is byte 6 and end is at the included data length. After the calculation the value 0x0007 needs to be added to not have false positives if the whole area is set to zeros. In the frame the value has a width of 16 bit and needs to have the little endian encoding.

#### 5.2.1.2 Local Sequence (8 Bit)

The sequence counter must be incremented for each RPC frame with changed data. For each unseen incremental frame the SoM module will put the transferred data into its internal ring-buffer and after the length matches it will process the RPC call.

#### 5.2.1.3 Remote Sequence Acknowledge (8 Bit)

For each processed received RPC frame the AC needs to send out an acknowledge. If the received frame doesn't match the expected sequence the AC must leave the acknowledge on the previous sequence to trigger a resend from the SoM module. If the resend fails the AC can also perform a re-sync.

#### 5.2.1.4 Data Length (8 Bit)

Contains the length of the RPC data and must be between 0 (acknowledge-only frame) and 44.

#### 5.2.1.5 Flags (8 Bit)

| **Bit 0** | **Bit 1** | **Bit 2** | **Bit 3** |
|---|---|---|---|
| Sync Request | Sync Acknowledge | *Reserved* | Request Acknowledge |

Table 19 RPC Header Flags

##### 5.2.1.5.1 Sync Request

If set to 1 the SoM module enters the RPC synchronization.

#### 5.2.1.5.2 Sync Acknowledge

During the *Sync Request* both sides need to set the *Sync Acknowledge* flag, see RPC Synchronization for details.

#### 5.2.1.5.3 Request Acknowledge

Forces an acknowledge from the partner device.

### 5.2.2 RPC Synchronization

Before the SoM module is ready to operate and after a synchronization is lost the RPC must be synchronized. This is triggered by setting the **Sync Request** flag to 1.

| **AC** | **SoM Module** |
|---|---|
| Sync Request = 1<br>Local Sequence = 0<br>Local Sequence Acknowledge = 0<br>Remote Sequence Acknowledge = 0 | |
| | Sync Request = 1<br>Local Sequence = 0<br>Local Sequence Acknowledge = 0<br>Remote Sequence Acknowledge = 0 |
| Sync Request = 0<br>Sync Acknowledge = 1<br>Local Sequence = 1<br>RPC_Send(<empty>) | |
| | RPC_Receive()<br>→ Remote Sequence Acknowledge = 1<br><br>Sync Request = 0<br>Sync Acknowledge = 1<br>Local Sequence = 1<br>RPC_Send(<empty>) |
| RPC_Receive()<br>→ Local Sequence Acknowledge = 1<br>→ Remote Sequence Acknowledge = 1<br><br>Sync Acknowledge = 0 | |

| | |
|---|---|
| RPC_Start() | |
| | RPC_Receive()<br>→ Local Sequence Acknowledge = 1<br><br>RPC_Start() |

Table 20 RPC Synchronization

After the synchronization has finished the SoM both devices must initiate the generic RPC call SetupStateGet to verify if the partner device is in the same state. If one partner is already configured but the other is not, than the configured parter must reboot and the synchronization starts again.

#### *5.2.2.1 Normal Operation*

If the AC successfully passed the synchronization stage and configured the SoM module it must call the RPC API SetupDone. At this point the SoM module is ready for normal operation.

### 5.2.3 RPC Protocol

The GOAL RPC protocol uses a virtual push/pop stack to call remote API functions with arguments. Each call must have a return value that is usually set to GOAL_OK when the call succeeded.

#### *5.2.3.1 RPC Request/Response Structure*

An RPC request/response consists of the parts shown in Table 21 and Table 22.

| **Byte 0..1** | **Byte 2..5** | **Byte 6 .. x** | **Byte x+1 .. x+4** | **Byte x+5..x+8** | **Byte x+9** | **Byte x+10** | **Byte (x+11) .. (x+14)** |
|---|---|---|---|---|---|---|---|
| Static Identifier 0xaa, 0xee | Data Length from Byte 6 to Flags (included) | Data | Function Id (little endian) | RPC Id (little endian) | CTC Id | Flags | Fletcher-16 Checksum with Offset 0x0007 (little endian) |

Table 21 RPC Request Structure

| **Byte 0..1** | **Byte 2..5** | **Byte 6 .. x** | **Byte x+1** | **Byte x+2** | **Byte (x+3)..(x+4)** |
|---|---|---|---|---|---|
| Static | Data Length | Response | CTC Id | Flags | Fletcher-16 |

| | | | | | |
|---|---|---|---|---|---|
| Identifier 0xaa, 0xee | from Byte 6 to Flags (included) | Data | | | Checksum with Offset 0x0007 (little endian) |

Table 22 RPC Response Structure

#### 5.2.3.1.1 Static Identifier

The 2 byte static identifier is used on the SoM module to detect the start of a new RPC request or response. If the identifier is also contained in the data bytes the Fletcher-16 checksum will make sure that it don't gets threaded like an RPC request/response.

#### 5.2.3.1.2 Data Length

The data length contains the count of bytes starting with the "Function Id" and ending with the last data byte.

#### 5.2.3.1.3 RPC Id

The RPC id is the module or group id. For example GOAL PROFINET uses the RPC id GOAL_ID_PNIO to register its calls.

#### 5.2.3.1.4 Function Id

The function id is used as a sub id to map the function calls in the specific module to their handlers.

#### 5.2.3.1.5 CTC Id

The CTC id is used to match requests and responses to their specific MCTC internal handle. A response must use the same CTC id as the request.

#### 5.2.3.1.6 Flags

The flags define the type of the request, see Table 5: RPC Request/Response Flags for details.

| **Bit 0..1** |
|---|
| 0 – Response<br>1 – Request<br>2 – Info (Request without waiting for a response) |

Table 23 RPC Request/Response Flags

#### 5.2.3.1.7 Data

The data part contains the virtual stack of the RPC request/response.

#### 5.2.3.1.8 Fletcher-16 Checksum (16 Bit)

To calculate the Fletcher-16 checksum the Wikipedia entry https://en.wikipedia.org/wiki/Fletcher%27s_checksum#Example_calculation_of_the_Fletcher-16_checksum can be used. Start index is byte 16 and end is at the included data length. After the calculation the value 0x0007 needs to be added to not have false positives if the whole area is set to zeros. In the frame the value has a width of 16 bit and needs to have the little endian encoding.

## 5.3 GOAL PROFINET Data Mapper API

The GOAL PROFINET Data Mapper API allows to map PROFINET subslots, producer & consumer states and the connection information to the cyclic data stream of the Data Mapper that is used for example in MCTC transfers.

### 5.3.1 Map Subslot Data – goal_pnioDmSubslotAdd

The API goal_pnioDmSubslotAdd maps input and output data of the given subslot to the also given instances of the DM.

| Parameter | Description |
|---|---|
| GOAL_PNIO_T *pPnio | GOAL PROFINET instance |
| uint32_t idMiDmPeerFrom | Handle that receives data from CC |
| uint32_t idMiDmPeerTo | Handle that sends data to CC |
| GOAL_MI_DM_PART_T **ppPartDataOut | Output pointer to store the DM partition |
| GOAL_MI_DM_PART_T **ppPartDataIn | Output pointer to store the DM partition |
| uint32_t idApi | API id |
| uint16_t idSlot | Slot id |
| uint16_t idSubslot | Subslot id |
| uint32_t lenDataOut | Output data length |
| uint32_t lenDataIn | Input data length |

Table 24 goal_pnioDmSubslotAdd API Description

### 5.3.2 Map Subslot IOCS/IOPS - goal_pnioDmSubslotIoxsAdd

The API goal_pnioDmSubslotIoxsAdd maps IOCS and IOPS states of the given subslot to the given instances of the DM.

| Parameter | Description |
|---|---|
| GOAL_PNIO_T *pPnio | GOAL PROFINET instance |
| uint32_t idMiDmPeerFrom | Handle that receives data from CC |
| uint32_t idMiDmPeerTo | Handle that sends data to CC |

| | |
|---|---|
| GOAL_MI_DM_PART_T **ppPartIocsOut | Output pointer to store the DM partition |
| GOAL_MI_DM_PART_T **ppPartIopsOut | Output pointer to store the DM partition |
| GOAL_MI_DM_PART_T **ppPartIocsIn | Output pointer to store the DM partition |
| GOAL_MI_DM_PART_T **ppPartIopsIn | Output pointer to store the DM partition |
| uint32_t idApi | API id |
| uint16_t idSlot | Slot id |
| uint16_t idSubslot | Subslot id |

Table 25 : goal_pnioDmSubslotIoxsAdd API Description

### 5.3.3 Map APDU Status – goal_pnioDmApduAdd

The API goal_pnioDmApduAdd maps the APDU status to the given instance of the DM.

| Parameter | Description |
|---|---|
| GOAL_PNIO_T *pPnio | GOAL PROFINET instance |
| uint32_t idMiDmPeerTo | (ignored) ~~Handle that sends data to CC~~ |
| GOAL_MI_DM_PART_T **ppPartApduOut | Output pointer to store the DM partition |

Table 26 goal_pnioDmApduAdd API Description

### 5.3.4 Map Data Provider Status – goal_pnioDmDpAdd

The API goal_pnioDmDpAdd maps the Data Provider status to the given instance of the DM.

| Parameter | Description |
|---|---|
| GOAL_PNIO_T *pPnio | GOAL PROFINET instance |
| uint32_t idMiDmPeerTo | (ignored) ~~Handle that sends data to CC~~ |
| GOAL_MI_DM_PART_T **ppPartDp | Output pointer to store the DM partition |

Table 27 goal_pnioDmApduAdd API Description

# 6 Application Programming Interface

This chapter lists the API functions that are provided by SoM.

## 6.1 device specific functions

### 6.1.1 appl_ccmRpcInit

**Purpose:** Register SoM API in GOAL (appl_init)

This function registers the SoM specific API in GOAL and must be called in appl_init. It returns a GOAL_STATUS_T status and has no parameters.

