|  |  |
| --- | --- |
| **User Documentation:**  **Framemanipulator** | |
|  | |
| Zum Ändern von Titel und Thema klicken Sie im Menü Datei auf Eigenschaften - keinesfalls direkt eingeben. | |
|  |  |
| Date: | January 21, 2014 |
| Documentation: | V 0.1.0 |

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 |
|  |  |  |  |  |
|  |  |  |  |  |

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 basic knowledge of the Ethernet-protocol POWERLINK and the Controlled Nodes is of advantage for reading this document.

1. Table of Contents

1 Introduction 4

1.1 General Features 4

1.2 Framemanipulator Detection 4

1.3 The different Hardware Platforms 5

1.3.1 BeMicro RTE Stick 5

1.3.2 INK-Board 5

1.4 System Overview 6

1.5 Frame Structures 7

2 Features 8

2.1 Features for openConformance Test 8

2.1.1 Dropping of specific frames in a specified sequence 8

2.1.2 Delay of specified frames for a specified amount of time (in a specified sequence) 8

2.1.3 Manipulation of POWERLINK-Header fields (in a specified sequence) 8

2.1.4 Manipulation of Ethernet-CRC (in a specified sequence) 8

2.1.5 Truncate payload of PReq-Frames 8

2.1.6 Example: Typical Test sequence of openConformance Test using FM e.g. “Loss of SoC”: 8

2.2 The Different Manipulation Tasks 9

2.3 Device Features 10

2.3.1 Framemanipulator Configuration 10

2.3.1.1 Control the FM by using MN-PRes Payload: 10

2.3.1.2 Configuration using SDO Transfer: 10

2.3.2 Ethernet Traffic 10

2.3.3 Framemanipulator Features 10

3 Automation Studio Integration 11

4 Control Interface 13

4.1 Operations 13

4.2 Status Flags 14

4.3 Control Examples 15

5 Manipulation Tasks 16

5.1 Task Settings 17

5.1.1 Dropping Frames 18

5.1.2 Delaying Frames 18

5.1.3 Manipulating the Headers 20

5.1.4 CRC Distortion 21

5.1.5 Truncate the Frame 21

5.2 Frame Selection 22

5.3 Examples for Different Tasks 23

5.3.1 Drop SoC of the First POWERLINK Cycle 23

5.3.2 Cut PRes of NodeID 5 in POWERLINK Cycle Nr. 3 24

5.3.3 Manipulate SVID of ASnds 24

6 References 25

7 Figure Index 26

8 Table Index 27

# 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.

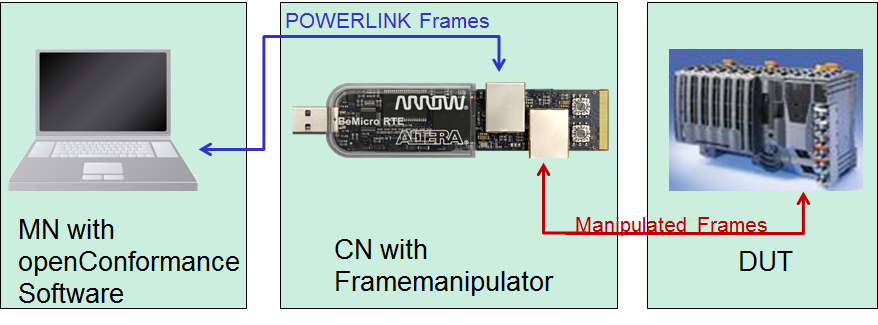


Figure 1: Test Arrangement with the Framemanipulator

## Framemanipulator Detection

The present Framemanipulator can be detected und differenced by numerous objects. The expected values of the FM version 0.1.0 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 | 0.1.0 |

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 be also 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 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 suited well for a finished product and 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

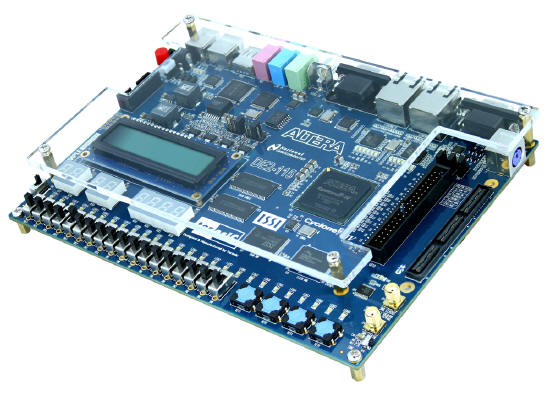


Figure 3: DE2-115 Development and Education Board [Altera]

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:

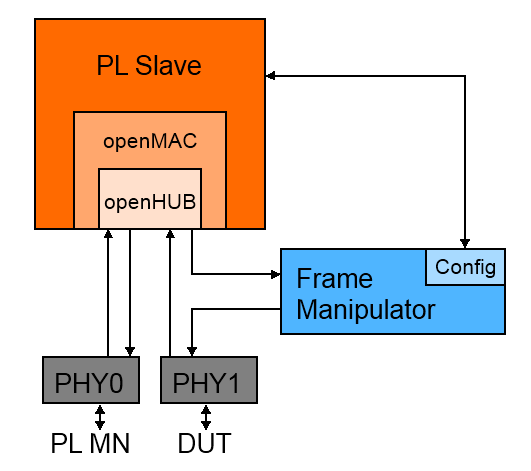


Figure 4: POWERLINK Slave with Framemanipulator

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 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

Ethernet frames 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.

