|  |  |
| --- | --- |
| **User Documentation:**  **Framemanipulator** | |
|  | |
| Zum Ändern von Titel und Thema klicken Sie im Menü Datei auf Eigenschaften - keinesfalls direkt eingeben. | |
|  |  |
| Date: | January 23, 2014 |
| Documentation: | V 0.2.0 |

1. Versions

|  | Version | Date | Comment | Edited by |
| --- | --- | --- | --- | --- |
|  | 0.0.1 | August 31, 2012 | First Edition | Sebastian Mülhausen |
|  | 0.1.0 | January 21, 2014 | Clean-up documentation for the official FM V0.1.0 | Sebastian Mülhausen |
|  | 0.1.1 | January 21, 2014 | Update for FM V0.1.1 | Sebastian Mülhausen |
|  | 0.2.0 | January 23, 2014 | Update for FM V0.2.0 | Sebastian Mülhausen |

Table 1: Versions

1. Safety Notices

Safety notices in this document are organized as follows:

|  | Safety notice | Description |
| --- | --- | --- |
|  | Danger! | Disregarding the safety regulations and guidelines can be life-threatening. |
|  | Warning! | Disregarding the safety regulations and guidelines can result in severe injury or heavy damage to material. |
|  | Caution! | Disregarding the safety regulations and guidelines can result in injury or damage to material. |
|  | Information: | Important information used to prevent errors. |

Table 2: Safety notices

1. Purpose of this document

This document describes the functions and the usage of the POWERLINK Framemanipulator. Its target group are the users of this device. At least a basic knowledge of the Ethernet-protocol POWERLINK and the Controlled Nodes is of advantage for reading this document.

1. Table of Contents

1 Introduction 5

1.1 General Features 5

1.2 Framemanipulator Detection 5

1.3 The Different Hardware Platforms 6

1.3.1 BeMicro RTE Stick: 6

1.3.2 INK-Board: 6

1.4 System Overview 7

1.5 Frame Structures 8

2 Features 9

2.1 Frame Manipulations 9

2.2 OpenSAFETY Packets Manipulation 10

2.3 Device Features 12

2.3.1 Framemanipulator Configuration 12

2.3.1.1 Control the FM by using MN-PRes Payload: 12

2.3.1.2 Configuration using SDO Transfer: 12

2.3.2 Ethernet Traffic 12

2.3.3 Framemanipulator Features 12

3 Conditions for OpenSAFETY Manipulations 13

4 Automation Studio Integration 14

5 Control Interface 16

5.1 Operations 16

5.2 Status and Error Flags 17

5.3 Status LEDs 18

5.4 Control Examples 19

6 Manipulation Tasks 20

6.1 Task Settings 21

6.1.1 Dropping Frames 23

6.1.2 Delaying Frames 23

6.1.3 Manipulating the Headers 25

6.1.4 CRC Distortion 26

6.1.5 Truncate the Frame 26

6.1.6 Repetition of the Safety Packets 27

6.1.7 Loss of Safety Packets 27

6.1.8 Insertion of Safety Packets 28

6.1.9 Output of Safety Packets in Incorrect Sequence 28

6.1.10 Output of Incorrect Safety Packet-Data 29

6.1.11 Delay of Safety Packets 30

6.1.12 Masquerade Safety Packets 30

6.2 Frame Selection 31

6.3 Examples for Different Tasks 32

6.3.1 Drop SoC of the First POWERLINK Cycle 33

6.3.2 Cut PRes of NodeID 5 in POWERLINK Cycle Nr. 3 33

6.3.3 Manipulate SVID of ASnds 34

6.3.4 OpenSAFETY Packet Repetition 34

6.3.5 OpenSAFETY Incorrect Sequence 35

6.3.6 OpenSAFETY Packet Insertion 36

7 Appendix 37

7.1 References 37

7.2 Figure Index 38

7.3 Table Index 39

# Introduction

## General Features

The Framemanipulator is a runtime configurable device for the POWERLINK openConformance test and extends it with accurate timing features and frame manipulation possibilities. Furthermore the Ethernet-Framemanipulator is implemented as a scalable (generic IP-cores with SW) collaboration of different modules. Easy reuse for other purposes and extendibility was considered during the design phase.

The Framemanipulator (FM) will be put in series in between the POWERLINK MN and the DUT (Device under Test), thus it requires at least two Ethernet ports.

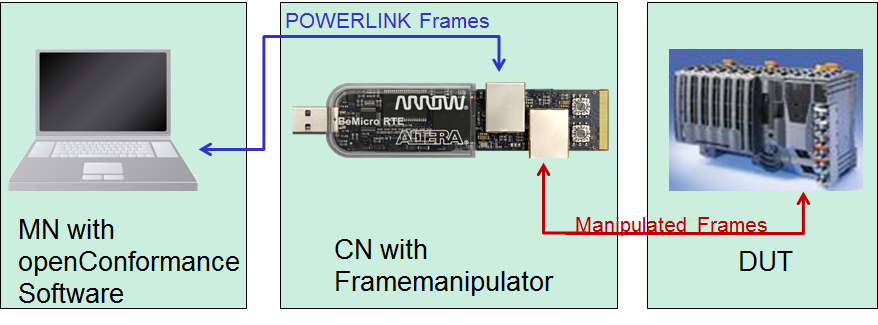


Figure 1: Test arrangement with the Framemanipulator

## Framemanipulator Detection

The present Framemanipulator can be detected und differenced by numerous objects. The expected values of the FM version 0.2.0 are shown in Table 3.

Table 3: Objects for detection of the Framemanipulator

|  |  |  |  |
| --- | --- | --- | --- |
| Name: | Object: | Data Type: | Expected value: |
| Vendor-ID | 0x1018/1 | U32 | 0x0100006C |
| Product Code | 0x1018/2 | U32 | 0x00000064 |
| FM-Version | 0x3005 | String | 0.2.0 |

Older versions of the Framemanipulator were created without the FM-Version object. An error will occur, when object 0x3005 is read.

## The Different Hardware Platforms

The Framemanipulator was developed for the implementation of the following platforms. These designs can also be used as a reference for the porting to other Altera Evaluation Boards.

### BeMicro RTE Stick:



Figure 2: BeMicro RTE-Stick [Arroweurope]

The BeMicro RTE stick is a good candidate for the implementation. It has RMIIs, enough Logic Elements (LE) for the hardware with its EP4CE22 of the Cyclone IV FPGA Family and 66 M9Ks (9x1024=>8192Bit) for internal memory. It is small and handy and gets its power from the USB port. It is also well suited for the finished product and is a very practical application platform in combination with the openConformance test.

Thanks to the RMIIs in slave mode, the FIFOs for synchronization of the data to the FPGA system clock are already included in the PHYs. Therefore they don’t have to be added separately to the FPGA hardware.

### INK-Board:

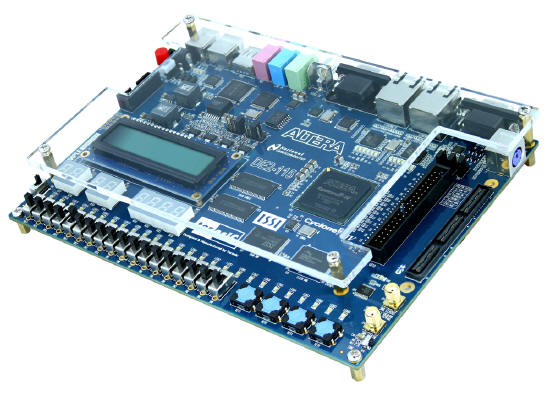


Figure 3: DE2-115 Development and education board [Altera]

For development and larger series of internal tests the DE2-115 Development Board, also called INK-Board (Industrial Networking Kit), was used. It has an EP4CE115 with 432 M9Ks. With this amount of memory it is possible to record multiple POWERLINK cycles with SignalTap.

## System Overview

The FPGA hardware consists of a normal POWERLINK slave with the actual Manipulator as an extension. The raw structure is depicted below:

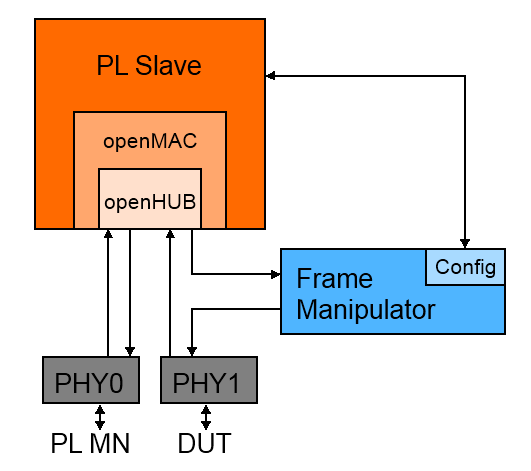


Figure 4: POWERLINK-Slave with Framemanipulator

POWERLINK frames of PHY0/Master and PHY1/DUT are transmitted by the Hub. The Framemanipulator IP-core possesses two data interfaces suitable for RMII PHYs. One is for the incoming frames and connected to the HUB output. The other interface is the output and is connected towards the PHY of the device under test. Therefore, all manipulations are **only** perceived by the DUT.

The Framemanipulator possesses an internal filter in hardware to eliminate unwanted frames. Only POWERLINK, ARP- and IP- (needed for e.g. SDO over UDP) frames are transferred to the DUT.

The PL Slave processes following tasks:

* Configuration of the PHYs with its openMAC
* Supply of the frame transfer via openHUB
* Transfer the configurations from the openConformance software to the Framemanipulator

## Frame Structures

The Ethernet frame consist of the Preamble (toggling Bits for synchronization), the Ethernet header, data and the Frame Check Sequence (FCS or CRC) for error detection.

A header consists of the Mac Addresses and the information about the following payload. This data can carry another header with data of a higher layer. This nesting of another protocol is called protocol stack. You can find e.g. HTTP data with a TCP header, the TCP Frame has an IP header and the IP Frame has an Ethernet header.

