|  |  |
| --- | --- |
| **User Documentation: The Framemanipulator** | |
| **SRS\_Framemanipulator** | |
| Zum Ändern von Titel und Thema klicken Sie im Menü Datei auf Eigenschaften - keinesfalls direkt eingeben. | |
|  |  |
| Date: | August 31, 2012 |
| Project Number: | AT-xx-xxxxxx |

1. Versions

|  | Version | Date | Comment | Edited by |
| --- | --- | --- | --- | --- |
|  | 1.0 | August 31, 2012 | First Edition |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Table 1: Versions

1. Distribution

|  | Name | Company, Department | Amount | Remarks |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Table 2: Distribution

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 3: Safety notices

1. Table of Contents

Introduction 5

1.1 General Features 5

1.2 Frame Structures 6

1.3 The Different Manipulation Tasks 7

2 Framemanipulator 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 Device Features 9

2.2.1 Framemanipulator Configuration 9

2.2.1.1 Control the FM by using MN-PRes Payload: 9

2.2.1.2 Configuration using SDO Transfer: 9

2.2.2 Ethernet Traffic 9

2.2.3 Framemanipulator Features 9

3 System Overview 10

4 Hardware Reference Platforms 11

4.1 BeMicro RTE Stick 11

4.2 INK-Board 11

5 FPGA Implementation 12

5.1 SOPC-Builder 12

5.2 Modification of the Schematic File 14

5.2.1 Example RMII: BeMicro 15

5.2.2 Workaround MII: INK-Bord 16

5.3 Modification of the POWERLINK Software 18

5.3.1 Integration of the Framemanipulator Sources 18

5.3.2 SDO-Transfer (objdict.h) 19

5.3.3 VendorID and Product Code (EplCfg.h) 20

5.3.4 PDO-Transfer (main.c) 21

5.3.5 XDC-File 22

5.4 Timing Characteristics 24

5.4.1 Delay of the Start-Signal 24

5.4.2 Delay of the writing the Task Objects 24

5.4.3 Delay Framemanipulator IP\_core 25

5.4.4 Delay of the Whole System 25

6 POWERLINK Network Configuration 27

7 Framemanipulator Control 29

7.1 Operations 29

7.2 Status Flags 30

7.3 Control Examples 31

8 Manipulation Tasks 32

8.1 Task Settings 33

8.1.1 Dropping Frames 34

8.1.2 Delaying Frames 34

8.1.3 Manipulating the Headers 36

8.1.4 CRC Distortion 37

8.1.5 Truncate the Frame 37

8.2 Frame Selection 38

8.3 Examples for Different Tasks 39

8.3.1 Drop SoC of the First POWERLINK Cycle 39

8.3.2 Cut PRes of NodeID 5 in POWERLINK Cycle Nr. 3 40

8.3.3 Manipulate SVID of ASnds 40

9 References 41

10 Figure Index 42

11 Table Index 44

12 Index 45

# Introduction

## General Features

An at runtime configurable Ethernet frame manipulator was developed. It serves as additional device for the POWERLINK openConformance test and extends it with accurate timing features and frame manipulation possibilities.

Furthermore the Ethernet frame manipulator 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 frame manipulator (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.