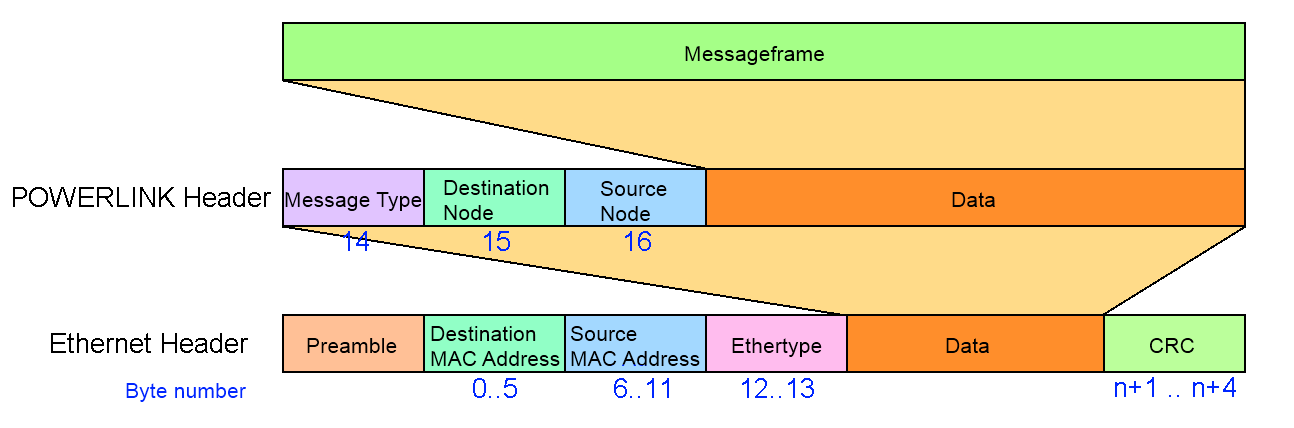


Figure 5: Ethernet and POWERLINK Header

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

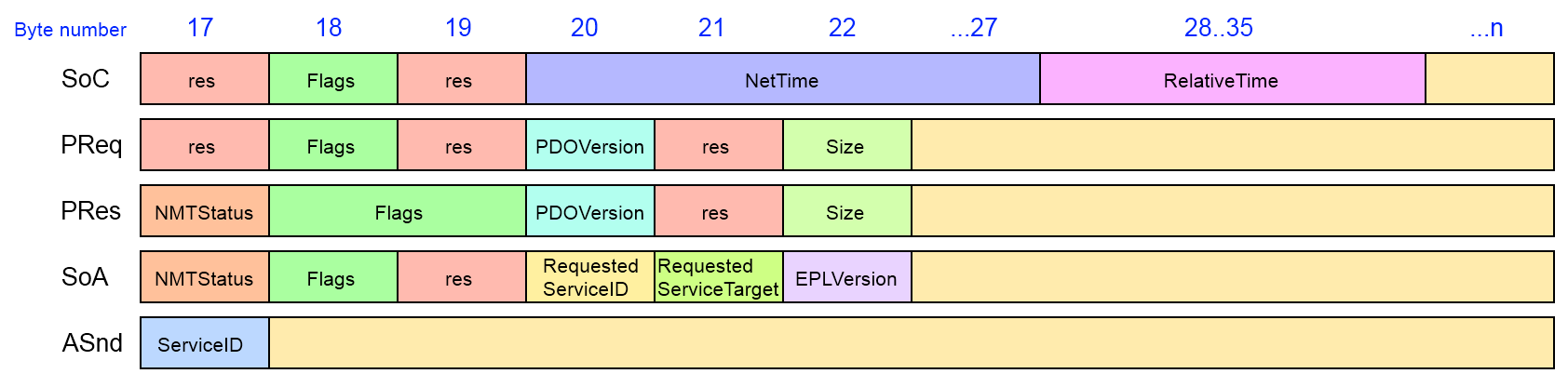


Figure 6: Structure of the POWERLINK Frames

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 Service ID or flags.

# Features

## Features for openConformance Test

### Dropping of specific frames in a specified sequence

The Frame-Manipulator (FM) Device is able to drop specific types of Ethernet Frames in a certain frame-type-count sequence. E.g. the FM should drop the 2nd and the 5th POWERLINK SoC frame at reception of a TBD start command.