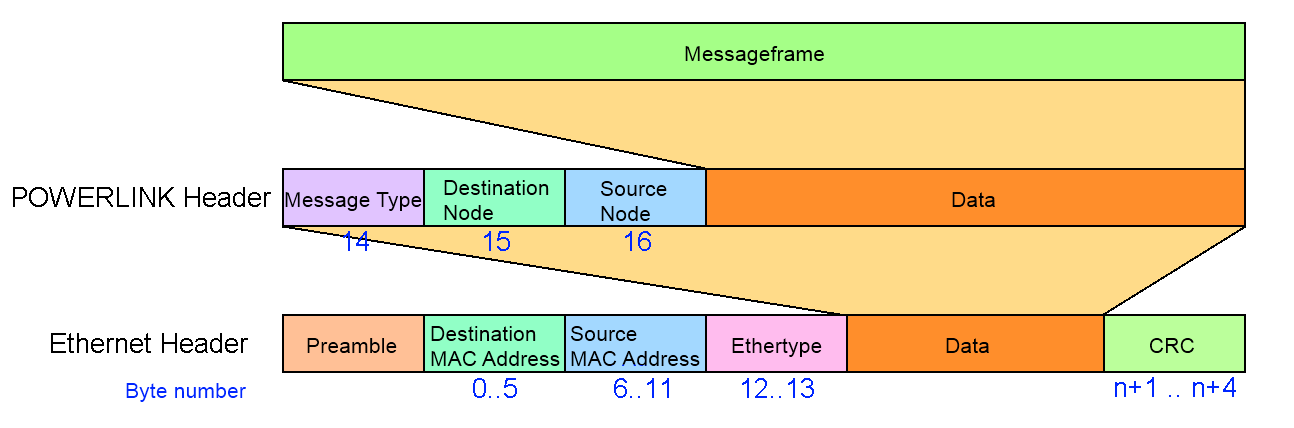


Figure 5: Ethernet and POWERLINK header

An Ethertype of the value 0x88AB is the token for a nested POWERLINK frame with POWERLINK-header. It consists of the number of the Destination and Source Node, the MessageType and Messagedata. The structure of the different messages is depicted below.

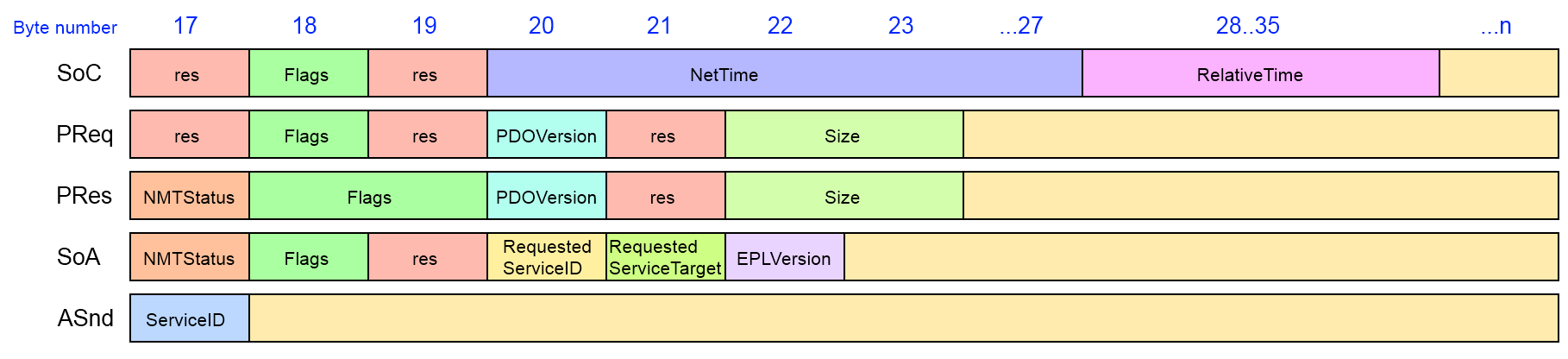


Figure 6: Structure of the POWERLINK frames

The structure varies from the different types of the POWERLINK frames. Special messages like SDO frames of the initialization and settings can only be detected with the embedded information like ServiceID or flags.

# Features

## Frame Manipulations

The different tasks of the manipulator are depicted below. The different parts of the frames are coloured like this:

* Preamble = Green
* Header data = Red
* Payload = Blue
* CRC = Yellow

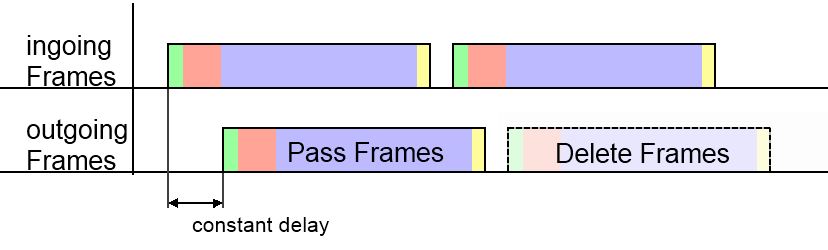


Figure 7: Tasks pass and delete frames

* **Pass** unselected frames without any disturbances
  + Keep a constant delay of all outgoing frames (like in *Figure 7*)
* **Remove** selected frames

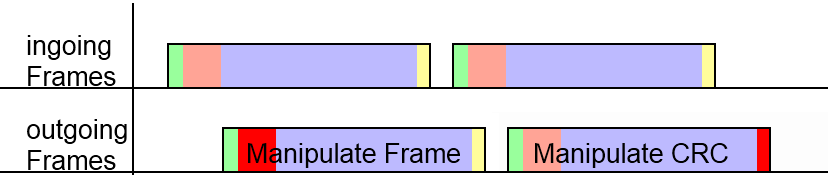


Figure 8: Tasks manipulate frame header and CRC

* **Manipulate** the values of the frame headers with a valid CRC (like in *Figure 8*)
* **Distort** the CRC

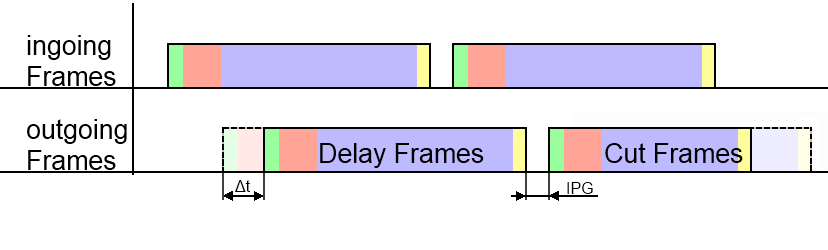


Figure 9: Tasks delay and truncate frames

* **Delay** frames of a certain amount of time (like in *Figure 9*)
  + with keeping the IPG (Inter Package Gap) to the following frame
* **Truncate** frames with a valid CRC

## OpenSAFETY Packets Manipulation

There are special frame manipulations to modify the DUT-packets of isochronous openSAFETY frames (packet A0 of Figure 10). To use these tasks, some conditions have to take into account (Chapter 3).

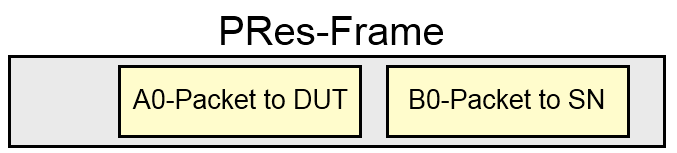


Figure 10: Safety frame with two safety packets

The manipulations are:

* **Repetition** of a safety packet with buffering the exchanged ones (Figure 11)

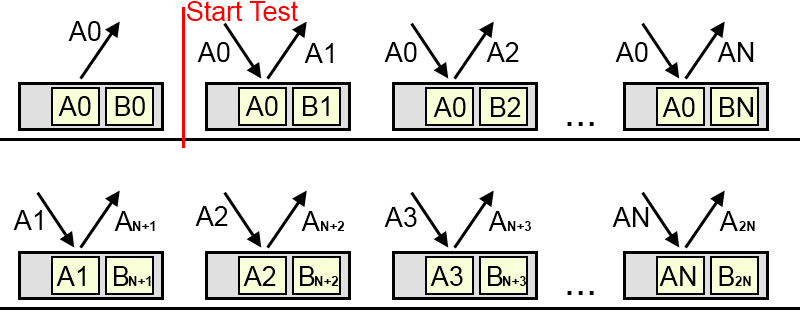


Figure 11: Task Repetition of safety packets

* **Loss** of the safety Packet (Figure 12)

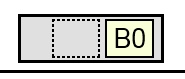


Figure 12: Task Packet-Loss

* **Insertion** of a safety packet, which was originally destined for another station (Figure 13)

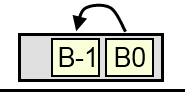


Figure 13: Task Insertion

* Release safety packets from the buffer in the **incorrect sequence** (Figure 14)

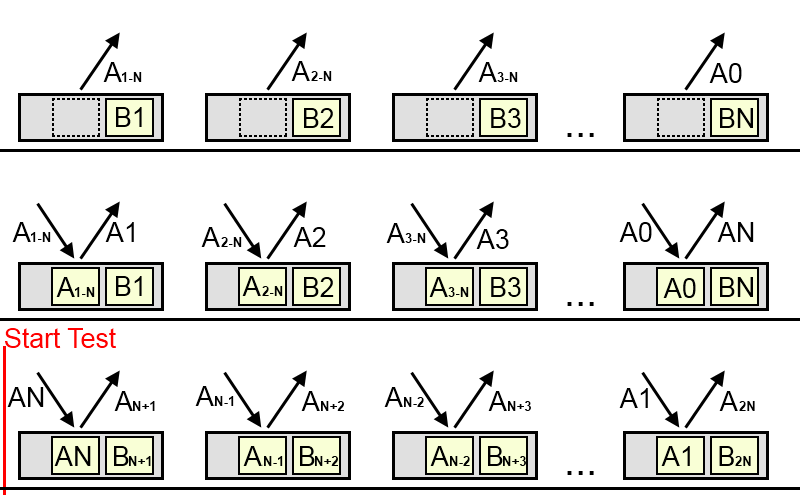


Figure 14: Task Incorrect-Sequence

* Distort the payload of the safety packet to create **incorrect data** and a false safety CRC (Figure 15)

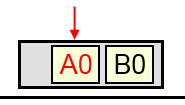


Figure 15: Task Incorrect-Data

* Buffer safety packets to create an **delay** (Figure 16)

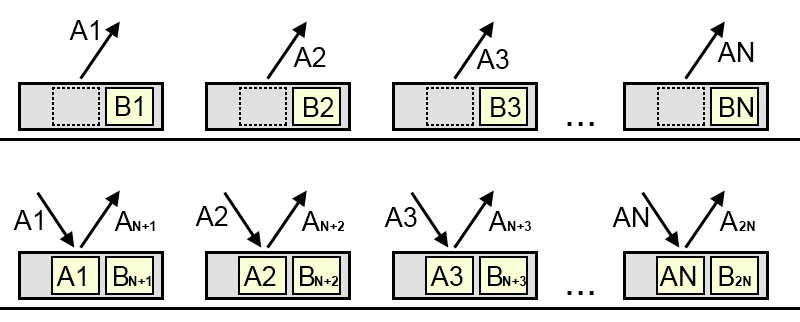


Figure 16: Task Packet-Delay

* **Masquerade** the safety frame by exchanging it with random data (Figure 17)

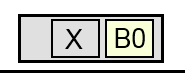


Figure 17: Task Masquerade

## Device Features

### Framemanipulator Configuration

#### Control the FM by using MN-PRes Payload:

This is an approach for a first test with the drawback of lacking flexibility.