**Function Prototype:**

```
GOAL_STATUS_T goal_ddRpcInit(
    void
);
```

**Example:**

```
/****************************************************************************/
/** Application Init
 *
 * Build up the device structure and initialize the Profinet stack.
 */
GOAL_STATUS_T appl_init(
    void
)
{
    GOAL_STATUS_T res = GOAL_OK;                /* result */

    /* initialize ccm RPC interface */
    res = appl_ccmRpcInit();
    if (GOAL_RES_ERR(res)) {
        goal_logErr("Initialization of ccm RPC failed");
    }

    return res;
}
```

### 6.1.2 appl_ccmUpdateAllow

**Purpose:** Enable firmware update in the Communication Controller

This function enables the possibility to update the firmware of the SoM CC module. It returns a GOAL_STATUS_T status and has no parameters.

**Function Prototype**

```
GOAL_STATUS_T appl_ccmUpdateAllow(
    void
);
```

**Example:**

For an example, where firmware update capability can be enabled and disabled using the web server please check example project 01_pnio_io_mirror (chapter 7.1).

### 6.1.3 appl_ccmUpdateDeny

**Purpose:** Disable firmware update in the Communication Controller

This function disables the possibility to update the firmware of the SoM CC module. Possibly use of this function is to disable firmware update possibilities during a cyclic communication relation.
It returns a GOAL_STATUS_T status and has no parameters.

**Function Prototype:**

```
GOAL_STATUS_T appl_ccmUpdateDeny(
    void
);
```

**Example:**

For an example, where firmware update capability can be enabled and disabled using the web server please check example project 01_pnio_io_mirror (chapter 7.1).

### 6.1.4 appl_ccmInfo

**Purpose:** Get version and device information

This function reads information from the SoM:

- MAC address (serial number)
- Software version
- device type

**Function Prototype:**

```
GOAL_STATUS_T appl_ccmInfo(
    char *strVersion,                       /**< target string for version */
    uint8_t lenStrVersion,                  /**< length of str buffer */
    GOAL_ETH_MAC_ADDR_T *macAddress,        /**< mac address buffer */
    uint16_t *pDevType                      /**< SoM device type */
);
```

**Example:**

```
/* get version and information of ccm */
if (GOAL_RES_OK(res)) {
    res = appl_ccmInfo(strVersion, APPL_VERSION_LEN, &mac, &devType);
}
```

```
if (GOAL_RES_OK(res)) {
    goal_logInfo("ccm version : %s", strVersion);
    goal_logInfo("ccm device : %u", devType);
    goal_logInfo("ccm Serial : %02x:%02x:%02x:%02x:%02x:%02x",
        mac[0], mac[1], mac[2], mac[3], mac[4], mac[5]);
}
```

### 6.1.5 appl_ccmFaultStateSet

**Pupose**: Set behaviour of fieldbus communication on fault

This function determines the behaviour of the SoM regarding cyclic communication. By default, a cyclic communication is stopped when communication to the application controller (AC) is lost.

**Function Prototype:**

```
GOAL_STATUS_T appl_ccmFaultStateSet(
    APPL_CCM_FAULT_STATE_T faultState         /**< fault state to enter */
);
```

**Example:**

```
/* set fault state behaviour */
if (GOAL_RES_OK(res)) {
    res = appl_ccmFaultStateSet(APPL_CCM_FAULT_STATE_ENTER);
}
```

### 6.1.6 appl_ccmCommResetSet

**Pupose**: Set behaviour of SoM module on SPI sync reset request

A sync reset request is requested by the AC upon restart, while the CC was previously setup with a running AC application.
This function determines the behaviour of the SoM regardinging this reset request. By default, no reset is done. If this option is enabled, reset is done.
The state of this setting is stored in non volatile memory on the CC. A value of 0 disables the reset. A value of 1 enables the reset.

**Function Prototype:**

```
GOAL_STATUS_T appl_ccmCommResetSet(
    uint8_t value                             /**< option value */
);
```

**Example:**

```
/* set sync reset behaviour */
if (GOAL_RES_OK(res)) {
    /* enable reset on sync reset request from AC */
    res = appl_ccmCommResetSet(1);
```

```
}
```

### 6.1.7 appl_ccmLogEnable

**Pupose**: Enable transport of AC log messages to the CC

Once this function is executed, log messages from the AC's logging buffer are continuously transferred to the CC module. Those are then accessible as the CC's own log messages through the management interface.

**Function Prototype:**

```
GOAL_STATUS_T appl_ccmLogEnable(
    void
);
```

**Example:**

```
/* enable logging to CC */
if (GOAL_RES_OK(res)) {
    res = appl_ccmLogEnable();
}
```

### 6.1.8 appl_ccmLogToAcEnable

**Pupose**: Enable transport of CC log messages to the AC

Once this function is executed, log messages from the CC's logging buffer are continuously transferred to the AC module. Those are then accessible as the AC's own log messages through the local log mechanism, e.g. serial console or terminal.

**Function Prototype:**

```
GOAL_STATUS_T appl_ccmLogToAcEnable(
    void
);
```

**Example:**

```
/* enable logging from CC  to AC*/
if (GOAL_RES_OK(res)) {
    res = appl_ccmLogToAcEnable();
}
```

## 6.2 Device Detection

### 6.2.1 goal_ddInit - Register GOAL dd API in GOAL (appl_init)

**Purpose:** Initialize usage of dd component

This function registers the GOAL dd specific API in GOAL and must be called in appl_init. It returns a GOAL_STATUS_T status and has no parameters.

**Function Prototype:**

```
GOAL_STATUS_T goal_ddInit (
    void
);
```

**Example:**

```
GOAL_STATUS_T appl_init( void
)
{
    GOAL_STATUS_T res;  /**< GOAL result */

        /* initialize GOAL dd API */
        res = goal_ddInit();
        if (GOAL_RES_ERR(res)) {
            goal_logErr("failed to initialize GOAL dd API");
        }
    return res;
}
```

### 6.2.2 goal_ddNew - Register GOAL dd API in GOAL (appl_setup)

**Purpose**: Create a new goal dd instance

This function creates an instance of GOAL dd in GOAL and must be called in appl_setup. A valid instance is requirement for using the GOAL dd API.
It returns a GOAL_STATUS_T status and has the following parameters.

| Parameter | Description |
|---|---|
| GOAL_DD_T **ppHdl | returned GOAL dd instance handle |
| uint32_t bitmaskFeatures | initial instance features to be enabled |

Table 28 goal_ddNew parameters

The parameter bitmaskFeatures is a bitmask which disables single features of GOAL dd if set:

| Bit | Feature |
|---|---|
| 0 | disable HELLO request (used for device scan detection) |
| 1 | disable WINK command |
| 2 | disable GETLIST command (read list of available CM variables) |
| 3 | disable GETCONFIG command (read cm variables value) |
| 4 | disable SETCONFIG command (write cm variables value) |

| | |
|---|---|
| 5 | disable SETIP command (configure IP through GOAL dd |

Table 29 feature bitmask parameter

**Function Prototype:**

```
GOAL_STATUS_T goal_ddNew(
    GOAL_DD_T **ppHdlDd,                        /**< DD handle */
    uint32_t bitmaskFeatures                    /**< initial features */
);
```

**Example:**

```
static GOAL_DD_T *pHdlDd;   /**< GOAL dd handle */

GOAL_STATUS_T appl_setup(
    void
)
{
    GOAL_STATUS_T res;  /**< GOAL result */
        /* create GOAL dd instance */
        res = goal_ddNew(&pHdlDd, GOAL_DD_FEAT_ALL);
        if (GOAL_RES_ERR(res)) {
            goal_logErr("failed to create GOAL dd instance");
        }
    return res;
}
```

### 6.2.3 goal_ddCustomerIdSet

**Purpose:** Configure the customer id of GOAL dd instance

This function configures the customer Id of the given GOAL dd instance. The customer Id is a property of the underlying protocol which is contained in each request using GOAL dd. There is a special customer Id whith value of zero, which causes every request to be processed. If the customer Id is not equal to zero, a request will only be processed if the customer Id of the request equals the customer Id of the GOAL dd instance.
The customer Id is stored in non-volatile memory.

| Parameter | Description |
|---|---|
| GOAL_DD_T *pHdl | GOAL dd instance handle |
| uint32_t customerId | instance customer Id |

Table 30 goal_ddCustomerIdSet parameters

**Function Prototype:**

```
GOAL_STATUS_T goal_ddCustomerIdSet(
    GOAL_DD_T *pHdlDd,                          /**< dd handle */
    uint32_t cid                                /**< customer ID */
);
```

**Example:**

```
/* configure DD properties */
res = goal_ddCustomerIdSet(pHdlDd, APPL_DD_CUSTOMER_ID);
if (GOAL_RES_ERR(res)) {
    goal_logErr("failed to configure DD customer id");
}
```

### 6.2.4 goal_ddModuleNameSet

**Purpose:** Configure the name of GOAL dd instance

This function configures the module name of the given GOAL dd instance. The module Id is a property of the device stored in non-volatile memory. It is used my the management tool for device naming. The module name length is limited to 20 bytes.

| Parameter | Description |
|---|---|
| GOAL_DD_T *pHdl | GOAL dd instance handle |
| uint8_t *str | instance module name |

Table 31 goal_ddModuleNameSet parameters

**Function Prototype:**

```
GOAL_STATUS_T goal_ddModuleNameSet(
    GOAL_DD_T *pHdlDd,                          /**< dd handle */
    uint8_t *str                                /**< module name */
);
```

**Example:**

```
res = goal_ddModuleNameSet(pHdlDd, APPL_DD_MODULE_NAME);
if (GOAL_RES_ERR(res)) {
    goal_logErr("failed to configure DD module name");
}
```

### 6.2.5 goal_ddFeaturesSet

**Purpose:** Configure the features of the GOAL dd instance

This function configures the features to be disabled for the given GOAL dd instance. This property is stored in the device stored in non volatile memory.

| Parameter | Description |
|---|---|
| GOAL_DD_T *pHdl | GOAL dd instance handle |
| uint32_t bitmaskFeatures | instance features bitmask |

Table 32 goal_ddFeaturesSet parameters

For parameter description see function “goal_ddNew”.