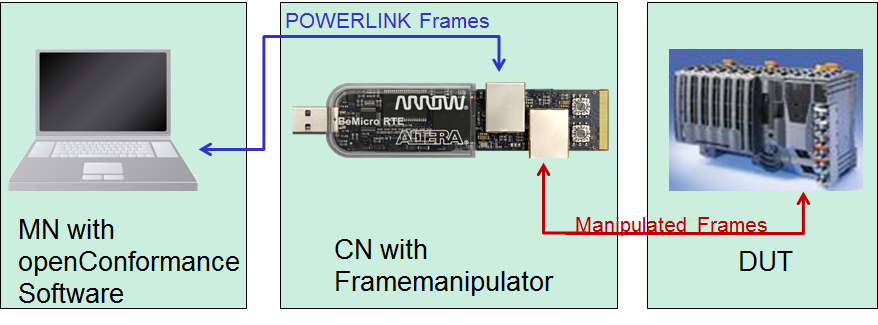


Figure 1: Test Arrangement with the Framemanipulator

## 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.

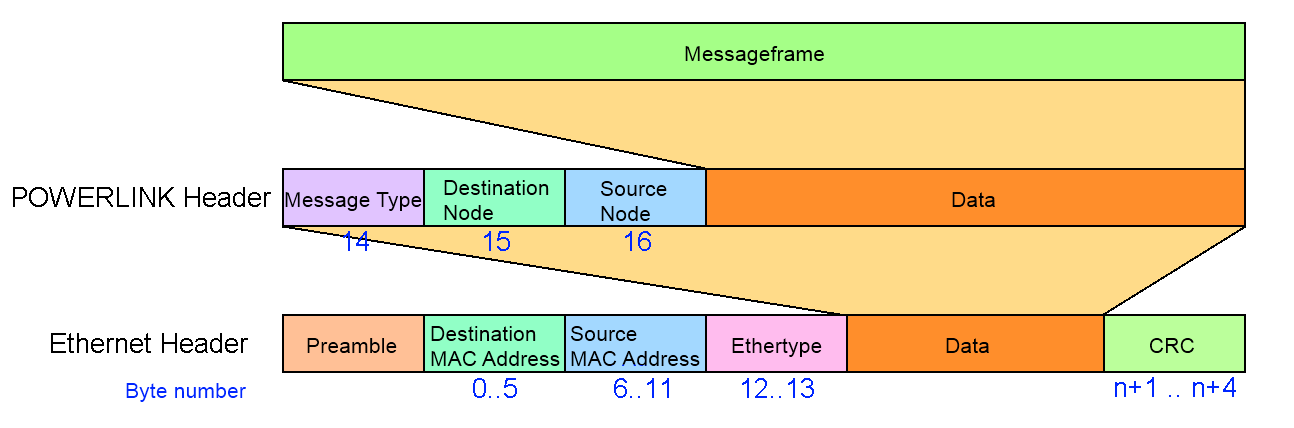


Figure 2: Ethernet and POWERLINK Header

An Ethertype of the value 88ABh 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.

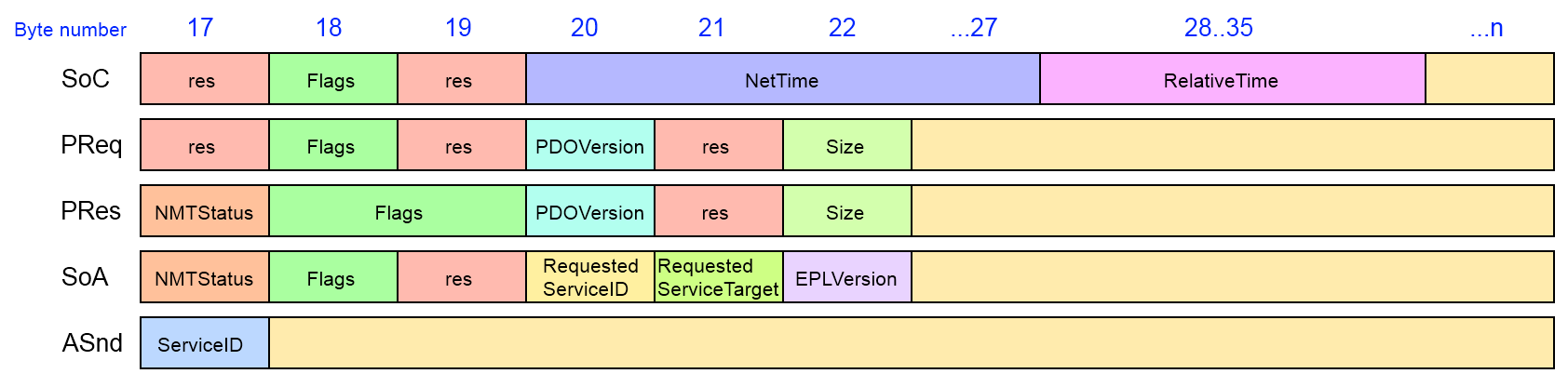


Figure 3: Structure of the POWERLINK Frames

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.

## 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

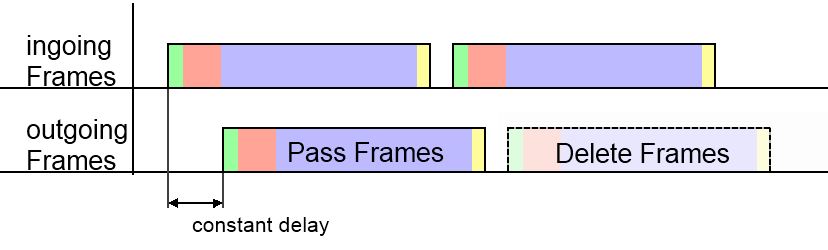


Figure 4: Tasks Pass and Delete Frames

* **Pass** unselected frames without any disturbances
  + Keep a constant delay of all outgoing frames (like in *Figure 4*)
* **Remove** selected frames