* Starting of the test is transmitted by MN-PReq.
* Status indications of the FM takes place by using its PRes-Payload.

#### Configuration using SDO Transfer:

It is very flexible and requires a POWERLINK CN implementation which forwards the tasks to the actual Framemanipulator module.

* Configuration of the test is transmitted via SDO-Write.

### Ethernet Traffic

The Framemanipulator device introduces

* as less delay
* as less jitter

as possible for an forwarded or modified Ethernet frame. Generally speaking, the frame passes as less layers as possible and software based decisions on forwarding a frame was avoided.

### Framemanipulator Features

According to the features above, the FM has the following tasks:

* Ethernet frame filters which react on a matching pattern
* Manipulation of the Ethernet frame header and CRC is possible
* Frame counter (for sequence determination)
* Control and Task Interface
* Objects for the specific identification of a FM-device
* Timer functionality for frame delay
* MAC-CRC bypass capability (precalculated CRC) or manipulation of the CRC
* POWERLINK CN implementation which is able to access the control and task interface
* Two Ethernet ports – also with POWERLINK CN implementation

# Conditions for OpenSAFETY Manipulations

The Framemanipulator is able to run special tasks for the manipulation of the openSAFETY packets, which are limited in their usage. They can be used multiple times successively in a series of test, but shouldn’t be mixed with another openSAFETY- or a delay-task. A wrong combination will create a task-error (Chapter 5.2). The openSAFETY tasks are:

* Repetition
* Packet-Loss
* Insertion
* Incorrect Sequence
* Incorrect Data
* Packet-Delay
* Masquerade

These manipulations are designed to modify the openSAFETY packets of the isochronous POWERLINK phase, which are sent via cross-communication in the PRes of a SafeLogic (SL) and its SafetyNode (SN). Therefore, the set-up for a SN as the device under test is extended by a SL and an identical SN (Figure 18, left) with the same configuration. The identical SN is needed to create a second safety packet of the same size (packet B0 in Figure 10), which can be exchanged in the *Insertion* manipulation.

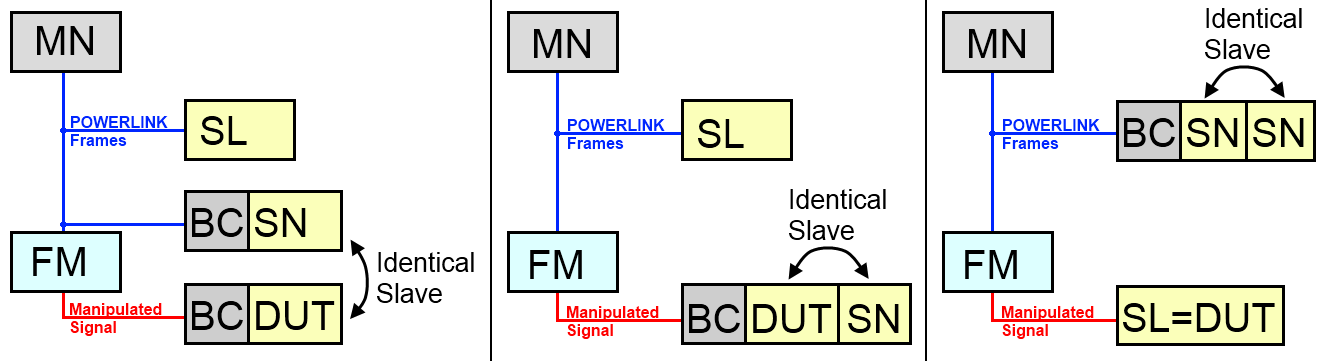


Figure 18: Test set-up for openSAFETY-Manipulations: Testing of a SN (left and middle), SL on the right

# Automation Studio Integration

The Framemanipulator-Slave is connected for testing purposes to a POWERLINK Master. In this example, an X20CP1484-1-PLC from B&R is used with the Automation Studio V3.0.90.21 SP03 software:

**1.** Import of the .xdc-file, which is located in folder *\app\objdict*:

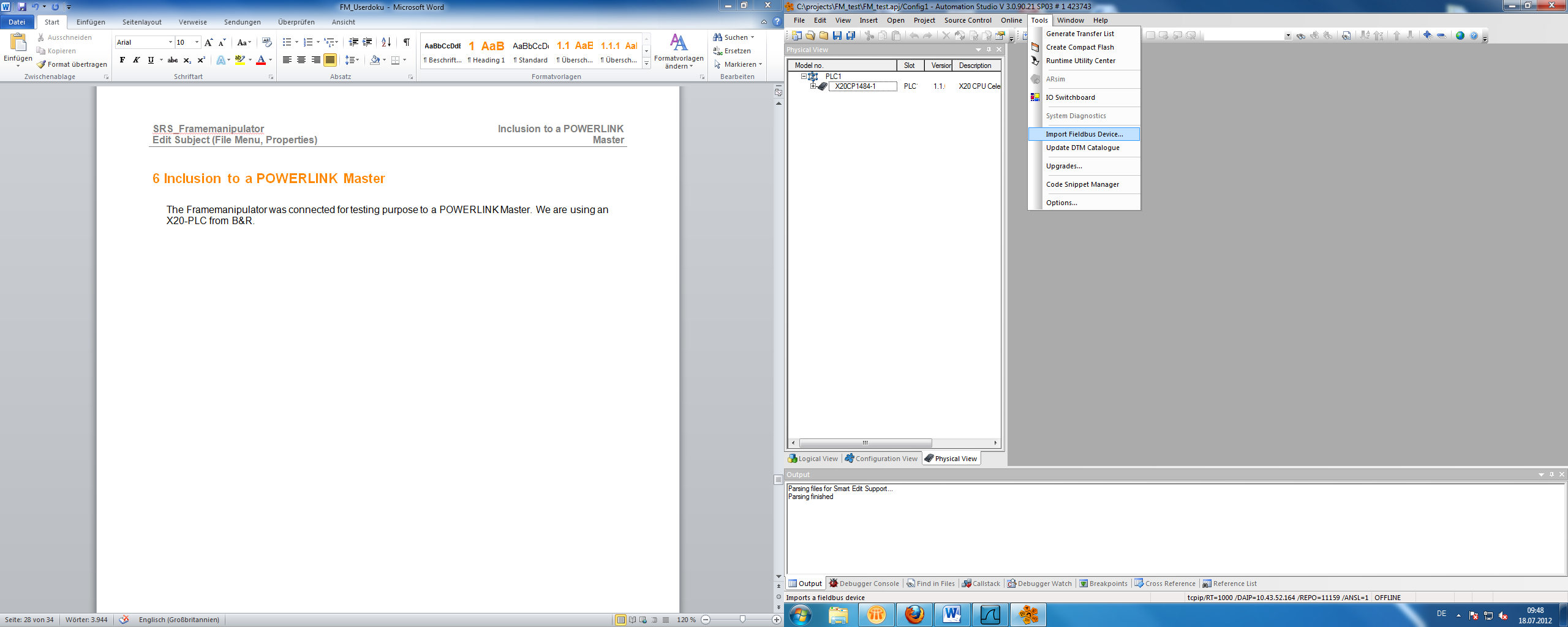


Figure 19: Automation Studio; Import Fieldbus-Device

**2.** Go to the X20-PLC and select “Open POWERLINK” via right click to insert our CN with the Framemanipulator with e.g. NodeID 2 (as well as our DUT):

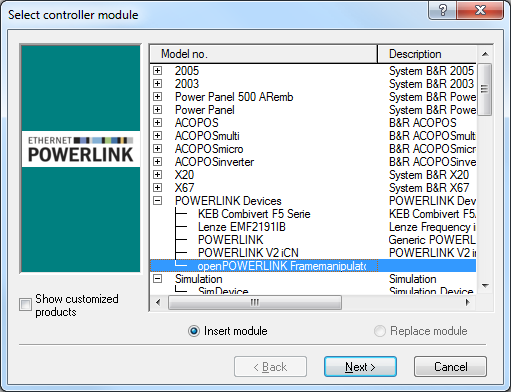


Figure 20: Selection of the CN with the Framemanipulator

**3.** The changes of the .xdd-file can be seen in the “*I/O Configuration*” of the Framemanipulator CN:

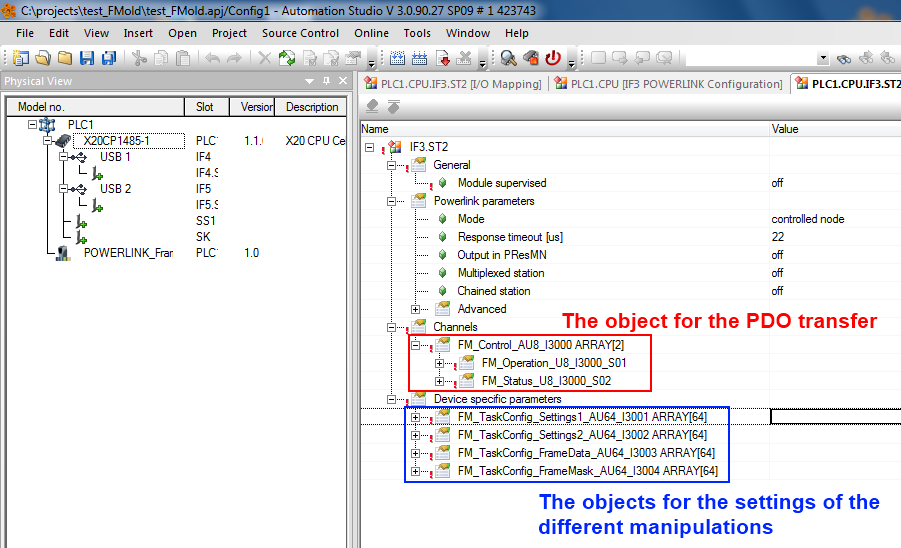


Figure 21: Automation Studio: I/O configuration of the Framemanipulator CN

The PDO-operation-object can be mapped in the “I/O Mapping” to another signal or variable.