### Delay of specified frames for a specified amount of time (in a specified sequence)

Same case like in 2.1.1 but instead of dropping the frame it is delayed.

### Manipulation of POWERLINK-Header fields (in a specified sequence)

Same case like in 2.1.1 but instead of dropping the frame its content is manipulated.

The FM is informed about the modified data with Byte-Offset / Bit-Offset / Value / (Size) of the Ethernet data to be modified.

### Manipulation of Ethernet-CRC (in a specified sequence)

Same case like in 2.1.1 but instead of dropping the frame its CRC is manipulated.

### Truncate payload of PReq-Frames

The FM is able to shorten a POWERLINK PReq frame.

E.g. the PReq size can be modified that it will not match the mapping size at the DUT.

### Example: Typical Test sequence of openConformance Test using FM e.g. “Loss of SoC”:

1. The openConformance tool configures the FM via POWERLINK (e.g. SDO. In a first attempt via PRes MN).
2. The openConformance tool starts dropping of SoC frames in a specific sequence-
3. Start of the actual test in the openConformance at a known POWERLINK cycle number + additional FM delay (e.g. 1 cycle), in a way that the FM and the openConformance test tool start at the same POWERLINK cycle.
4. The FM executes the dropping of SoC frames in a specific sequence.
5. The openConformance tool verifies the DUT NMT-Status, which shall have e.g. fallen back to PreOP1.

## The Different Manipulation Tasks

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

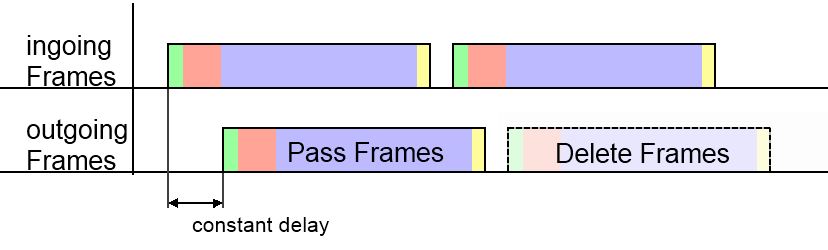


Figure 7: Tasks Pass and Delete Frames

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

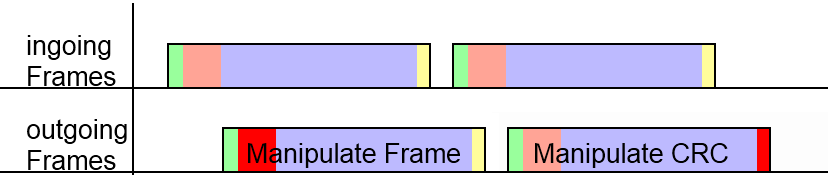


Figure 8: Tasks Manipulate Frame Header and CRC

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

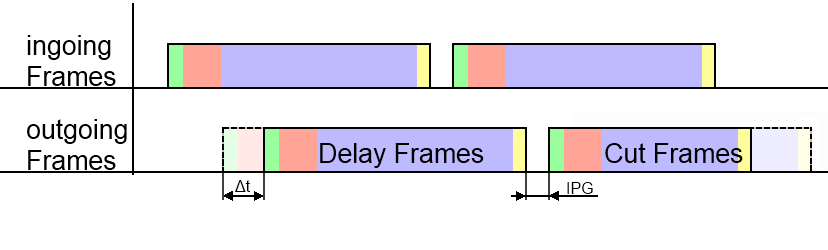


Figure 9: Tasks Delay and Truncate Frames

* **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

## 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 witch 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, a frame passes as less layers as possible and software based decisions on forwarding a frame was avoided.

### Framemanipulator Features

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

* Ethernet frame filters which signal 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 (required by 2.1.2)
* 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

# Automation Studio Integration

The Framemanipulator-Slave is connected for testing purpose 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*:

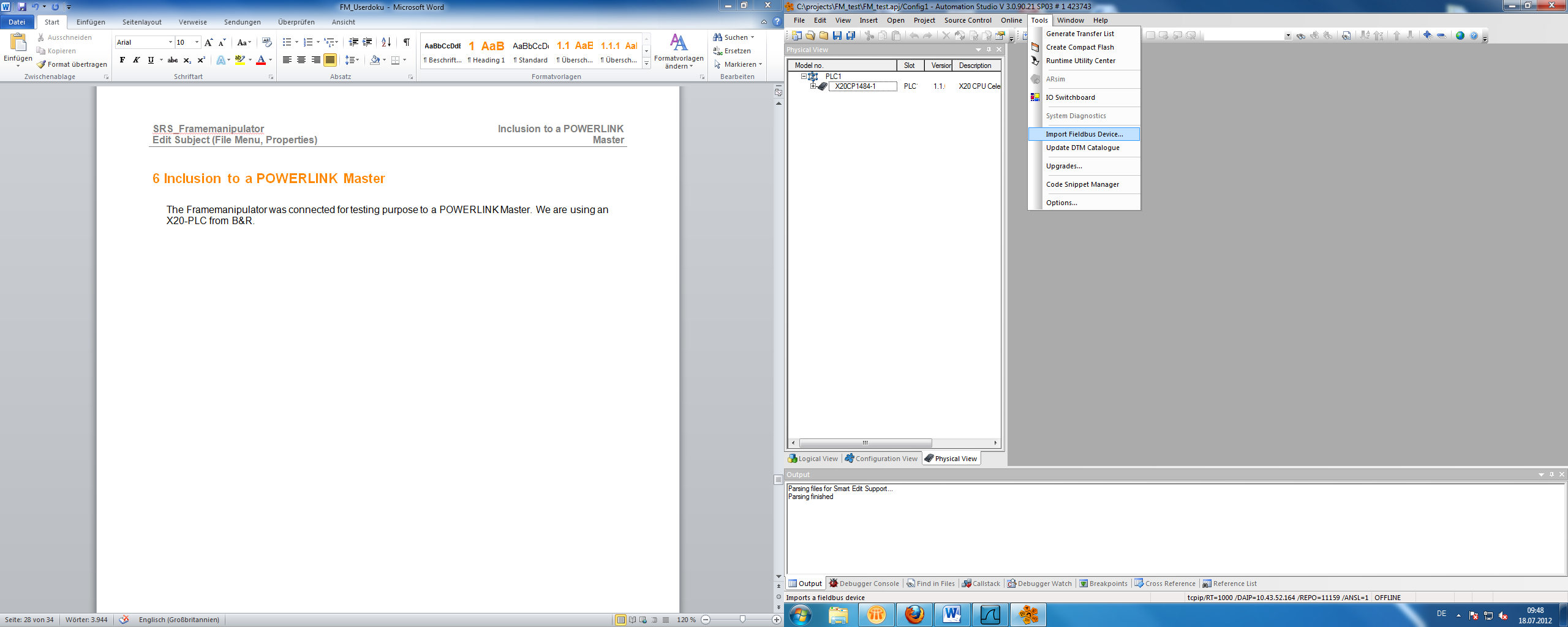


Figure 10: Automation Studio; Import Fieldbus Device

**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):

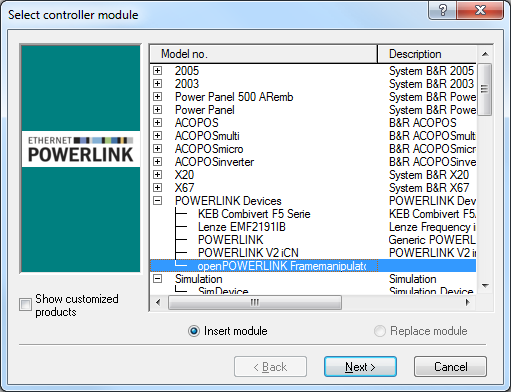


Figure 11: Selection of the CN with the Framemanipulator

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

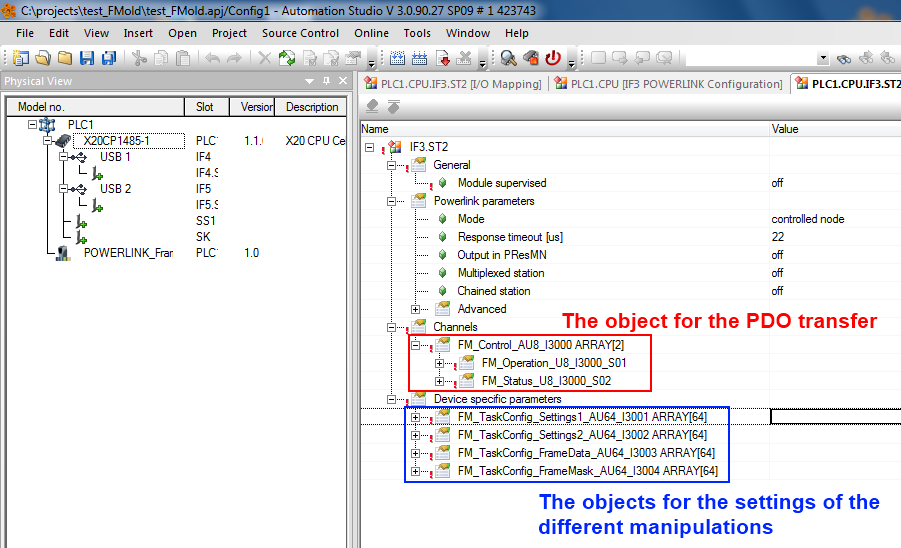


Figure 12: Automation Studio: I/O Configuration of the Framemanipulator CN

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 13).



Figure 13: Reverse order of bytes in the AS-configuration

# Control Interface

The Framemanipulator control is the interface to the openConformance software. It is connected to 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 subindex 2.

## Operations

The different operations of the Framemanipulator are sent via PReq to object 0x3000/1.