**Function Prototype:**

```
GOAL_STATUS_T goal_ddFeaturesSet(
    GOAL_DD_T *pHdlDd,                          /**< dd handle */
```

```
    uint32_t bitmaskFeatures                    /**< bitmask with feature disable b
its set */
);
```

**Example:**

```
res = goal_ddFeaturesSet(pHdlDd, APPL_DD_FEATURES);
if (GOAL_RES_ERR(res)) {
    goal_logErr("failed to configure DD features");
}
```

### 6.2.6 goal_ddCallbackReg

**Purpose:** Configure callback for GOAL dd instance

This function registers a callback to the given GOAL dd instance.
Type of the callback:

```
typedef GOAL_STATUS_T (* GOAL_DD_FUNC_CB_T)(
    struct GOAL_DD_T *pHdlDd,                   /**< dd handle */
    GOAL_DD_CB_ID_T cbId,                       /**< callback id */
    GOAL_DD_CB_DATA_T *pCbData                  /**< callback data */
);
```

**Function Prototype:**

```
static GOAL_STATUS_T ddCallback(
    GOAL_DD_T *pHdlDd,                          /**< dd handle */
    GOAL_DD_CB_ID_T cbId,                       /**< callback ID */
    GOAL_DD_CB_DATA_T *pCbData                  /**< callback data */
);
```

**Example:**

```
/****************************************************************************/
/** Application Setup
 *
 * This function must setup all used protocol stacks.
 *
 */
GOAL_STATUS_T appl_setup(
    void
)
{
    GOAL_STATUS_T res;                          /* result */

    res = goal_ddCallbackReg(pHdlDd, (GOAL_DD_FUNC_CB_T) ddCallback);

    return res;
}


/****************************************************************************/
/** goal dd callback
 *
 */
 static GOAL_STATUS_T ddCallback(
    GOAL_DD_T *pHdlDd,                          /**< dd handle */
    GOAL_DD_CB_ID_T cbId,                       /**< callback ID */
```

```
    GOAL_DD_CB_DATA_T *pCbData                  /**< callback data */
)
{
    UNUSEDARG(pHdlDd);
    UNUSEDARG(pCbData);

    switch (cbId) {
        case GOAL_DD_CB_ID_WINK:
            goal_logInfo("Blink command received");
            break;
        default:
            break;
    }

    return GOAL_OK;
}
```

### 6.2.7 goal_ddSessionFeatureActivate

**Purpose:** Temporalily activation of features of GOAL dd instance

This function termporarily enabled features for the given GOAL dd instance. This property is *NOT* stored in non volatile memory.

| Parameter | Description |
|---|---|
| GOAL_DD_T *pHdl | GOAL dd instance handle |
| uint32_t bitmaskFeatures | instance features bitmask |

Table 33 goal_ddSessionFeatureActivation parameters

ATTENTION: The parameter bitmaskFeatures here is used iverted to the function for permanent configuration of the features, thus a bit set here, enabled the given feature.

**Function Prototype:**

```
GOAL_STATUS_T goal_ddSessionFeatureActivate(
    GOAL_DD_T *pHdlDd,                          /**< dd handle */
    uint32_t bitmaskFeatures                    /**< bitmask with feature enable bi
ts set */
);
```

**Example:**

```
/* temporarily enable capability to respond to hello requests (device detection) */
res = goal_ddSessionFeatureActipHdlDd, GOAL_DD_FEAT_HELLO);
```

### 6.2.8 goal_ddFilterAdd

**Purpose:** Limit access to cm variables

By default an external application as the Management Tool has total access to all CM variables of the device. This is a handy feature for development, but for production purpose one wants to limit access to only the variables that are required for minimal functionality using the Management

Tool. Therefore filters were introduced, which do this task. Following filters are predefined:

| Filter ID | Filter Name |
|---|---|
| 0 | GOAL_DD_ACCESS_FILTER_SET_ALL |
| 1 | GOAL_DD_ACCESS_FILTER_SET_BASIC |
| 2 | GOAL_DD_ACCESS_FILTER_SET_HIDDEN |

Table 34 goal_ddFilterAdd filter sets

| Filter ID | Filter Actions | Purpose |
|---|---|---|
| 0 | Full access granted to all variables | Development |
| 1 | Read Access to all variables of the NET module (IP settings), Read Access to all variables of the ETH module (MAC, status), Full access to all variables of the LM module (logging) | Production Code with minimal support of the Management Tool |
| 2 | Disables read access to the web server authentication strings | Production Code |

Table 35 goal_ddFilterAdd predefined filters

### 6.2.8.1 definition of filter GOAL_DD_ACCESS_FILTER_SET_ALL

```
/**< complete access */
static GOAL_DD_VAR_T ddAccessAll[] = {
    {
        .pNext = NULL,
        .modId = GOAL_DD_VAR_ALL,
        .varId = GOAL_DD_VAR_ALL,
        .access = (1 << GOAL_DD_ACCESS_READ) | (1 << GOAL_DD_ACCESS_WRITE)
    }
};
```

### 6.2.8.2 definition of filter GOAL_DD_ACCESS_FILTER_SET_BASIC

```
/**< list of variables that are required for basic functionality */
static GOAL_DD_VAR_T ddAccessListBasic[] = {
    {   /* read access to network settings */
        .pNext = (struct GOAL_DD_VAR_T *) &ddAccessListBasic[1],
        .modId = GOAL_ID_NET,
        .varId = GOAL_DD_VAR_ALL,
```

```
            .access = (1 << GOAL_DD_ACCESS_READ)
        },
        {   /* read access to ethernet properties */
            .pNext = (struct GOAL_DD_VAR_T *) &ddAccessListBasic[2],
            .modId = GOAL_ID_ETH,
            .varId = GOAL_DD_VAR_ALL,
            .access = (1 << GOAL_DD_ACCESS_READ)
        },
        {   /* access to logs */
            .pNext = NULL,
            .modId = GOAL_ID_LM,
            .varId = GOAL_DD_VAR_ALL,
            .access = (1 << GOAL_DD_ACCESS_READ) | (1 << GOAL_DD_ACCESS_WRITE)
        }
    };
```

### 6.2.8.3 definition of filter GOAL_DD_ACCESS_FILTER_SET_HIDDEN

```
    static GOAL_DD_VAR_T ddAccessListHidden[] = {
        {   /* disable read access to HTTP user level 0 */
            .pNext = (struct GOAL_DD_VAR_T *) &ddAccessListHidden[1],
            .modId = GOAL_ID_HTTP,
            .varId = 2, /* USERLEVEL0 */
            .access = (1 << GOAL_DD_ACCESS_WRITE)
        },
        {   /* disable read access to HTTP user level 1 */
            .pNext = (struct GOAL_DD_VAR_T *) &ddAccessListHidden[2],
            .modId = GOAL_ID_HTTP,
            .varId = 3, /* USERLEVEL1 */
            .access = (1 << GOAL_DD_ACCESS_WRITE)
        },
        {   /* disable read access to HTTP user level 2 */
            .pNext = (struct GOAL_DD_VAR_T *) &ddAccessListHidden[3],
            .modId = GOAL_ID_HTTP,
            .varId = 4, /* USERLEVEL2 */
            .access = (1 << GOAL_DD_ACCESS_WRITE)
        },
        {   /* disable read access to HTTP user level 3 */
            .pNext = NULL,
            .modId = GOAL_ID_HTTP,
            .varId = 5, /* USERLEVEL3 */
            .access = (1 << GOAL_DD_ACCESS_WRITE)
        },
    };
```

**Function Prototype:**

```
    GOAL_STATUS_T goal_ddFilterAdd(
        GOAL_DD_T *pHdlDd,                          /**< dd handle */
        GOAL_DD_ACCESS_FILTER_SET_T setId           /**< set id */
    );
```

**Example:**

```
        /* enable for full access */
    #if 0
        res = goal_ddFilterAdd(pHdlDd, GOAL_DD_ACCESS_FILTER_SET_ALL);
        if (GOAL_RES_ERR(res)) {
            goal_logErr("failed to set dd access filter");
        }
    #endif
```

```
    res = goal_ddFilterAdd(pHdlDd, GOAL_DD_ACCESS_FILTER_SET_HIDDEN);
    if (GOAL_RES_ERR(res)) {
        goal_logErr("failed to set dd access filter");
    }
```

## 6.3 profinet stack

see profinet documentation

## 6.4 ethernet ip stack

see ethernet ip documentation

## 6.5 web server

see GOAL programmer's manual

## 6.6 networking

### 6.6.1 goal_netRpcInit

**Purpose:** Initialize RPC functionality for networking

If networking functionality (IP settings, UDP channel, TCP channel) is required, this function needs to be called during application initialization.