The objects for the manipulation tasks can be used by filling it with an initial value or using the function “*EplSDOWrite*” of the “*AsEPL*” library. Please consider when using Automation-Studio that the byte-order of the configuration is in reverse (Figure 22).



Figure 22: Reverse order of bytes in the AS-configuration

# Control Interface

The Framemanipulator control is the interface to the openConformance software. It is connected to the object 0x3000 and transfers the data via PDO. The controlling is done via different operations of subindex 1. The resulting states and errors are collected in 0x3000/2.

There are also two status LEDs to display the execution of the series of test or the occurrence of an internal error.

## Operations

The different operations of the Framemanipulator are sent via PReq to object 0x3000/1. The following ones are currently available:

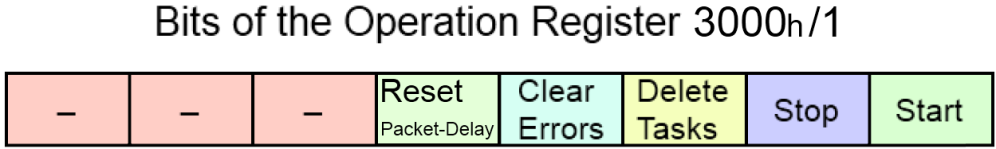


Figure 23: FM operation register

**Start manipulations (0x01):**

* Starts a new series of test (as long as the error register is empty)
* Restarts if the current test hasn’t finished
* The following frames are manipulated by the rules of the different tasks

**Stop manipulations (0x02):**

* Deactivates the current test

**Delete all tasks (0x04):**

* Sets all task objects to zero
* Deactivates the currently running test
* Needed after an *Incorrect-Sequence* task together with the packet-buffer operation 0x10 to prevent the creation of a new delay

**Clear all errors of the error register (0x08):**

* Set of all error bits (of chapter 5.2) to zero

**Reset the delay of packets created by openSAFETY tasks (0x10):**

* All stored packets of the tasks *Repetition*, *Incorrect-Sequence* and *Packet-Delay* will be deleted. This operation should be used after the series of test. The loss of these packets could affect the value of the error counter.

The different operation bits can also be sent in combination with each other. The first four operations (lower Nibble) are activated by the positive edge of this signal.

## Status and Error Flags

The status and errors flags of the Framemanipulator are stored in the following register. These are sent by the PRes to the MN via object 0x3000/2.

The following ones are currently available:

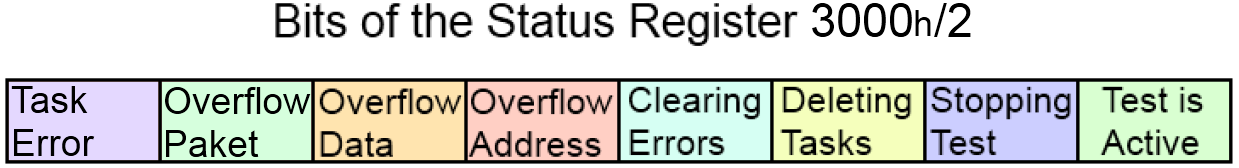


Figure 24: FM status register

**Operation Status (0x01-0x0F):**

* The tagged operation bit is currently processed

**Overflow of Address-Buffer (0x10):**

* Too many frames were stored during a delay task
* The default settings allow a storage of 64 different frames

**Overflow of Frame-Data-Buffer (0x20):**

* Too many frames or too big frames were stored during a delay task
* The default setting allows to store 2048 Bytes of data

*Solutions for Frame-Data- and Address-Buffer overflow***:**

* Reducing the number of stored frames
  + Reduction the delay-time
  + Changing the delay-type to e.g. 0x04 (*store only SoCs*)

**Overflow of Packet-Buffer (0x40):**

* Too many safety packets were stored during a task
* The default setting allows to store 1024 different packets (512 at *Incorrect-Sequece*) with a total of up to 16384 Bytes of data.
* This could happen, when the packet buffer wasn’t cleared with its operation (0x10) after the last series of test with safety tasks
* A reset of this Buffer via operation-flag 0x10 is recommended

**Error in the safety task configuration occurred (0x80):**

* Different safety tasks were used in a series of test, or…
* The configured tasks are overlapping (A safety task was started, while the last one hasn’t been finished yet), or…
* The two different safety packets are overlapping (wrong configuration of an *Insertion*-task), or…
* The packet sizes of multiple safety tasks differs in a series of test, or…
* The number of manipulated safety frames *N* of multiple *Incorrect-Sequence*-tasks differs in a series of test, or…
* A safety manipulation is used with a delay-task in a series of test

Occurred errors abort the current test of series and prevent the start of new ones. Their Error-Flags can be reseted by writing the *Clear all Error Flags* operation (0x08).

## Status LEDs

The Framemanipulator also has two status LEDs on the left side of the device (Figure 25). One of them is the active-LED, which is turned on during the series of test. The other one is the abort-LED. It will be activated, when the test was stopped by the user or an error occurred. It will only turn off, when all errors are cleared with the corresponding flag of the operation register.

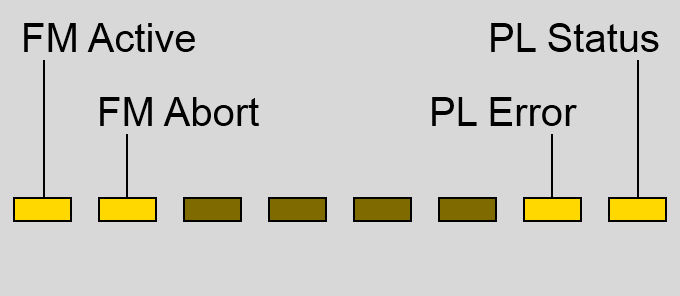


Figure 25: LEDs of the Framemanipulator device: FM- on the left, PL-status-LEDs on the right.

## Control Examples

The following examples are made for a better understanding of the behaviour of the Framemanipulator and the usage of the different operations.

*Example 1: Clear Error:*

An overflow of the frame-data-buffer occurred (PRes has a value of 0x20) as shown in *Figure 26.*

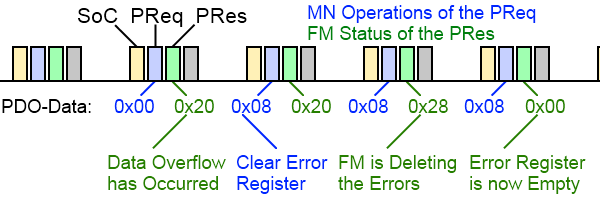


Figure 26: Clear error operation

We start a *Clear all Error Flags* operation with a 0x08 of the PReq Payload. This value stays for the following PReqs, but it doesn’t need to be constant.

The Framemanipulator receives the positive edge of the flag in the following POWERLINK-cycle. The processing of the operation is tagged with the set of the corresponding status bit (error & 0x08).

After the reset, the payload of the PRes is set to 0x00 again.

*Example 2: Behaviour of series of tests*:

A task with the drop of the SoC of the fifth cycle has to be started:

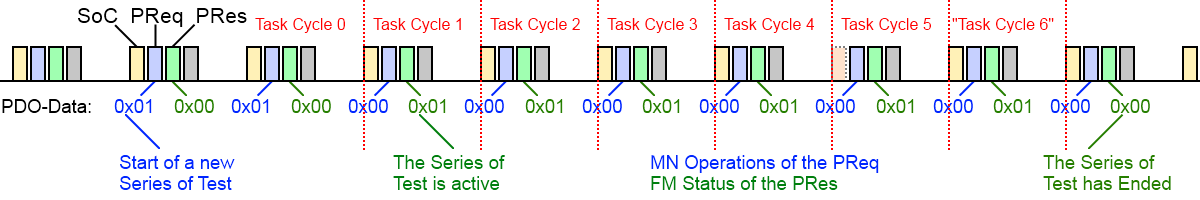


Figure 27: Start task operation with dropping the fifth SoC

After receiving the positive edge of the start flag, the Framemanipulator starts a series of test and responses with setting the corresponding bit. The transfer-time of the start-flag takes the FM of version 0.2.0 always two POWERLINK cycles. This time could differ when another version is used. For that reason it is recommended to use the status-register of the FM for checking the actual start of the series of test.

The dropping of the SoC is processed in the fifth cycle. Thus it is the only configured task, the series of test ends with reaching the sixth cycle and resets the status bit in the following one.

# Manipulation Tasks

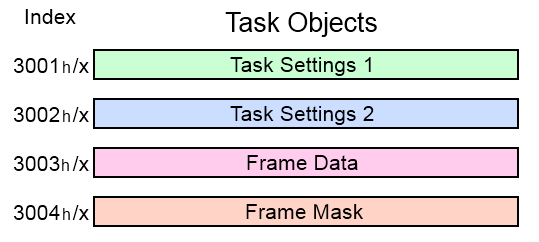


Figure 28: Objects of the manipulation tasks

The different manipulation tasks are configured by the subindices of the objects 0x3001 – 0x3004. Their data content is shown in *Figure 28*.