The following ones are currently available:

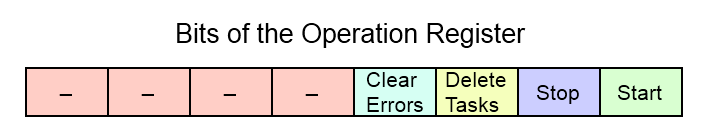


Figure 14: FM Operation Register

**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

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

* Set of all error bits (of chapter 4.2) to zero

The different operation bits can also be sent in combination with each other. All operations are activated by the positive edge of this signal.

## Status 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:

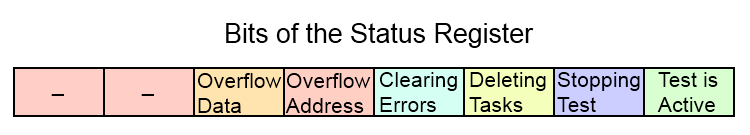


Figure 15: FM Status Register

**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 a storage of 2048 Bytes of data

*Solutions***:**

* Increasing of the frame-buffer size (*gSizeFrameBuffer*) of the Framemanipulator-design
* Reducing the number of stored frames
  + Reduction the delay-time
  + Changing the delay-type to e.g. 0x04 (*store only SoCs*)

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

## 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 16.*

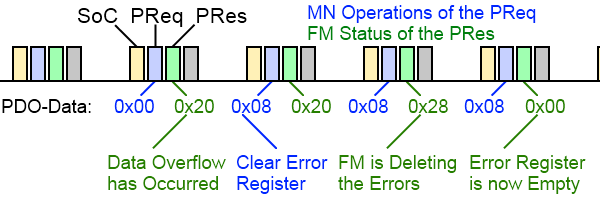


Figure 16: Clear Error Operation

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 receiving of the flags positive edge is acknowledged by the Framemanipulator 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:

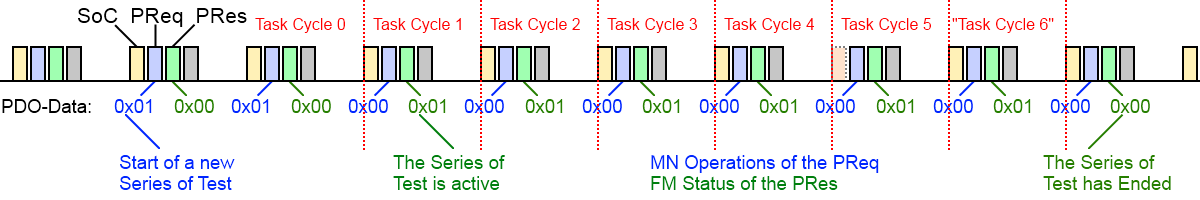


Figure 17: Start Task Operation with dropping the fifth SoC

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 0.1.0 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

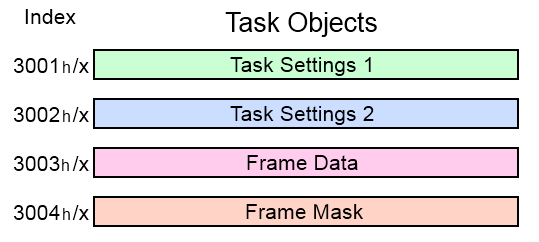


Figure 18: Objects of the Manipulation Tasks

The different manipulation tasks are configured by the subindices of the objects 0x3001 – 0x3004. Their data content is shown in *Figure 18*.

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 subindex number 8 will only process the tasks of subindex 1 to 7.

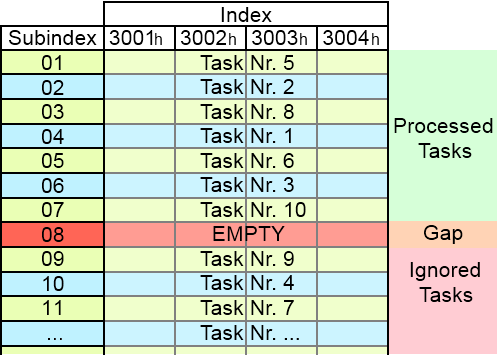


Figure 19: Example of the used Task Objects

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

## Task Settings

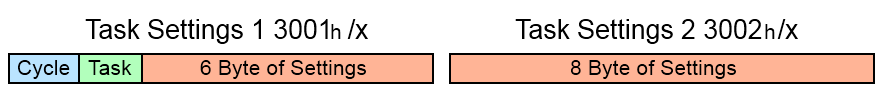


Figure 20: Basic Structure of the Task Setting Objects

The basic structure of the setting objects 0x3001 and 0x3002 is shown in *Figure 20*. 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

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:

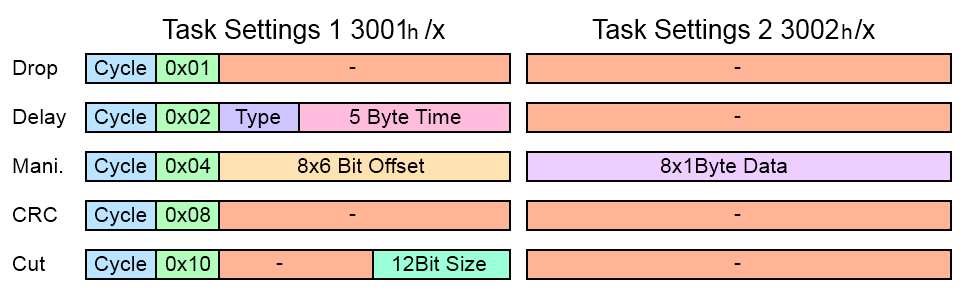


Figure 21: Whole Structure of the Task Setting Objects

### Dropping Frames

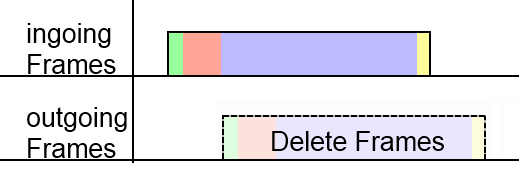


Figure 22: Task: Dropping Frames

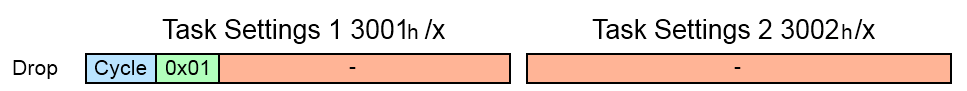


Figure 23: Task Setting of Dropping Frames

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

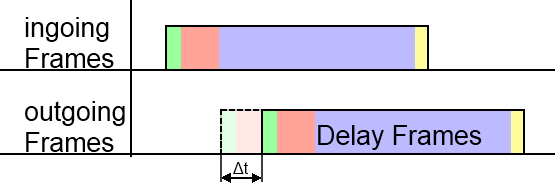


Figure 24: Task: Delaying Frames

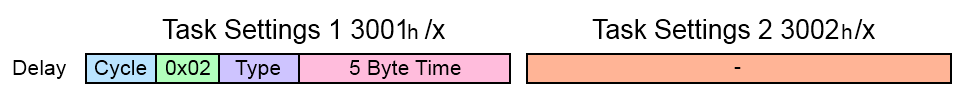


Figure 25: Task Setting of Delaying Frames

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 26* is delayed:

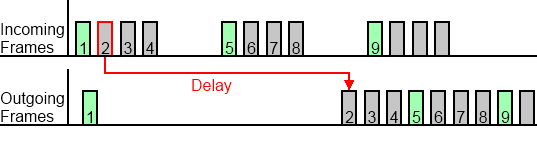
****

Figure 26: 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:

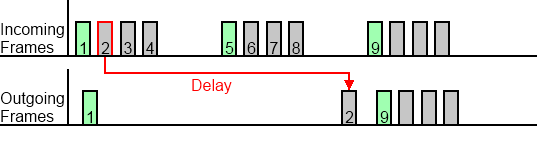


Figure 27: Delaying Frame 2; Dropping of all other Frames during the Delay

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

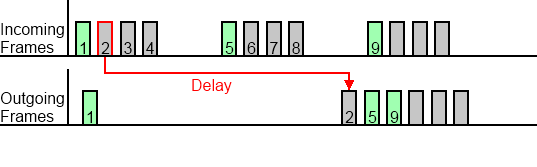


Figure 28: Delaying Frame 2; Storing of the Green Coloured SoC-Frames like Frame 5