**Function Prototype:**

```
GOAL_STATUS_T goal_netRpcInit(
    uint32_t id                                /**< id for MA instance */
)
```

**Example:**

```
/*************************************************************************/
/** Application Init
 *
 * Initialize the net stack
 */
GOAL_STATUS_T appl_init(
    void
)
{
    GOAL_STATUS_T res;                         /* result */

    /* initialize goal net */
    res = goal_netRpcInit(GOAL_NET_ID_DEFAULT);
    if (GOAL_RES_ERR(res)) {
        goal_logErr("Initialization of goal net RPC failed");
    }

    return res;
}
```

## 6.7 goal_maNetOpen - open network channel

**Purpose**:
Open the network media adapter for usage.

**Function Prototype:**

```
GOAL_STATUS_T goal_maNetOpen(
    uint32_t id,                            /**< id of NET handler to use */
    GOAL_MA_NET_T **ppNetHdl                /**< pointer to store NET handler *
/
);
```

**Example:**

```
/*************************************************************************/
/** Application Setup
 *
 * This function is called by the GOAL init-stage system to open UDP channels.
 *
 * API functions from earlier stages are allowed to be used here.
 */
GOAL_STATUS_T appl_setup(
    void
)
{
    GOAL_STATUS_T res;                      /* result */
    GOAL_MA_NET_T *pMaNet;                  /* net ma handle */

    res = goal_maNetOpen(GOAL_NET_ID_DEFAULT, &pMaNet);
    if (GOAL_RES_ERR(res)) {
        goal_logErr("error opening network MA");
    }

    return res;
}
```

### 6.7.1 goal_maNetClose - close network

**Purpose:**
Close the network MA

**Function Prototype:**

```
GOAL_STATUS_T goal_maNetClose(
    GOAL_MA_NET_T *pNetHdl                  /**< pointer to store NET handler *
/
);
```

### 6.7.2 goal_maNetGetById - get network MA handle

**Purpose:** Get the network MA handle which was previously open for usage

**Function Prototype:**

```
GOAL_STATUS_T goal_maNetGetById(
    GOAL_MA_NET_T **ppHdlMaNet,                 /**< NET handle ref ptr */
    uint32_t id                                 /**< MA id */
);
```

**Example:**

```
res = goal_maNetGetById(&pMaNet, GOAL_NET_ID_DEFAULT);
if (GOAL_RES_ERR(res)) {
     goal_logErr("error getting network MA");
}
```

### 6.7.3 goal_maNetIpSet – set ip address

**Purpose:** Set the network interface IP address

**Function Prototype:**

```
GOAL_STATUS_T goal_maNetIpSet(
    GOAL_MA_NET_T *pNetHdl,                     /**< pointer to store NET handler *
/
    uint32_t addrIp,                            /**< IP address */
    uint32_t addrMask,                          /**< subnet mask */
    uint32_t addrGw,                            /**< gateway */
    GOAL_BOOL_T flgTemp                         /**< temporary IP config flag */
);
```

**Example:**

```
/* set IP address */
ip = MAIN_APPL_IP;
nm = MAIN_APPL_NM;
gw = MAIN_APPL_GW;
res = goal_maNetIpSet(pMaNet, ip, nm, gw, GOAL_FALSE);
if (GOAL_RES_ERR(res)) {
    goal_logErr("error while setting IP address");
    return res;
}
```

## 6.8 tcp channel

### 6.8.1 goal_maChanTcpOpen - open the tcp channel MA

**Purpose:** Opens the networking MA for further usage. This needs to be done once at application startup.

**Function Prototype:**

```
GOAL_STATUS_T goal_maChanTcpOpen(
    uint32_t id,                                /**< MA id */
    GOAL_MA_CHAN_TCP_T **ppHdlMaChanTcp         /**< CHAN_TCP handle ref ptr */
);
```

**Example:**

```
res = goal_maChanTcpOpen(GOAL_NET_ID_DEFAULT, &pMaTcp);
if (GOAL_RES_ERR(res)) {
    goal_logErr("error getting tcp MA");
    return res;
}
```

### 6.8.2 goal_maChanTcpNew - create a new tcp channel

**Purpose:** This function creates a new tcp channel.

**Function Prototype:**

```
GOAL_STATUS_T goal_maChanTcpNew(
    GOAL_MA_CHAN_TCP_T *pChanTcpHdl,            /**< pointer to store CHAN_TCP hand
ler */
    GOAL_NET_CHAN_T **ppChanHandle,             /**< pointer to channel handle */
    GOAL_NET_CHAN_T *pChanOut,                  /**< pointer to channel handle for
output */
    GOAL_NET_ADDR_T *pAddr,                     /**< remote address */
    GOAL_NET_TYPE_T type,                       /**< connection type */
    GOAL_MA_CHAN_TCP_CB_T callback              /**< channel callback */
);
```

**Example:**

```
/* register TCP server */
GOAL_MEMSET(&addr, 0, sizeof(GOAL_NET_ADDR_T));
addr.localPort = (uint16_t) (MAIN_APPL_TCP_PORT + cnt);
res = goal_maChanTcpNew(pMaTcp, &pChan, NULL, &addr, GOAL_NET_TCP_LISTENER,
                        tcpCallback);
if (GOAL_OK != res) {
    goal_logErr("error while opening TCP server channel on port %"FMT_u32,
                (uint32_t) MAIN_APPL_TCP_PORT + cnt);
    return res;
}
```

### 6.8.3 goal_maChanTcpActive - activate a created tcp channel

**Purpose:** Activate a previously created tcp channel. Once it is activated, it established connection or accepts incoming connection requests.

**Function Prototype:**

```
GOAL_STATUS_T goal_maChanTcpActivate(
    GOAL_MA_CHAN_TCP_T *pChanTcpHdl,            /**< pointer to store CHAN_TCP hand
ler */
    GOAL_NET_CHAN_T *pChanHandle                /**< channel handle */
);
```

**Example:**

```
/* activate channel */
res = goal_maChanTcpActivate(pMaTcp, pChanTcp);
if (GOAL_OK != res) {
    goal_logErr("error while enabling TCP channel");
```

```
    return;
}
```

### 6.8.4 goal_maChanTcpSetNonBlocking - set channel to non blocking

**Purpose:** Set socket to non blocking mode for reading

**Function Prototype:**

```
GOAL_STATUS_T goal_maChanTcpSetNonBlocking(
    GOAL_MA_CHAN_TCP_T *pChanTcpHdl,            /**< pointer to store CHAN_TCP hand
ler */
    GOAL_NET_CHAN_T *pChanHandle,               /**< channel handle */
    GOAL_BOOL_T flgOption                       /**< non blocking state */
);
```

**Example:**

```
/* set TCP channel to non-blocking */
res = goal_maChanTcpSetNonBlocking(pMaTcp, pChanTcp, GOAL_TRUE);
if (GOAL_OK != res) {
    goal_logErr("error while setting TCP channel to non-blocking");
    return;
}
```

### 6.8.5 goal_maChanTcpGetRemoteAddr - get remote address of tcp channel

**Purpose:** Get the ip address of the remote end point of the tcp channel.

**Function Prototype:**

```
GOAL_STATUS_T goal_maChanTcpGetRemoteAddr(
    GOAL_MA_CHAN_TCP_T *pChanTcpHdl,            /**< pointer to store CHAN_TCP hand
ler */
    GOAL_NET_CHAN_T *pChanHandle,               /**< channel handle */
    GOAL_NET_ADDR_T *pAddr                      /**< remote address */
);
```

**Example:**

```
/* get IP Address of remote node */
res = goal_maChanTcpGetRemoteAddr(pMaTcpHdl, pChan, &remote);
if (GOAL_RES_ERR(res)) {
    goal_logErr("Failed to get Remote Address for socket %p", (void *) pChan);
    return;
}
```

### 6.8.6 goal_maChanTcpSend - send data through tcp channel

**Purpose:** Send data to a previously opened TCP channel.

**Function Prototype:**

```
GOAL_STATUS_T goal_maChanTcpSend(
    GOAL_MA_CHAN_TCP_T *pChanTcpHdl,            /**< pointer to store CHAN_TCP hand
```

```
ler */
    GOAL_NET_CHAN_T *pChanHandle,                /**< channel handle */
    GOAL_BUFFER_T *pBuf                          /**< buffer with data to send */
);
```

**Example:**

```
/* echo message */
goal_maChanTcpSend(pMaTcpHdl, pChan, pBuf);
```

## 6.9 udp channel

### 6.9.1 goal_maChanUdpOpen - open the udp channel MA

**Purpose:** Open the udp channel ma. This needs to be done once at application startup.

**Function Prototype:**

```
GOAL_STATUS_T goal_maChanUdpOpen(
    uint32_t id,                                 /**< MA id */
    GOAL_MA_CHAN_UDP_T **ppHdlMaChanUdp          /**< CHAN_UDP handle ref ptr */
);
```

**Example:**

```
/* open udp channel MA and create new channel */
res = goal_maChanUdpOpen(GOAL_NET_ID_DEFAULT, &pMaChanUdp);
if (GOAL_RES_OK(res)) {
    GOAL_MEMSET(&addr, 0, sizeof(GOAL_NET_ADDR_T));
    addr.localPort = MAIN_APPL_UDP_PORT_1;
    res = goal_maChanUdpNew(pMaChanUdp, &pChan1, NULL, &addr, GOAL_NET_UDP_SERVER,
udpCallback);
}
```

### 6.9.2 goal_maChanUdpGetById - get the udp channel MA handle

Purpose: This function gets the handle of the udp channel ma which was previously opened.