Subindex 0 contains the number of following subindices. These are written via SDO and contain eight byte of data each. The configuration of a specific task consists of 4 corresponding subindexes in the objects 0x3001-0x3004.

E.g. the first task has its settings in 0x3001/1 and 0x3002/1. The configuration of its frame-filter is contained in 0x3003/1 and its corresponding mask in 0x3004/1.

The sequence of different tasks is negligible (meaning: a task which should run in POWERLINK-Cycle 5 can be configured before one that runs in cycle 3). However, it should be avoided to create gaps between different tasks.

The Framemanipulator compares the received frame with all configured tasks to find a matching frame-filter, until it reaches the last entry or an empty task (gap). This behaviour was chosen in order to achieve a constant and deterministic time-delay for frame-processing.

For example: A configuration of ten tasks on subindices 1 to 11 without using the subindex number 8 will only process the tasks of subindex 1 to 7 (Figure 29).

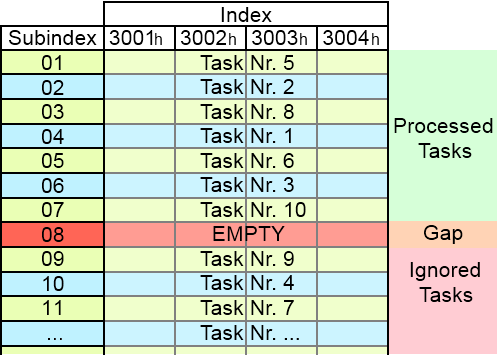


Figure 29: Example of the used task objects

Unused tasks can be deactivated by writing a cycle-number of 0 to the corresponding subobject of object 0x3001.

## Task Settings

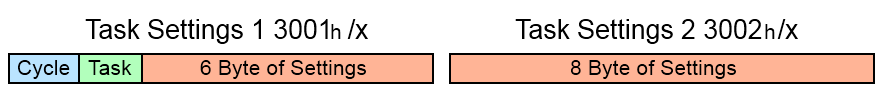


Figure 30: Basic structure of the task setting objects

The basic structure of the setting objects 0x3001 and 0x3002 is shown in *Figure 30*. The first byte of object 0x3001 contains the active cycle for this task.

**Byte 1: Cycle:**

* **0**: Task is inactive
  + Useful for small reconfigurations
  + Deactivated tasks are not interpreted as a gaps
* **1 – 254**: Task is active in the cycle of this number
  + The Framemanipulator starts counting incoming SoCs after receiving the start signal
* **255**: Task is active in all cycles until test is finished
  + Useful for manipulating infrequent events, whose cycle number is unknown.

The second byte contains the task-type:

**Byte 2: Task:**

* **0x01**: Drop frames
* **0x02**: Delay frames
* **0x04**: Manipulate the frame-header fields
* **0x08**: Distort CRC
* **0x10**: Truncate frames
* **0x81**: Repetition
* **0x82**: Packet-Loss
* **0x83**: Insertion
* **0x84**: Incorrect Sequence
* **0x85**: Incorrect Data
* **0x86**: Packet-Delay
* **0x87**: Masquerade

The meaning of the remaining 14 bytes depends on the selected task-type. The different possibilities are shown below and explained in detail in the following chapters:

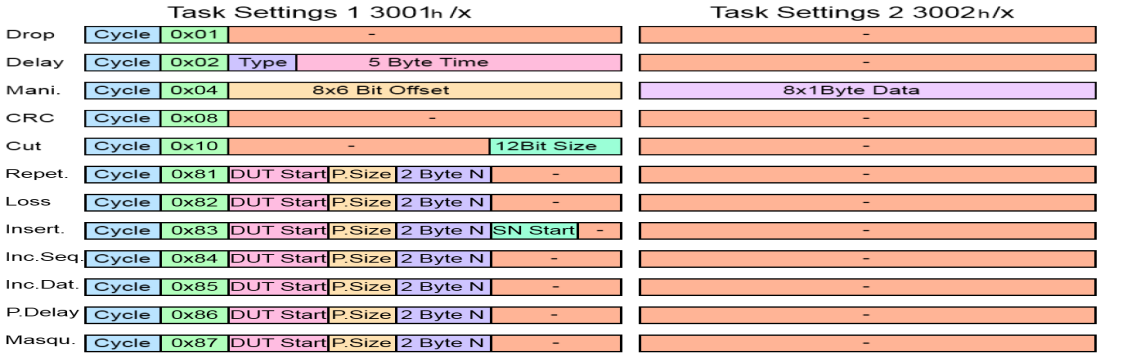


Figure 31: Whole structure of the task setting objects

### Dropping Frames

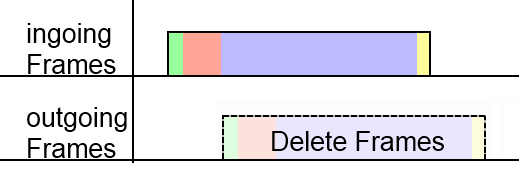


Figure 32: Task: Dropping frames

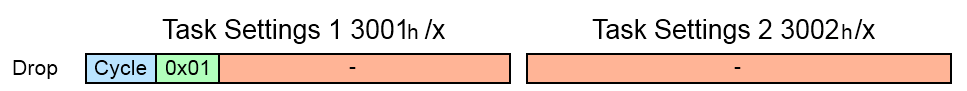


Figure 33: Task setting of dropping frames

The task for dropping selected frames doesn’t require further settings. Deleted frames can be used for error detection, like “Loss of SoC”

### Delaying Frames

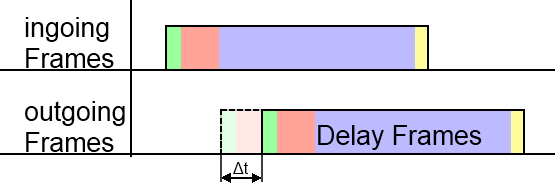


Figure 34: Task: Delaying frames

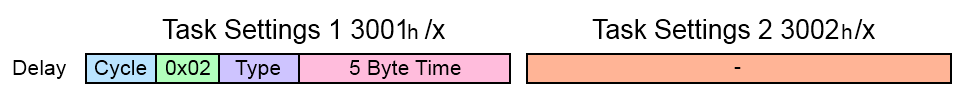


Figure 35: Task setting of delaying frames

The setting for delaying frames also contains a **delay-type** Byte, as well as the desired **delay-time** in 10ns steps:

**Byte 4 to 8: delay-time:**

* Adds an additional delay of .
  + Delays up to ns => ≈ 3 hours
* The delay of the Framemanipulator design (about 2,8µs) is negligible for the purpose of delaying frames

The Framemanipulator is able to manipulate a frame for a multiple amount of POWERLINK cycles. The incoming frames during this time can be handled in different ways.

The third Byte for the **delay-type** contains following information:

**Third Byte: delay-type**

* **0x01**: stores all incoming frames
  + All frames are stored while frame 2 of *Figure 36* is delayed:

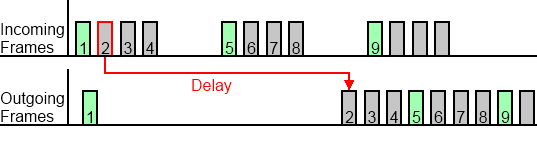
****

Figure 36: Delaying frame 2; Storing all other frames until frame 2 is passed on

* **0x02**: delete all incoming frames
  + All incoming frames are dropped during the delay:

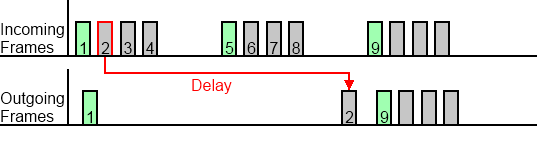


Figure 37: Delaying frame 2; Dropping of all other frames during the delay

* **0x04**: stores only SoCs
  + Only SoCs are stored. Other frame-types are dropped:

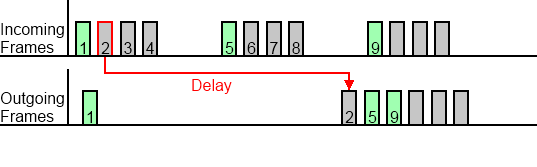


Figure 38: Delaying frame 2; Storing of the green coloured SoC-frames like frame 5

The number of storable frames depends on the variable *gBytesOfTheFrameBuffer*, which is adjustable in the component of the SOPC-Builder design. The default value generates a memory for up to 34 different frames.

**Attention:**

Don’t forget, that also manipulated and delayed frames can be dropped by the wrong *delay-type* and won’t be processed with its tasks.

### Manipulating the Headers

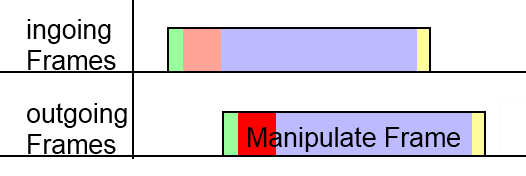


Figure 39: Task: Manipulating the headers

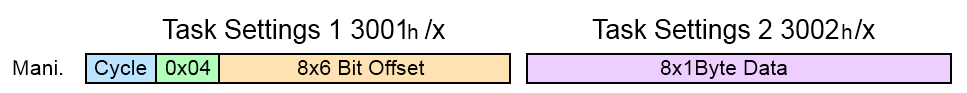


Figure 40: Task setting of manipulating the headers

The settings of the header manipulation include offsets and data to replace 8 different Bytes:

**Bytes 3 to 8: Offsets:**

* Contains 8 different offsets for the 8 Bytes of replaced data
* Each offset has a size of 6 Bit
  + The first 64 Bytes can be manipulated
* Offsets with the value of 0 are unused

**All 8 Byte of the second settings object: Data:**

* Contains 8 Byte to replace 8 header fields
* Fist Byte of data is connected to the first 6 Bit offset, and so on…

The manipulation can for example be used to create wrong Message- or Ethertypes.