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

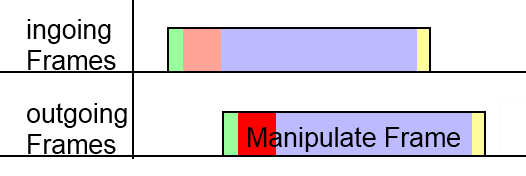


Figure 29: Task: Manipulating the Headers

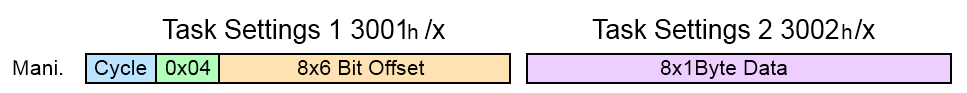


Figure 30: Task Setting of Manipulating the Headers

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

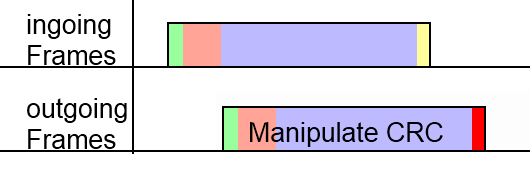


Figure 31: Task: CRC Distortion

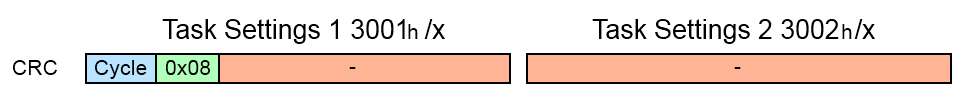


Figure 32: Task Setting of CRC Distortion

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

### Truncate the Frame

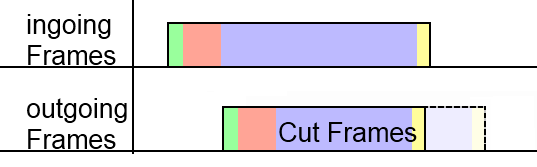


Figure 33: Task: Truncate the Frame

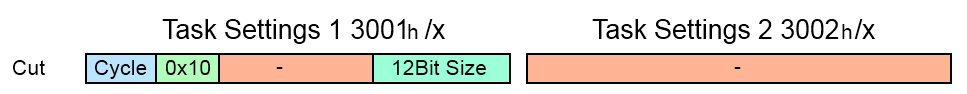


Figure 34: Task Setting of Frame Truncation

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

## Frame Selection

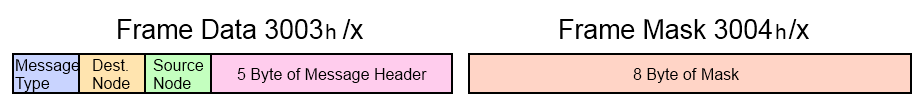


Figure 35: Basic Structure of the Responsible Objects for Filtering the Frames

The basic structure of the frame data object is shown in *Figure 35*. The Framemanipulator possesses an internal filter in hardware for the Ethertype. It only allows POWERLINK- (0x88AB) 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 (even if some like the upper two bytes of the SoC NetTime or the reserved bytes are never used):

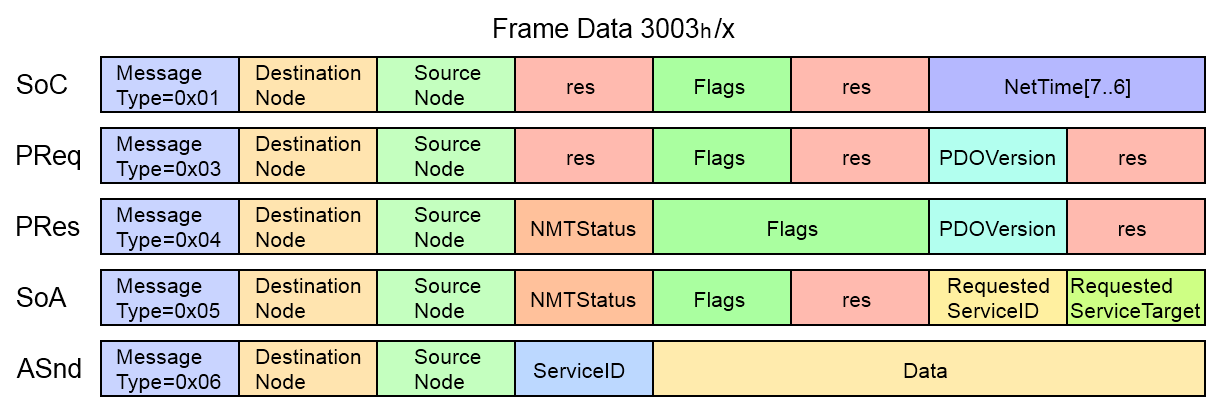


Figure 36: Filter Object Structure of the Different Messages

The mask of 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

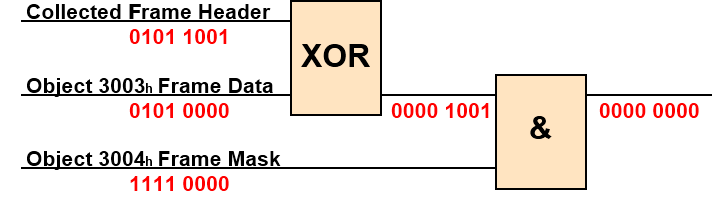


Figure 37: Filter Example: Data Conformance

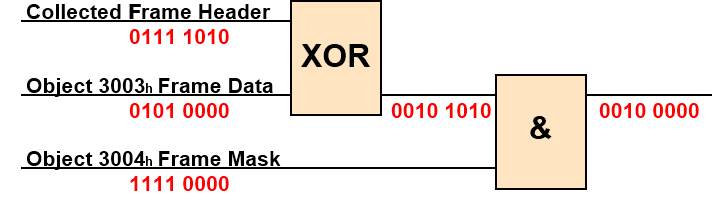


Figure 38: Filter Example: Data Conformance not fulfilled

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 are 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:

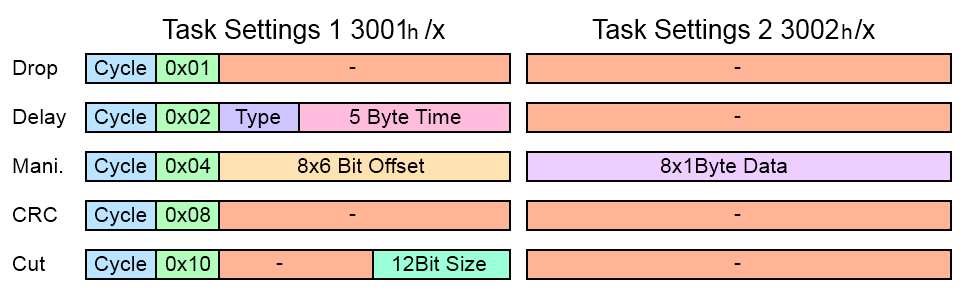


Figure 39: Task Setting Objects Structure

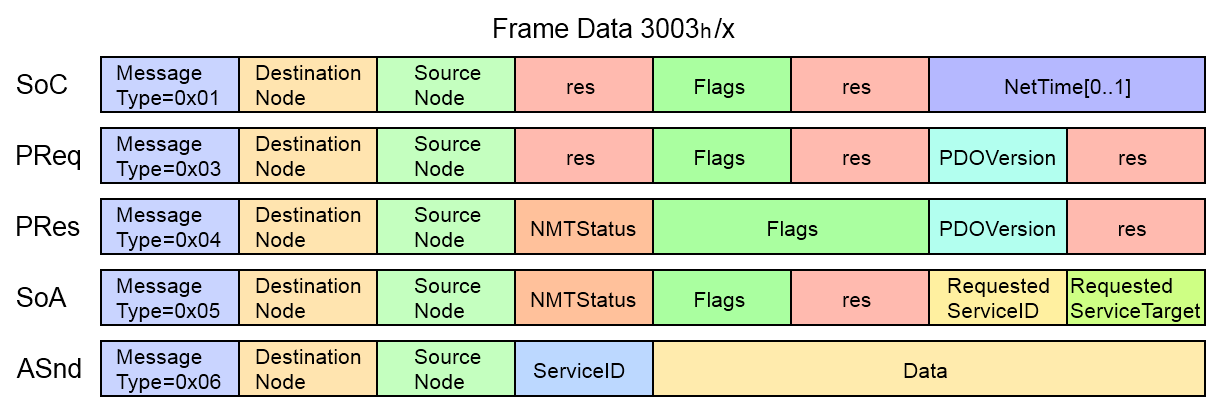


Figure 40: Frame Data Object Structure

### 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):

* **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 |

# 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 4

Figure 2: BeMicro RTE Stick [Arroweurope] 5

Figure 3: DE2-115 Development and Education Board [Altera] 5

Figure 4: POWERLINK Slave with Framemanipulator 6

Figure 5: Ethernet and POWERLINK Header 7

Figure 6: Structure of the POWERLINK Frames 7

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: Automation Studio; Import Fieldbus Device 11

Figure 11: Selection of the CN with the Framemanipulator 11

Figure 12: Automation Studio: I/O Configuration of the Framemanipulator CN 12

Figure 13: Reverse order of bytes in the AS-configuration 12

Figure 14: FM Operation Register 13

Figure 15: FM Status Register 14

Figure 16: Clear Error Operation 15

Figure 17: Start Task Operation with dropping the fifth SoC 15

Figure 18: Objects of the Manipulation Tasks 16

Figure 19: Example of the used Task Objects 16

Figure 20: Basic Structure of the Task Setting Objects 17

Figure 21: Whole Structure of the Task Setting Objects 17

Figure 22: Task: Dropping Frames 18

Figure 23: Task Setting of Dropping Frames 18

Figure 24: Task: Delaying Frames 18

Figure 25: Task Setting of Delaying Frames 18

Figure 26: Delaying Frame 2; Storing all other Frames until Frame 2 is Passed on 19

Figure 27: Delaying Frame 2; Dropping of all other Frames during the Delay 19

Figure 28: Delaying Frame 2; Storing of the Green Coloured SoC-Frames like Frame 5 19

Figure 29: Task: Manipulating the Headers 20

Figure 30: Task Setting of Manipulating the Headers 20

Figure 31: Task: CRC Distortion 21

Figure 32: Task Setting of CRC Distortion 21

Figure 33: Task: Truncate the Frame 21

Figure 34: Task Setting of Frame Truncation 21

Figure 35: Basic Structure of the Responsible Objects for Filtering the Frames 22

Figure 36: Filter Object Structure of the Different Messages 22

Figure 37: Filter Example: Data Conformance 22

Figure 38: Filter Example: Data Conformance not fulfilled 22

Figure 39: Task Setting Objects Structure 23

Figure 40: Frame Data Object Structure 23

# Table Index

Table 1: Versions 2

Table 2: Safety notices 2

Table 3: Objects for detection of the Framemanipulator 4

# 