Function Prototype:

```
GOAL_STATUS_T goal_maChanUdpGetById(
    GOAL_MA_CHAN_UDP_T **ppHdlMaChanUdp,         /**< CHAN_UDP handle ref ptr */
    uint32_t id                                  /**< MA id */
);
```

### 6.9.3 goal_maChanUdpNew - create a new udp channel

**Purpose:** Create a new udp channel.

**Function Prototype:**

```
GOAL_STATUS_T goal_maChanUdpNew(
    GOAL_MA_CHAN_UDP_T *pChanUdpHdl,             /**< pointer to store CHAN_UDP hand
ler */
    GOAL_NET_CHAN_T **ppChanHandle,              /**< pointer to channel handle */
```

```
    GOAL_NET_CHAN_T *pChanOut,                  /**< pointer to channel handle for
output */
    GOAL_NET_ADDR_T *pAddr,                     /**< remote address */
    GOAL_NET_TYPE_T type,                       /**< connection type */
    GOAL_MA_CHAN_UDP_CB_T callback              /**< channel callback */
);
```

**Example:**

```
if (GOAL_RES_OK(res)) {
     GOAL_MEMSET(&addr, 0, sizeof(GOAL_NET_ADDR_T));
     addr.localPort = MAIN_APPL_UDP_PORT_1;
     res = goal_maChanUdpNew(pMaChanUdp, &pChan1, NULL, &addr, GOAL_NET_UDP_SERVER,
                             udpCallback);
 }
```

### 6.9.4 goal_maChanUdpClose - close the udp channel MA

**Purpose:** Close an existing channel

**Function Prototype:**

```
GOAL_STATUS_T goal_maChanUdpClose(
    GOAL_MA_CHAN_UDP_T *pChanUdpHdl,            /**< pointer to store CHAN_UDP hand
ler */
    GOAL_NET_CHAN_T *pChanHandle                /**< pointer to channel handle */
);
```

### 6.9.5 goal_maChanUdpSetNonBlocking - set the opened channel to non blocking access

**Purpose:** Set a channel to non blocking operation.

**Function Prototype:**

```
GOAL_STATUS_T goal_maChanUdpSetNonBlocking(
    GOAL_MA_CHAN_UDP_T *pChanUdpHdl,            /**< pointer to store CHAN_UDP hand
ler */
    GOAL_NET_CHAN_T *pChanHandle,               /**< channel handle */
    GOAL_BOOL_T flgOption                       /**< option value */
);
```

**Example:**

```
res = goal_maChanUdpSetNonBlocking(pMaChanUdp, pChan2, GOAL_TRUE);
if (GOAL_OK != res) {
    goal_logErr("error while setting UDP channel to non-blocking");
    return res;
}
```

### 6.9.6 goal_maChanUdpSetBroadcast - set the opened udp channel to broadcast operation

**Purpose:** Set a channel to broadcast.

**Function Prototype:**

```
GOAL_STATUS_T goal_maChanUdpSetBroadcast(
    GOAL_MA_CHAN_UDP_T *pChanUdpHdl,            /**< pointer to store CHAN_UDP hand
ler */
    GOAL_NET_CHAN_T *pChanHandle,               /**< channel handle */
    GOAL_BOOL_T flgOption                       /**< option value */
);
```

**Example:**

```
/* enable broadcast reception */
res = goal_maChanUdpSetBroadcast(pMaChanUdp, pChan1, GOAL_TRUE);
if (GOAL_RES_ERR(res)) {
    goal_logErr("error while setting UDP channel to receive broadcasts");
    return res;
}
```

### 6.9.7 goal_maChanUdpGetRemoteAddr - get remote address of the udp channel

**Purpose:** Get the remote address of a udp channel, thus the address it received data from.

**Function Prototype:**

```
GOAL_STATUS_T goal_maChanUdpGetRemoteAddr(
    GOAL_MA_CHAN_UDP_T *pChanUdpHdl,            /**< pointer to store CHAN_UDP hand
ler */
    GOAL_NET_CHAN_T *pChanHandle,               /**< channel handle */
    GOAL_NET_ADDR_T *pAddr                      /**< remote address */
);
```

### 6.9.8 goal_maChanUdpActivate - acticate a udp channel

**Purpose:** Activate a channel

**Function Prototype:**

```
GOAL_STATUS_T goal_maChanUdpActivate(
    GOAL_MA_CHAN_UDP_T *pChanUdpHdl,            /**< pointer to store CHAN_UDP hand
ler */
    GOAL_NET_CHAN_T *pChanHandle                /**< channel handle */
);
```

**Example:**

```
res = goal_maChanUdpActivate(pMaChanUdp, pChan2);
if (GOAL_RES_ERR(res)) {
    goal_logErr("error while enabling UDP channel");
    return res;
}
```

### 6.9.9 goal_maChanUdpSend - send data to the udp channel

**Purpose:** Send data to an open udp channel.

### Function Prototype:

```
GOAL_STATUS_T goal_maChanUdpSend(
    GOAL_MA_CHAN_UDP_T *pChanUdpHdl,            /**< pointer to store CHAN_UDP hand
ler */
    GOAL_NET_CHAN_T *pChanHandle,               /**< channel handle */
    GOAL_BUFFER_T *pBuf                         /**< buffer with data to send */
);
```

### Example:

```
/* echo message */
goal_maChanUdpSend(pMaChanUdp, pChan, pBuf);
```

# 7 Examples

## 7.1 01_pnio_io_mirror

### 7.1.1 Purpose

This example implements a PROFINET slave including a http management interface.

### 7.1.2 Configuration

By default no IP address is set. Using a configuration tool (e.g. Managment Tool) a valid configuration must be provided.

The device configuration is following:

| Slot | Subslot | Module |
|---|---|---|
| 1 | 1 | „I Signed8“ – 1 Byte Input data |
| 2 | 1 | „O Signed8“ – 1 Byte Output data |
| 3 | 1 | „I Signed16“ – 2 Byte Input data |
| 4 | 1 | „O Signed16“ – 2 Byte Output data |

Table 36 Example configuration

| Record Index | Data |
|---|---|
| 0x7000 | 3 byte of parameter data (read/write) |

The GSDML file for this example is located beside the application code:
***"goal\appl\2015013_SoM\ac\01_pnio_io_mirror\gsdml"***

### 7.1.3 Usage Hints

This example shows how to implement a basic profinet slave. It also shows handling of process data by mapping the output data of module „O Signed8“ to the input data of module „I Signed8“ and mapping output data of module „O Signed16“ to the input data of the module “I Signed16”.

Beside that a management interface under HTTP://<DEVICE-IP> is provided, where under “Device Management” access from the Management Tool or Firmware update can be enabled and disabled.

Beside that the example shows implementation of application specific record data (parameters).

This example also controls the LEDs on the shield module.

## 7.2 01_pnio_io_mirror_renesas

This example is equal to example 01_pnio_io_mirror, however it uses different Vendor and Device id (Renesas). The GSDML file which contains “Renesas” must be utilized.

## 7.3 02_eip_io_data

### 7.3.1 Purpose

This example implements a EtherNet/IP slave including a http management interface.

### 7.3.2 Configuration

This example requires a DHCP server to obtain an IP address.

The device configuration is following:

| Assembly ID | Size | Properties |
|---|---|---|
| 150 | 32 | Output Data |
| 100 | 32 | Input Data |
| 151 | 10 | Configuration Data |

Table 37 Example configuration

The EDS file for this example is located beside the application code:
***"goal\appl\2015013_SoM\ac\02_eip_io_data"***

### 7.3.3 Usage Hints

This example shows how to implement a basic EtherNet/IP slave. It also shows handling of process data by mapping the output data of Assembly 150 to the input data of Assembly 100.

Beside that a management interface under HTTP://<DEVICE-IP> is provided, where under “Device Management” access from the Management Tool or Firmware update can be enabled and disabled.

This example also controls the LEDs on the shield module.

## 7.4 02_eip_io_data_renesas

This example is equal to example 02_eip_io_data, however it uses different Vendor and Device id (Renesas). The EDS file which contains “Renesas” must be utilized.

## 7.5 05_pnio_01_simple_io

### 7.5.1 Purpose

This example implements a PROFINET slave.

### 7.5.2 Configuration

By default no IP address is set. Using a configuration tool (e.g. Managment Tool) a valid configuration must be provided.

The device configuration is following:

| Slot | Subslot | Module |
|---|---|---|
| 1 | 1 | „64 bytes Input“ – 64 byte input data |
| 2 | 1 | „64 bytes Output“ – 64 Byte Output data |

Table 38 Example configuration

The GSDML file for this example is located beside the application code:
***"goal\appl\2015013_SoM\ac\05_pnio_01_simple_io\gsdml"***

### 7.5.3 Usage Hints

This example shows how to implement a basic profinet slave. It also shows handling of process data by mapping the output data of module „64 bytes output“ to the input data of module „64 bytes input“.

Beside that a management interface under HTTP://<DEVCIE-IP> is provided, where under “Device Management” access from the Management Tool or Firmware update can be enabled and disabled.

## 7.6 05_pnio_01_simple_io_renesas

This example is equal to example 05_pnio_01_simple_io, however it uses different Vendor and Device id (Renesas). The GSDML file which contains “Renesas” must be utilized.

## 7.7 06_eip_io_data_static_ip

This example implements a EtherNet/IP slave including a http management interface.

### 7.7.1 Configuration

This example configures a static IP address at startup, which is 192.168.0.100. Any reconfiguration with the management tool will be overwritten after restart.

The device configuration is following:

| Assembly ID | Size | Properties |
|---|---|---|
| 150 | 32 | Output Data |
| 100 | 32 | Input Data |
| 151 | 10 | Configuration Data |

Table 39 Example configuration

The EDS file for this example is located beside the application code:
***"goal\appl\2015013_SoM\ac\02_eip_io_data"***

### 7.7.2 Usage Hints

This example shows how to implement a basic EtherNet/IP slave. It also shows handling of process data by mapping the output data of Assembly 150 to the input data of Assembly 100.

Beside that a management interface under HTTP://<DEVICE-IP> is provided, where under “Device Management” access from the Management Tool or Firmware update can be enabled and disabled.

This example also controls the LEDs on the shield module.

## 7.8 07_pnio_dsn

### 7.8.1 Purpose

This example implements a PROFINET slave using the PROFINET design tool. The project file is supplied.

### 7.8.2 Configuration

By default no IP address is set. Using a configuration tool (e.g. Managment Tool) a valid configuration must be provided.

The device configuration is following:

| Slot | Subslot | Module |
|---|---|---|
| 1 | 1 | „I Signed8“ – 1 Byte Input data |
| 2 | 1 | „O Signed8“ – 1 Byte Output |

| | | |
|---|---|---|
| | | data |
| 3 | 1 | „I Signed16“ – 2 Byte Input data |
| 4 | 1 | „O Signed16“ – 2 Byte Output data |

Table 40 Example configuration

The GSDML file for this example is located beside the application code:
***"goal\appl\2015013_SoM\ac\07_pnio_dsn\gsdml"***

The Design tool project file is located beside the generated gsdml file:
***"goal\appl\2015013_SoM\ac\07_pnio_dsn\gsdml\SoM.dsntool"***

### 7.8.3 Usage Hints

This example shows how to implement a basic profinet slave. It only contains the generated stub code from the design tool, application functions need tob e added (refer to example 01_pnio_io_mirror forthat).