### CRC Distortion

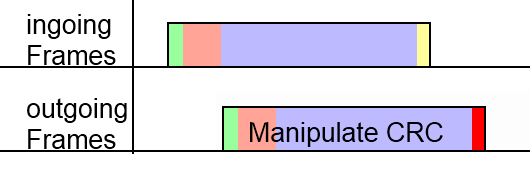


Figure 41: Task: CRC distortion

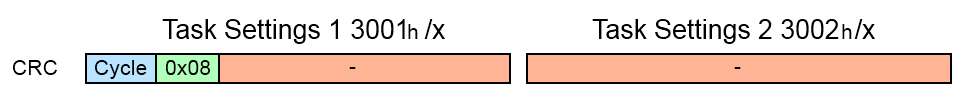


Figure 42: Task setting of CRC distortion

Frames can receive a false frame check sequence. Manipulated frames shouldn’t pass the MAC-layer of the DUT.

### Truncate the Frame

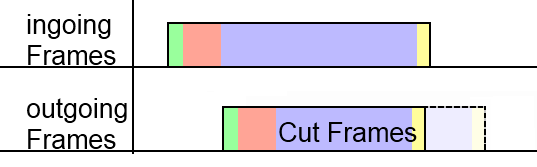


Figure 43: Task: Truncate the frame

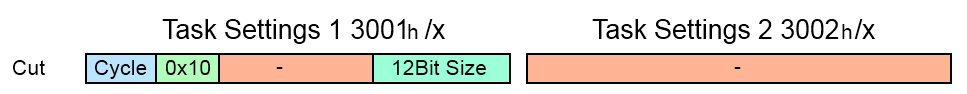


Figure 44: Task setting of frame truncation

The frame truncation setting contains the size of the outgoing frame.

**Last 12 Bits: Size of frame**

* The outgoing frame is truncated to size of this variable plus four Bytes of a valid CRC
  + A manipulation with size=60 creates a frame with Preamble plus a 64Bytes frame
    - This is the shortest frame, which is able to pass the MAC-layer
* Manipulations with a size value, which is bigger than the frame itself, leave the frame untouched

It is possible to test the MAC-layer of the DUT by creating frames with less than 64 Bytes.

POWERLINK frames with a PDO payload of e.g. 80Bytes can also be cut to shorten the payload.

* Received payload is smaller than the expected size (Byte number 23 of the message-header)
* The error should be noticed by the DUT

### Repetition of the Safety Packets

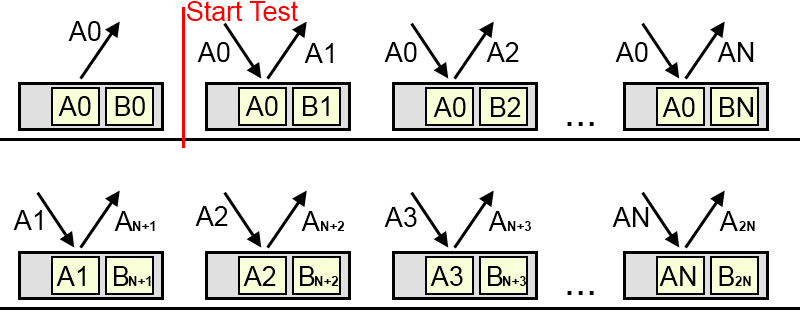


Figure 45: Task: Repetition of the safety packets

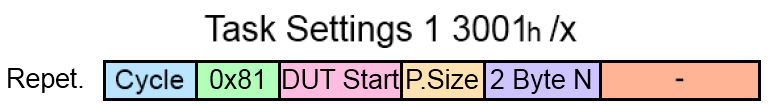
****

Figure 46: Task setting of packet repetition

The safety packet A0 (Figure 45) will be stored before the configured cycle starts and repeated in the following safety frames for N-times. The exchanged packets will be buffered and put out in the correct sequence. Every manipulation creates a delay of N-packets, which has to be removed after the series of test with the operation 0x10 (Chapter 5.1). Storing to many packets creates a packet-buffer overflow with the error-flag 0x40 (Chapter 5.2).

Additional parameters (Figure 46):

* **Byte 3: Offset of the manipulated DUT packet**
* **Byte 4: Size of the safety packet**
* **Bytes 5+6: Number of repeated packets N**

### Loss of Safety Packets

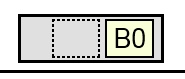


Figure 47: Task: Loss of safety packets

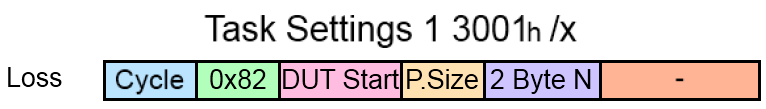


Figure 48: Task setting of packet loss

The safety packets will be removed (Figure 47).

Additional parameters (Figure 48):

* **Byte 3: Offset of the manipulated DUT packet**
* **Byte 4: Size of the safety packet**
* **Bytes 5+6: Number of deleted packets N**

### Insertion of Safety Packets

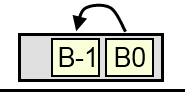


Figure 49: Task: Insertion of safety packets

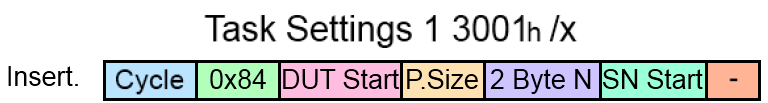


Figure 50: Task setting of Insertion

The DUT-packet will be exchanged with the last safety packet from another SN (Figure 49). Both packets should have the same size. The sequence of packets in the POWERLINK frame is not important.

The configuration will be checked for an overlapping of the two different safety packets. The error-flag 0x80 (Chapter 5.2) will be set.

Additional parameters (Figure 50):

* **Byte 3: Offset of the manipulated DUT packet**
* **Byte 4: Size of the safety packet**
* **Bytes 5+6: Number of exchanged packets N**
* **Byte 7: Offset of the second safety packet**

### Output of Safety Packets in Incorrect Sequence

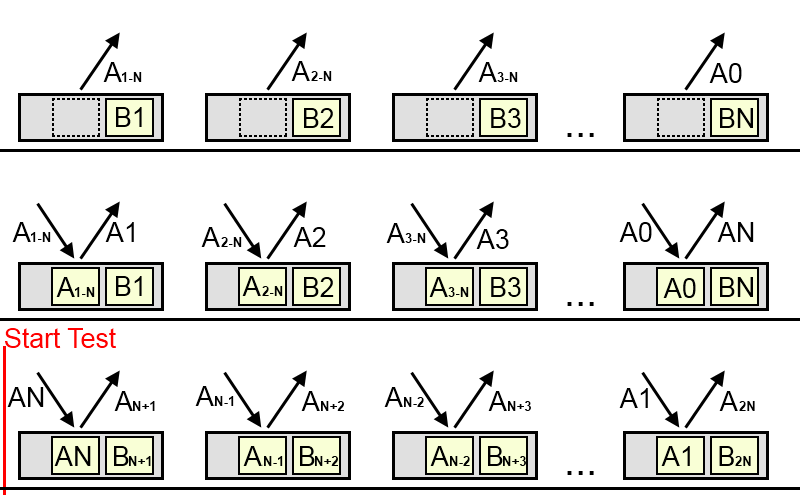


Figure 51: Task: Incorrect-Sequence of safety packets

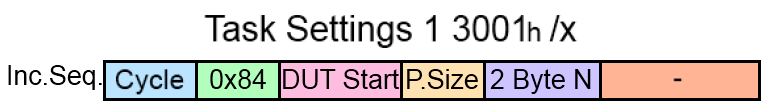
****

Figure 52: Task setting of Incorrect-Sequence

Multiple safety packets will be stored inside the packet buffer and released in reverse order. The storage starts with a *Packet-Delay* task (Chapter 6.1.11) once the configuration is done (first two rows in Figure 51). The change of the packet sequence starts with the cycle of each *Incorrect-Sequence* configuration.

This task requires twice as much entries in the address-memory of the packet-buffer. Therefore only 512 packets can be stored with the default setting of the Framemanipulator.

The *Packet-Delay* task creates a delay of N-packets, which has to be removed with the manipulation tasks itself after the series of test. The used operation is 0x14 (Chapter 5.1). A reset of the packet-buffer without clearing the tasks creates a new delay afterwards.

The delay is constant and doesn’t depend on the number of *Incorrect-Sequence* tasks. Storing to many packets creates a packet-buffer overflow with the error-flag 0x40 (Chapter 5.2).

Additional parameters (Figure 52):

* **Byte 3: Offset of the manipulated DUT packet**
* **Byte 4: Size of the safety packet**
* **Bytes 5+6: Number of exchanged packets N.**

**This parameter should stay the same when using multiple tasks in one series of test. Otherwise, the error-flag 0x80 (Chapter 5.2) will be set.**

### Output of Incorrect Safety Packet-Data

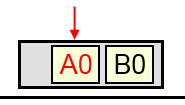


Figure 53: Task: Incorrect-Data manipulation of safety packets

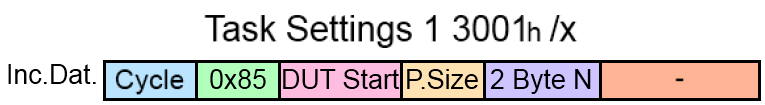


Figure 54: Task setting of Incorrect-Data

Distort the first Byte of the safety packet payload (Figure 53) to create a false safety CRC.

**Attention:**

According to the configuration of the DUT, the received safety packets can be sent without a payload. These packets will be distorted directly at their first safety CRC Byte.

Additional parameters (Figure 54):

* **Byte 3: Offset of the manipulated DUT packet**
* **Byte 4: Size of the safety packet**
* **Bytes 5+6: Number of manipulated packets N.**

### Delay of Safety Packets

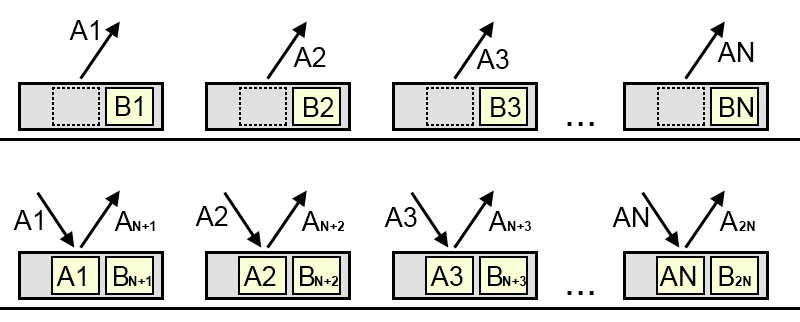


Figure 55: Task: Delay of safety packets

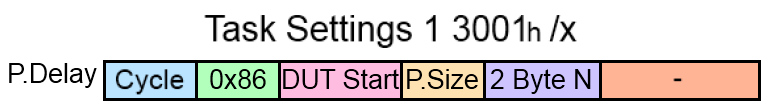
****

Figure 56: Task setting of Packet-Delay

The N selected packets will be deleted from the safety frame (Figure 55). The removed packets will be exchanged afterwards in the correct order. A delay of safety packets occur.

Every manipulation creates a delay of N-packets, which has to be removed after the series of test with the operation 0x10 (Chapter 5.1). Storing to many packets creates a packet-buffer overflow with the error-flag 0x40 (Chapter 5.2).

Additional parameters (Figure 56):

* **Byte 3: Offset of the manipulated DUT packet**
* **Byte 4: Size of the safety packet**
* **Bytes 5+6: Number of delayed packets N.**

### Masquerade Safety Packets

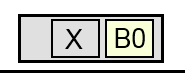


Figure 57: Task: Masquerade safety packets

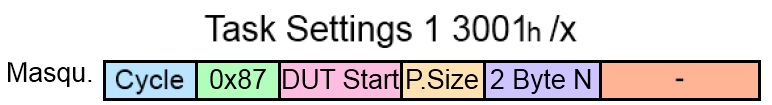


Figure 58: Task setting of Masquerade

The whole safety packets will be overwritten with random data (Figure 57). The packet will be exchanged with parts of the last SoC Timestamp.

Additional parameters (Figure 58):

* **Byte 3: Offset of the manipulated DUT packet**
* **Byte 4: Size of the safety packet**
* **Bytes 5+6: Number of manipulated packets N.**

## Frame Selection

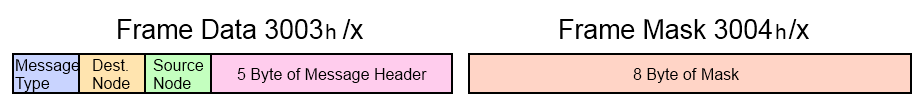


Figure 59: Basic structure of the responsible objects for filtering the frames

The basic structure of the frame data object is shown in *Figure 59*. The Framemanipulator possesses an internal filter in hardware for the Ethertype. It only allows POWERLINK- (0x88AB), ARP- (0x0806) and IP- (0x0800, needed for e.g. SDO over UDP) frames. Bytes 4 to 8 depend on the selected Message-Type. The different possibilities are shown below:

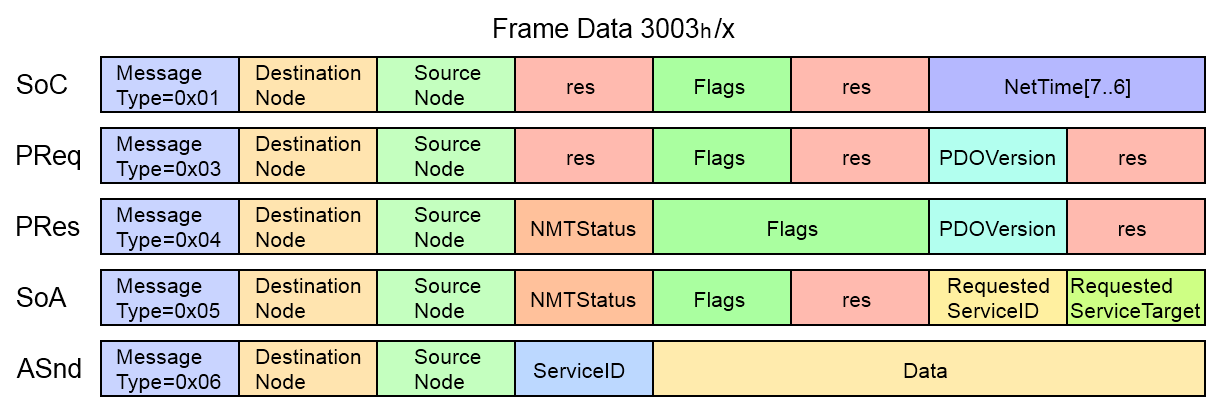


Figure 60: Filter object structure of the different messages

The mask of the object 0x3004 selects the Bits for the comparison.

This is done via the following formula:

Here are some examples:

The first 4 bit of the following 8 bit data should be compared in the same way.

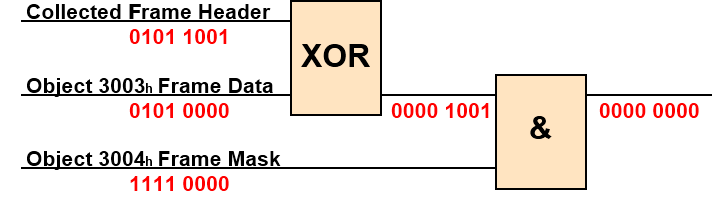


Figure 61: Filter example: Data conformance

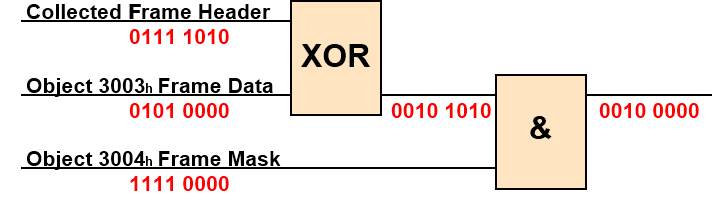


Figure 62: Filter example: Data conformance not fulfilled

Tasks with a conformant cycle number and fitting data and mask to the incoming frame are executed. A frame can have only one manipulation task at the same time. If more than one task is fitting to the frame, then the last one will be executed.

## Examples for Different Tasks

This chapter is used for some examples for a better understanding of the task configuration. The most important information are depicted below:

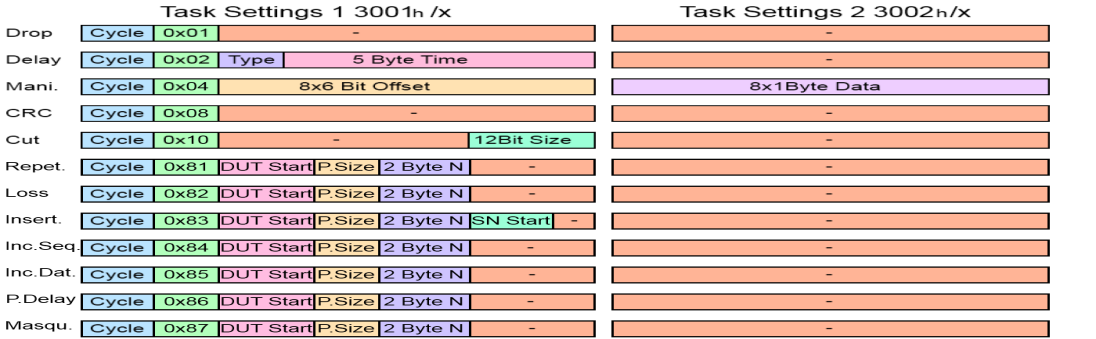


Figure 63: Task-setting objects structure

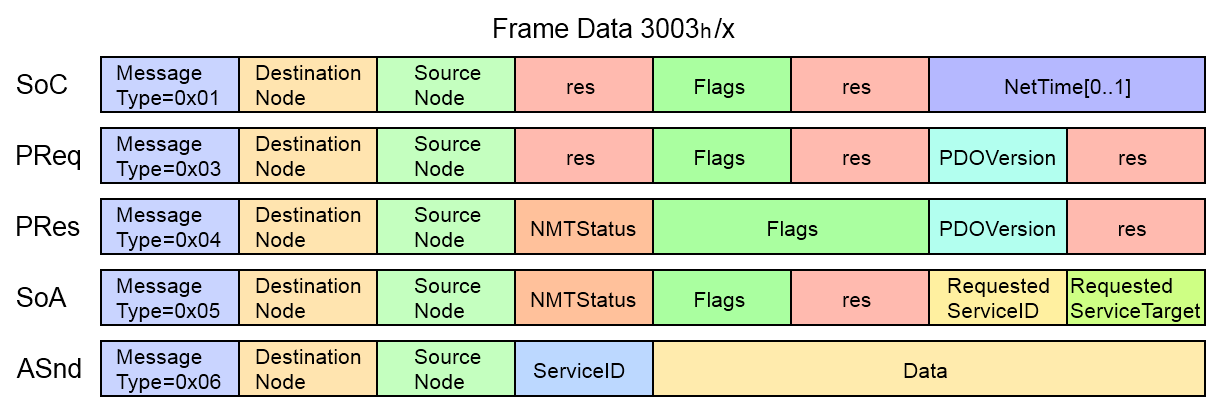


Figure 64: Frame-data object structure

### Drop SoC of the First POWERLINK Cycle

First, we start with a little task like dropping a SoC

* **0x3001/x Task Setting 1:**
  + Cycle 1 => 0x01 to the first Byte
  + Dropping => 0x01 to the second Byte
* **0x3002/x Task Setting 2:**
  + Dropping => empty
* **0x3003/x Frame Data:**
  + SoC => 0x01 to the first Byte
* **0x3004/x Frame Mask:**
  + Activate Message-Type => 0xFF to the first Byte

|  |  |
| --- | --- |
| Object | Data |
| 0x3001/x | 0x0101000000000000 |
| 0x3002/x | 0x0000000000000000 |
| 0x3003/x | 0x0100000000000000 |
| 0x3004/x | 0xFF00000000000000 |

### Cut PRes of NodeID 5 in POWERLINK Cycle Nr. 3

In this example we have another POWERLINK Slave in the network, which is communicating with our DUT via PDO over cross-communication. The PDO has e.g. a size of 70 Byte and should be truncated to the minimum frame size of 64 (including CRC => length = 60):

* **0x3001/x Task Setting 1:**
  + Cycle 3 => 0x03 to the first Byte
  + Cut => 0x10 to the second Byte
  + Length =>60 => 0x3C to the last Byte
* **0x3002/x Task Setting 2:**
  + Cut => empty
* **0x3003/x Frame Data:**
  + PRes => 0x04 to the first Byte
  + Source ID =>5 => 0x05 to the third Byte
* **0x3004/x Frame Mask:**
  + Activate Message-Type => 0xFF to the first Byte
  + Activate Source ID => 0xFF to the third Byte

|  |  |
| --- | --- |
| Object | Data |
| 0x3001/x | 0x031000000000003C |
| 0x3002/x | 0x0000000000000000 |
| 0x3003/x | 0x0400050000000000 |
| 0x3004/x | 0xFF00FF0000000000 |

### Manipulate SVID of ASnds

The ServiceID of the following ASnds should be manipulated from the value 5 to value 7. The exact cycle is unknown, but should be within the following POWERLINK cycles.