## 7.9 10_pnio_process_alarm

### 7.9.1 Purpose

This example implements a PROFINET slave and shows generation of process alarms.

### 7.9.2 Configuration

By default no IP address is set. Using a configuration tool (e.g. Managment Tool) a valid configuration must be provided.

The device configuration is following:

| Slot | Subslot | Module |
|---|---|---|
| 1 | 1 | „64 bytes Input“ – 64 byte input data |
| 2 | 1 | „64 bytes Output“ – 64 Byte Output data |

Table 41 Example configuration

The GSDML file for this example is located beside the application code:
***"goal\appl\2015013_SoM\ac\10_pnio_process_alarm\gsdml"***

### 7.9.3 Usage Hints

This example continuously generates process alarms to the PLC. Thus this example cannot be used with the current version of the Management Tool. A profinet capable PLC is required to handle the process alarms.

## 7.10 http_01_get

### 7.10.1 Purpose

This example shows usage of the web server. It shows implementing GET-request support.

### 7.10.2 Configuration

This example uses the configured IP address of the cc module. Check the Management Tool to get the IP address.

### 7.10.3 Usage Hints

Open the URL HTTP://<DEVICE-IP>:8081, where a simple web page is shown.

## 7.11 http_02_post

### 7.11.1 Purpose

This example shows usage of the web server. It shows implementing POST-request support.

### 7.11.2 Configuration

This example uses the configured IP address of the cc module. Check the Management Tool to get the IP address.

### 7.11.3 Usage Hints

Open the URL HTTP://<DEVICE-IP>:8080, where a simple web page is shown. There a small amount of data can be uploaded to the application.

## 7.12 http_03_list_res

### 7.12.1 Purpose

This example shows usage of the web server. It shows supporting of hirachical urls.

### 7.12.2 Configuration

This example uses the configured IP address of the cc module. Check the Management Tool to get the IP address.

### 7.12.3 Usage Hints

Open the URL HTTP://<DEVICE-IP>:8080, where a simple web page is shown. It provides several sub pages shown on the main page.

## 7.13 http_04_auth

This example shows usage of the web server. It shows supporting of basic authentication.

### 7.13.1 Configuration

This example uses the configured IP address of the cc module. Check the Management Tool to get the IP address.

### 7.13.2 Usage Hints

This example creates 4 pages with separate authentication data:

| URL | Credentials |
|---|---|
| HTTP://<DEVICE-IP>:8080 | index:level0 |
| HTTP://<DEVICE-IP>:8080/page1.html | page1:level1 |
| HTTP://<DEVICE-IP>:8080/page2.html | page2:level2 |
| HTTP://<DEVICE-IP>:8080/page3.html | page3:level3 |

When opening one of the listed urls the provides login credentials are required.

**ATTENTION**: This example modifies credentials of authentication level 0. If those settings are stored permanently in the CC module, firmware update will fail with default credentials. Reset settings to default value “” using the Management Tool.

## 7.14 http_05_template_cm

### 7.14.1 Purpose

This example shows usage of the web server. It shows implementing GET-request support with templating for CM variables.

### 7.14.2 Configuration

This example uses the configured IP address of the cc module. Check the Management Tool to get the IP address.

### 7.14.3 Usage Hints

Open the URL HTTP://<DEVICE-IP>:8080, where a simple web page is shown. It demonstrates

template replacement of template usage with CM variable reference.

The string [CM:12, 0] will be replaced with the IP address of the device.

## 7.15 http_06_template_list

This example shows usage of the web server. It shows implementing GET-request support with templating for a list.

### 7.15.1 Configuration

This example uses the configured IP address of the cc module. Check the Management Tool to get the IP address.

### 7.15.2 Usage Hints

Open the URL HTTP://<DEVICE-IP>:8080, where a simple web page is shown. It demonstrates template replacement of template usage with list support.

## 7.16 http_07_template_table

### 7.16.1 Purpose

This example shows usage of the web server. It shows implementing GET-request support with templating for a table.

### 7.16.2 Configuration

This example uses the configured IP address of the cc module. Check the Management Tool to get the IP address.

### 7.16.3 Usage Hints

Open the URL HTTP://<DEVICE-IP>:8080, where a simple web page is shown. It demonstrates template replacement of template usage with table support.

A table of sensors with different sensor values is generated.

## 7.17 net_01_udp_receive

### 7.17.1 Purpose

This example shows usage of the UDP. It shows implementing a echo server on a specific UDP port.

### 7.17.2 Configuration

This example sets the IP address 192.168.0.100/24.

### 7.17.3 Usage Hints

Using netcat the udp server can be tested. Just type some text, and the server will reply this message.

```
! I bit@bit-vm ~ netcat -u 192.168.0.100 1234
5.1s < Fr 18 Jan 2019 12:09:16 UTC
Testing the echo server using UDP!
Testing the echo server using UDP!
```

## 7.18 net_02_tcp_client

### 7.18.1 Purpose

This example shows usage of the TCP as a client. It shows how to connect to a TCP server.

### 7.18.2 Configuration

This example sets the IP address 192.168.0.100/24.
It expects a TCP server on a remote server with IP address 192.168.0.10 on port 1234.

### 7.18.3 Usage Hints

Using netcat a tcp server can be provided. Once the application runs, the client will send the following messages.

```
I bit@bit-vm ~ netcat -l 192.168.0.10 1234
Fr 18 Jan 2019 12:22:13 UTC
TESTSTRING 0TESTSTRING 1TESTSTRING 2TESTSTRING 3TESTSTRING 4⏎
```

## 7.19 net_03_tcp_server

### 7.19.1 Purpose

This example shows usage of the TCP as a server.

### 7.19.2 Configuration

This example sets the IP address 192.168.0.100/24.

### 7.19.3 Usage Hints

Using netcat a connection to the TCP server can be established. Just type some text, and the server will send the following messages.

```
I  bit@bit-vm  ~  netcat 192.168.0.100 1234
Fr 18 Jan 2019 12:22:13 UTC
Testing the echo server using TCP!
Testing the echo server using TDP!
```

# 8 Trouble Shooting

This chapter lists common pitfalls possibly occur when using the SoM examples.

## 8.1 Startup Issues

In case an example application is started but the expected application behaviour is not shown, please check the log:

```
2019-01-18 09:44:45   GOAL_LOG_SEV_INFO        7          [I|goal_miDmPartRegInt:275] part added to 'Write to
CC', pos: 1, len: 2
2019-01-18 09:44:45   GOAL_LOG_SEV_INFO        7          [I|appl_setup:798] TX: Slot 3 Subslot 1 DATA: 1
2019-01-18 09:44:45   GOAL_LOG_SEV_INFO        7          [I|appl_httpSetup:100] setup web server
2019-01-18 09:44:45   GOAL_LOG_SEV_INFO        7          [I|goal_httpNewAc:959] HTTP Application Core
successfully started
2019-01-18 09:44:45   GOAL_LOG_SEV_INFO        7          [I|appl_httpSetup:178] web server setup done
2019-01-18 09:44:45   GOAL_LOG_SEV_INFO        7          [I|appl_setup:822] ccm version : 1.0.0.0
2019-01-18 09:44:45   GOAL_LOG_SEV_INFO        7          [I|appl_setup:823] ccm device : 1
2019-01-18 09:44:45   GOAL_LOG_SEV_INFO        7          [I|appl_setup:825] ccm Serial : b4:e9:a3:00:75:39
2019-01-18 09:44:45   GOAL_LOG_SEV_INFO        7          [I|goal_miMctcRpcSyncLoop:1028] local setup done
2019-01-18 09:44:45   GOAL_LOG_SEV_INFO        7          fixed memory usage: 102096/262144 bytes
2019-01-18 09:44:45   GOAL_LOG_SEV_INFO        7          fixed memory usage: (39%)
2019-01-18 09:47:04   GOAL_LOG_SEV_ERROR       492        [CC_E|goal_miMctcMonitorRx:1250] data channel
offline: MCTC SPI
2019-01-18 09:47:13   GOAL_LOG_SEV_INFO        501        [CC_I|goal_miMctcMonitorRx:1239] data channel
online: MCTC SPI
2019-01-18 09:47:13   GOAL_LOG_SEV_ERROR       501        [CC_E|goal_miMctcRpcSyncLoop:956] sync needs local
reset to proceed
```

If the last log entry regarding „sync needs local reset to proceed“ is shown, the communication controller needs a reset. This can be achieved using the „RST“ button on the shield board.

## 8.2 Connection issues

If the SPI communication is not working at all, this can be seen on the following final log message:

```
2019-01-18 09:47:04   GOAL_LOG_SEV_ERROR       492        [CC_E|goal_miMctcMonitorRx:1250] data channel
offline: MCTC SPI
```

Please check the board connection and the proper execution of the application on the application controller.

## 8.3 IP configuration

If the switch between static IP configuration and DHCP fails after reboot, please check the follow CM variables using the management tool:

| Module ID | Variable ID |
|---|---|
| GOAL_ID_NET | IP |
| GOAL_ID_NET | NETMASK |

| | |
|---|---|
| GOAL_ID_NET | GW |
| GOAL_ID_NET | VALID |
| GOAL_ID_NET | DHCP_ENABLED |

To disable DHCP set the variable DHCP_ENABLED to 0. Make sure variable „VALID“ is set to 1. Upload these settings to the module and save those values permanently. After a reboot DHCP should be disabled.