* **0x3001/x Task Setting 1:**
  + Cycle: all => 0xFF to the first Byte
  + Manipulate => 0x04 to the second Byte
  + Offset 1 =>18 => 010010 the first 6 Bit of the third Byte
* **0x3002/x Task Setting 2:**
  + New Data 1 =>7 => 0x07 to the first Byte
* **0x3003/x Frame Data:**
  + ASnd => 0x06 to the first Byte
  + ServiceID =>5 => 0x05 to the fourth Byte
* **0x3004/x Frame Mask:**
  + Activate Message-Type => 0xFF to the first Byte
  + Activate ServiceID => 0xFF to the fourth Byte

|  |  |
| --- | --- |
| Object | Data |
| 0x3001/x | 0xFF04480000000000 |
| 0x3002/x | 0x0700000000000000 |
| 0x3003/x | 0x0600000500000000 |
| 0x3004/x | 0xFF0000FF00000000 |

### OpenSAFETY Packet Repetition

The safety packets of the SafetLogic PRes (Node Nr. 5) should be repeated 400-times. The safety packet starts at Byte 20 and has a size of 26 Bytes.

* **0x3001/x Task Setting 1:**
  + Cycle 1 => 0x01 to the first Byte
  + Repetition => 0x81 to the second Byte
  + Packet start =>20 => 0x14 to the third Byte
  + Packet size =>26 => 0x1A to the fourth Byte
  + Number of packets =>400 => 0x0190 to Byte five and six
* **0x3002/x Task Setting 2:**
  + Repetition => empty
* **0x3003/x Frame Data:**
  + PRes => 0x04 to the first Byte
  + Source ID =>5 => 0x05 to the third Byte
* **0x3004/x Frame Mask:**
  + Activate Message-Type => 0xFF to the first Byte
  + Activate Source ID => 0xFF to the third Byte
* **0x3000/1 Reset of the Packet Memory:**
  + Clear the occurred delay of packets after the series of test with 0x10

|  |  |
| --- | --- |
| Object | Data |
| 0x3001/x | 0x0181141A01900000 |
| 0x3002/x | 0x0000000000000000 |
| 0x3003/x | 0x0400050000000000 |
| 0x3004/x | 0xFF00FF0000000000 |

### OpenSAFETY Incorrect Sequence

The 200 safety packets of the SafetLogic PRes (Node Nr. 5) should be sent in the incorrect sequence. The safety packet starts at Byte 20 and has a size of 26 Bytes.

* **0x3001/x Task Setting 1:**
  + Cycle 1 => 0x01 to the first Byte
  + Incorrect Sequence => 0x84 to the second Byte
  + Packet start =>20 => 0x14 to the third Byte
  + Packet size =>26 => 0x1A to the fourth Byte
  + Number of packets =>200 => 0x00C8 to Byte five and six
* **0x3002/x Task Setting 2:**
  + Incorrect Sequence => empty
* **0x3003/x Frame Data:**
  + PRes => 0x04 to the first Byte
  + Source ID =>5 => 0x05 to the third Byte
* **0x3004/x Frame Mask:**
  + Activate Message-Type => 0xFF to the first Byte
  + Activate Source ID => 0xFF to the third Byte
* **0x3000/1 Start of the Series of Test after some Delay:**
  + Once the Framemanipulator is configured, it will start with a *Packet-Delay* task to buffer the 200 needed safety packets (Chapter 6.1.9). The test can be started with 0x01 as soon as the error counter of the DUT has returned to its original state.
* **0x3000/1 Reset of the Packet Memory and the task itself:**
  + Clear the occurred delay of packets after the series of test with 0x14

|  |  |
| --- | --- |
| Object | Data |
| 0x3001/x | 0x0184141A00C80000 |
| 0x3002/x | 0x0000000000000000 |
| 0x3003/x | 0x0400050000000000 |
| 0x3004/x | 0xFF00FF0000000000 |

### OpenSAFETY Packet Insertion

The safety packets for the DUT should be exchanged with the packets from another SN. The DUT packet starts at Byte 20, the packet of the other SN at 46. Both have a size of 26 Bytes. The safety frame is a PRes from a SL (Node Nr. 5) and should be manipulated 100-times.

* **0x3001/x Task Setting 1:**
  + Cycle 1 => 0x01 to the first Byte
  + Insertion => 0x83 to the second Byte
  + DUT packet start =>20 => 0x14 to the third Byte
  + Packet size =>26 => 0x1A to the fourth Byte
  + Number of packets =>100 => 0x0064 to Byte five and six
  + SN packet start => 46 => 0x2E to the seventh Byte
* **0x3002/x Task Setting 2:**
  + Insertion => empty
* **0x3003/x Frame Data:**
  + PRes => 0x04 to the first Byte
  + Source ID =>5 => 0x05 to the third Byte
* **0x3004/x Frame Mask:**
  + Activate Message-Type => 0xFF to the first Byte
  + Activate Source ID => 0xFF to the third Byte

A clear of the packet memory isn’t needed in this task.

|  |  |
| --- | --- |
| Object | Data |
| 0x3001/x | 0x0183141A00642E00 |
| 0x3002/x | 0x0000000000000000 |
| 0x3003/x | 0x0400050000000000 |
| 0x3004/x | 0xFF00FF0000000000 |

# Appendix

## References

Altera (14.Mai.2012)

<http://www.altera.com>

Arroweurope (11.Mai.2012)

<http://www.arroweurope.com>

Ethernet-POWERLINK (20.April.2012)

<http://www.ethernet-powerlink.org>

## Figure Index

Figure 1: Test arrangement with the Framemanipulator 5

Figure 2: BeMicro RTE-Stick [Arroweurope] 6

Figure 3: DE2-115 Development and education board [Altera] 6

Figure 4: POWERLINK-Slave with Framemanipulator 7

Figure 5: Ethernet and POWERLINK header 8

Figure 6: Structure of the POWERLINK frames 8

Figure 7: Tasks pass and delete frames 9

Figure 8: Tasks manipulate frame header and CRC 9

Figure 9: Tasks delay and truncate frames 9

Figure 10: Safety frame with two safety packets 10

Figure 11: Task Repetition of safety packets 10

Figure 12: Task Packet-Loss 10

Figure 13: Task Insertion 10

Figure 14: Task Incorrect-Sequence 11

Figure 15: Task Incorrect-Data 11

Figure 16: Task Packet-Delay 11

Figure 17: Task Masquerade 11

Figure 18: Test set-up for openSAFETY-Manipulations: Testing of a SN on the left, SL on the right 13

Figure 19: Automation Studio; Import Fieldbus-Device 14

Figure 20: Selection of the CN with the Framemanipulator 14

Figure 21: Automation Studio: I/O configuration of the Framemanipulator CN 15

Figure 22: Reverse order of bytes in the AS-configuration 15

Figure 23: FM operation register 16

Figure 24: FM status register 17

Figure 25: LEDs of the Framemanipulator device: FM- on the left, PL-status-LEDs on the right. 18

Figure 26: Clear error operation 19

Figure 27: Start task operation with dropping the fifth SoC 19

Figure 28: Objects of the manipulation tasks 20

Figure 29: Example of the used task objects 20

Figure 30: Basic structure of the task setting objects 21

Figure 31: Whole structure of the task setting objects 22

Figure 32: Task: Dropping frames 23

Figure 33: Task setting of dropping frames 23

Figure 34: Task: Delaying frames 23

Figure 35: Task setting of delaying frames 23

Figure 36: Delaying frame 2; Storing all other frames until frame 2 is passed on 24

Figure 37: Delaying frame 2; Dropping of all other frames during the delay 24

Figure 38: Delaying frame 2; Storing of the green coloured SoC-frames like frame 5 24

Figure 39: Task: Manipulating the headers 25

Figure 40: Task setting of manipulating the headers 25

Figure 41: Task: CRC distortion 26

Figure 42: Task setting of CRC distortion 26

Figure 43: Task: Truncate the frame 26

Figure 44: Task setting of frame truncation 26

Figure 45: Task: Repetition of the safety packets 27

Figure 46: Task setting of packet repetition 27

Figure 47: Task: Loss of safety packets 27

Figure 48: Task setting of packet loss 27

Figure 49: Task: Insertion of safety packets 28

Figure 50: Task setting of Insertion 28

Figure 51: Task: Incorrect-Sequence of safety packets 28

Figure 52: Task setting of Incorrect-Sequence 28

Figure 53: Task: Incorrect-Data manipulation of safety packets 29

Figure 54: Task setting of Incorrect-Data 29

Figure 55: Task: Delay of safety packets 30

Figure 56: Task setting of Packet-Delay 30

Figure 57: Task: Masquerade safety packets 30

Figure 58: Task setting of Masquerade 30

Figure 59: Basic structure of the responsible objects for filtering the frames 31

Figure 60: Filter object structure of the different messages 31

Figure 61: Filter example: Data conformance 31

Figure 62: Filter example: Data conformance not fulfilled 31

Figure 63: Task-setting objects structure 32

Figure 64: Frame-data object structure 32

## Table Index

Table 1: Versions 2

Table 2: Safety notices 2

Table 3: Objects for detection of the Framemanipulator 5