## 8.4 Downgrade to version 1.0

A downgrade to version 1.0 may fail if the AC application uses the reset input of the ccm module. To successfully do a downgrade to the version 1.0, disable any AC connection and update the module in stand alone mode.

Versions of the firmware after 1.0 mitigate the influence of an external reset during firmware update.

# 9 Targets

## 9.1 RENESAS Synergy

### 9.1.1 Development Environment

- e2studio Version 5.4.0.023
- ssp Version 1.3.0

### 9.1.2 Supported Hardware

#### *9.1.2.1 Renesas S7G2-SK*

https://www.renesas.com/kr/en/products/synergy/hardware/kits/sk-s7g2.html

- SPI Channel: 8

| Function | Pin |
|---|---|
| SPI MISO | IOPORT_PORT_01_PIN_04 |
| SPI MOSI | IOPORT_PORT_01_PIN_05 |
| SPI SCK | IOPORT_PORT_01_PIN_06 |
| SPI CS | IOPORT_PORT_05_PIN_07 |
| RESET | IOPORT_PORT_06_PIN_14 |

Table 42 SPI Pinning used on Renesas S7G2-SK Board

- UART Channel: 2

| Function | Pin |
|---|---|
| RXD | IOPORT_PORT_03_PIN_01 |
| TXD | IOPORT_PORT_03_PIN_02 |

Table 43 UART Pinning used on Renesas S7G2-SK Board

#### *9.1.2.2 Arrow Aris*

https://www.arrow.de/campaigns/aris

- SPI Channel: 0

| Function | Pin |
|---|---|
| SPI MISO | IOPORT_PORT_04_PIN_10 |
| SPI MOSI | IOPORT_PORT_04_PIN_11 |

| | |
|---|---|
| SPI SCK | IOPORT_PORT_04_PIN_12 |
| SPI CS | IOPORT_PORT_02_PIN_03 |
| RESET | IOPORT_PORT_05_PIN_05 |

Table 44 Pinning used on Arrow ARIS Board

- UART Channel: 7

| Function | Pin |
|---|---|
| RXD | IOPORT_PORT_04_PIN_02 |
| TXD | IOPORT_PORT_04_PIN_01 |

Table 45 UART Pinning used on Arrow ARIS Board

### 9.1.3 Adaption for customer hardware

To apply the provided SoM AC examples to a customer hardware, at least the following properties must be adapted to this board.

Since the applications base on the GOAL middleware, the proper way to adapt the example to a new hardware would be to create a board specific ***goal_target_board*** module.

The following chapters show where adaption is required:

#### *9.1.3.1 SPI configuration*

In ***goal_target_board.h*** there is a configuration option for the chip select pin of the SPI interface. This needs to be adapted to the target hardware:

```
/*****************************************************************************/
/* Configuration */
/*****************************************************************************/
#define GOAL_DRV_SPI_CS_PIN       IOPORT_PORT_05_PIN_07 /**< SPI chip select pin */
```

In ***goal_target_board.c*** within the function *goal_tgtBoardInit* the SPI driver is registered.

```
/* register SPI driver */
res = goal_drvSpiSynReg(GOAL_ID_DEFAULT, 8);
if (GOAL_RES_ERR(res)) {
    goal_logErr("failed to register Synergy SPI driver");
    return res;
}
```

The second parameter of *goal_drvSpiSyReg* defines the SPI channel to use.

Configuration of the SPI needs to be done in e2studio:

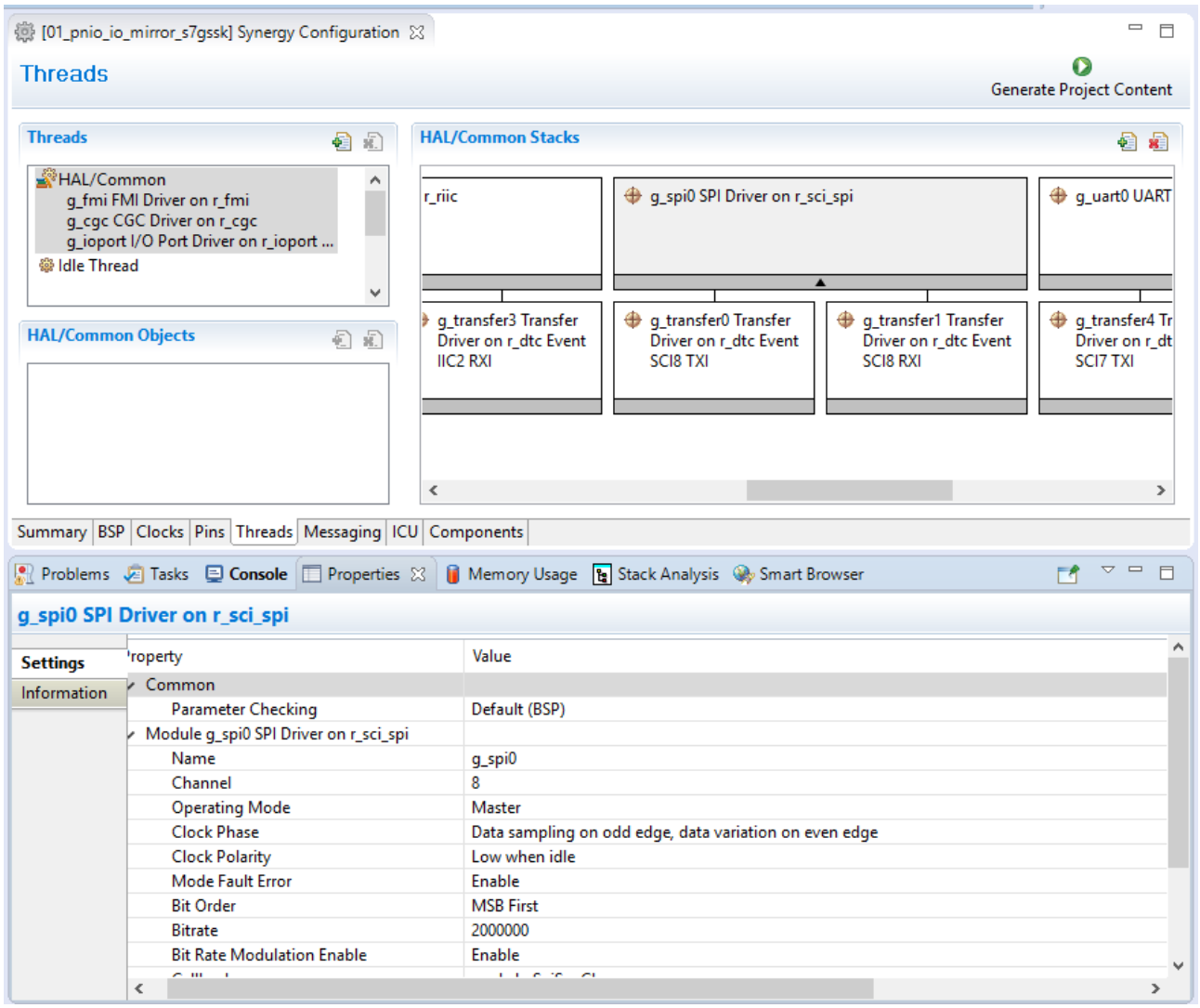


Figure 4 e2studio SPI properties

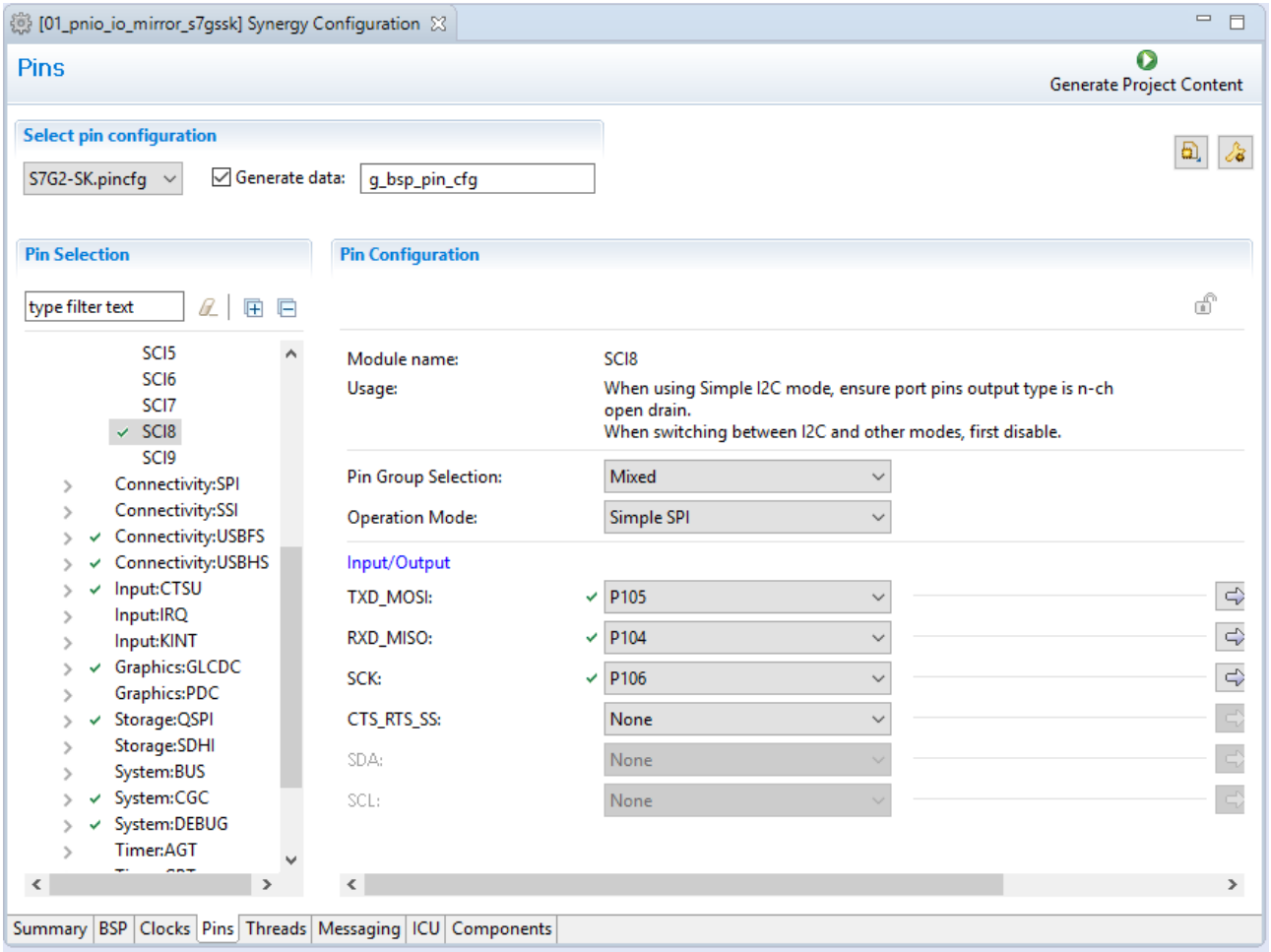

Figure 5 e2studio SPI pinning

### 9.1.3.2 LED control

The goal middleware provides mechanism for LED control. The board files of the example code contain registration of a driver for the SoM shield, which contains several LEDs. The driver, located in "plat\drv\led\iic\synergy7_SoMshield" controls the LEDs on this additional board, which are connected by an I2C bus.

```
#if GOAL_CONFIG_DRV_LED_IIC_SYNERGY7_SoMSHIELD == 1
    /* register LED driver */
    if (GOAL_RES_OK(res)) {
        res = goal_drvLedIicSynergy7SoMshieldReg(
            LEDS_OFF, GOAL_ID_MA_LED_DEFAULT);
    }

    if (GOAL_RES_OK(res)) {
        res = goal_maLedGetById(&pMaLed, GOAL_ID_MA_LED_DEFAULT);
    }

    if (GOAL_RES_OK(res)) {
        goal_maLedRegisterLed(pMaLed, GOAL_MA_LED_LED1_RED, 0);
        goal_maLedRegisterLed(pMaLed, GOAL_MA_LED_LED1_GREEN, 2);
```

```
            goal_maLedRegisterLed(pMaLed, GOAL_MA_LED_LED2_RED, 4);
            goal_maLedRegisterLed(pMaLed, GOAL_MA_LED_LED2_GREEN, 6);
            goal_maLedRegisterLed(pMaLed, GOAL_MA_LED_ETHERCAT, 8);
            goal_maLedRegisterLed(pMaLed, GOAL_MA_LED_PROFINET, 10);
            goal_maLedRegisterLed(pMaLed, GOAL_MA_LED_MODBUS, 12);
            goal_maLedRegisterLed(pMaLed, GOAL_MA_LED_ETHERNETIP, 14);
            goal_maLedRegisterLed(pMaLed, GOAL_MA_LED_CANOPEN, 16);
        }
    #endif
```

For a customer hardware a board specific driver should be implemented and registered accordingly.

#### *9.1.3.3 Timer*

The GOAL middleware requires a timer, which periodically calls the function goal_targetTimerCallback each millisecond. This timer is also initiated within the e2studio project.

#### *9.1.3.4 ThreadX*

The e2studio projects for the SoM requires HWOS features of ThreadX. The reach a specific resolution the ThreadX property “Timer Ticks Per Second” was increased to 1000.

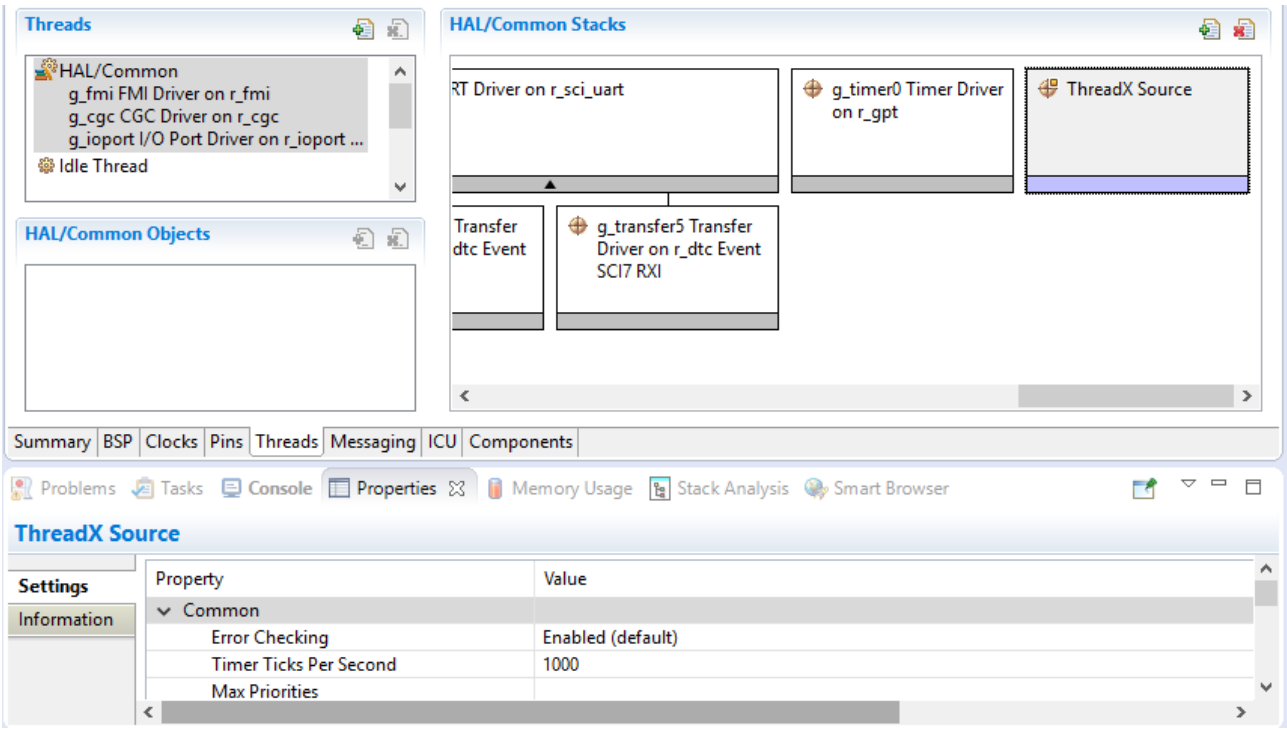


Figure 6 ThreadX configuration

### 9.1.4 Logging

There are multiple possibilities for logging. Once the SPI, thus RPC communication is established,

log messages of the AC can be transferred to the CC. Therefore, these messages can be seen using the Management tool. This possibility is enabled using an RPC function (see chapter 6.1.7).

Beside that local logging is possible using a UART. For both supported SYNERGY based development boards a UART is provided on the shield module. Please refer to the shield module documentation for details.

The UART is initialized with a baudrate of 115200 bps, 8 databits, no paritiy, 1 stop bit and no handshake. The connection provides both UART signals with 3.3 Volt level.

## 9.2 STM32

### 9.2.1 Development Environment

- STM Cube IDE 1.0

### 9.2.2 Supported Hardware

#### *9.2.2.1 Nucleo-STM32F429ZI*

https://www.st.com/en/evaluation-tools/nucleo-f429zi.html

- SPI Channel: 1

| Function | Pin |
|---|---|
| SPI MISO | GPIO A / PIN 6 |
| SPI MOSI | GPIO A / PIN 7 |
| SPI SCK | GPIO A / PIN 5 |
| SPI CS | GPIO D / PIN 14 |
| RESET | GPIO F / PIN 13 |

Table 46 SPI Pinning used on Nucleo-STM32F429ZI Board

- UART Channel: 3

| Function | Pin |
|---|---|
| RXD | GPIO C / PIN 9 |
| TXD | GPIO C / PIN 8 |

Table 47 UART Pinning used on Nucleo-STM32F429ZI Board

### 9.2.3 Adaption to customer hardware

SPI:
See driver in plat/drv/spi/stm32f4xx

UART:
See driver in plat/drv/uart/stm32fxxx

LED:
See driver in plat/drv/led/iic/stm32f4xx_ccmshield.c|h

### 9.2.4 Logging

There are multiple possibilities for logging. Once the SPI, thus RPC communication is established, log messages of the AC can be transferred to the CC. Therefore, these messages can be seen using the Management tool. This possibility is enabled using an RPC function (see chapter 6.1.7).

Beside that local logging is possible using a UART. For the examples the provided USB-UART Converter on the debugging interface is supported and UART logging is enabled by default.

The UART is initialized with a baudrate of 115200 bps, 8 databits, no paritiy, 1 stop bit and no handshake.

## 9.3 Raspberry Pi

### 9.3.1 Development Environment

- Raspberry Pi with raspian version 9 (stretch)

### 9.3.2 Supported Hardware

- Tested on raspberry pi 1 and 3
- SPI Channel: 1

| Function | Pin |
|---|---|
| SPI MISO | GPIO 9 |
| SPI MOSI | GPIO 10 |
| SPI SCK | GPIO 11 |
| SPI CS | GPIO 8 |
| RESET | GPIO 13 |

Table SPI Pinning used on raspberry pi

### 9.3.3 Adaption to customer hardware

SPI:
See driver in plat/drv/spi/linux

LED:
See driver in plat/drv/led/iic/iic_linux

### 9.3.4 Logging

There are multiple possibilities for logging. Once the SPI, thus RPC communication is established, log messages of the AC can be transferred to the CC. Therefore, these messages can be seen using the Management tool. This possibility is enabled using a RPC function (see chapter 6.1.7).

Beside that logging is possible to the console. Using the RPC function ***appl_ccmLogToAcEnable()*** also log messages from the CC module can be seen here.