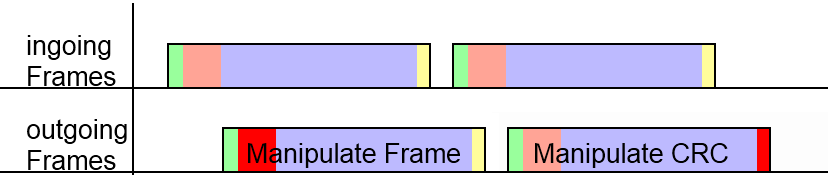


Figure 5: Tasks Manipulate Frame Header and CRC

* **Manipulate** the values of the frame headers with a valid CRC (like in *Figure 5*)
* **Distort** the CRC

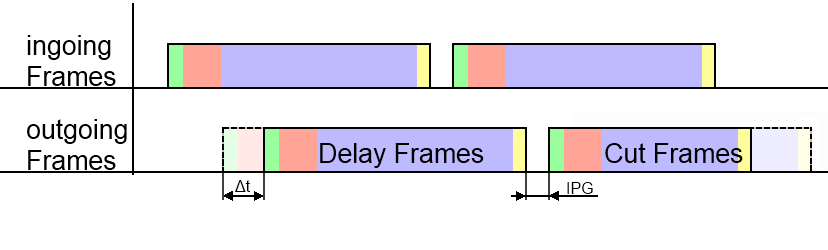


Figure 6: Tasks Delay and Truncate Frames

* **Delay** frames of a certain amount of time (like in *Figure 6*)
  + with keeping the IPG (Inter Package Gap) to the following frame
* **Truncate** frames with a valid CRC

# Framemanipulator 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.

## 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 frame manipulator module.

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

### Ethernet Traffic

The frame manipulator 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
* 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

# System Overview

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

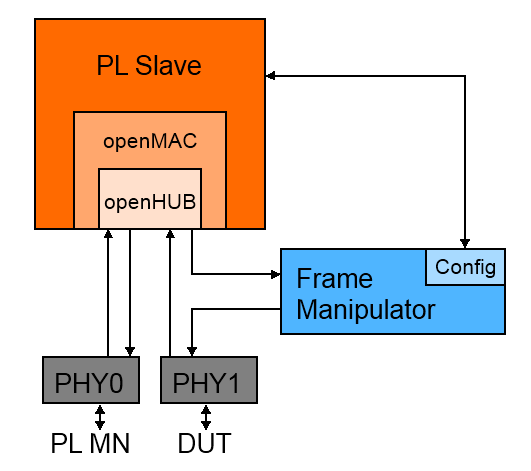


Figure 7: POWERLINK Slave with Framemanipulator

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

# Hardware Reference Platforms

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

## BeMicro RTE Stick



Figure 8: BeMicro RTE Stick [Source: 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

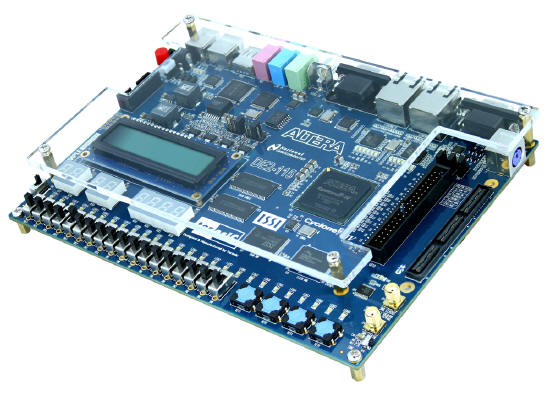


Figure 9: DE2-115 Development and Education Board [Source: Altera]

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.

It also has an expansion Header, where signals could be recorded with an external oscilloscope.

# FPGA Implementation

The Framemanipulator IP-Core is available in the “*release\_ip-core*” folder. It can be updated by using the “*update\_ip-core.bat”* of the main folder. It was designed for Altera FPGAs and contains the *\_hw.tcl*-files for the SOPC-Builder and Qsys.

It is also possible to port this IP-Core to other FPGAs. The *FrameManipulator.vhd* can be used as a separate component or as a source for other tcl-files.

This Framemanipulator hardware can be included to e.g. an existing design like the POWERLINK-CNDK (Controlled Node Development Kit) from B&R. The Direct-IO design of the BeMicro RTE Stick is used with Quartus II 10.1 SP1 in the following example.

This project is also available in *“example\_CNDK\BeMicro\_Q10*”.

We start with the modification of the SOPC-Design by adding the “*FrameManipulator\_sopc\_hw.tcl”*.

## SOPC-Builder

**1.** Copy the Framemanipulator HW files into the *IP\_core* folder and start the *SOPC-Builder* of the Quartus project.

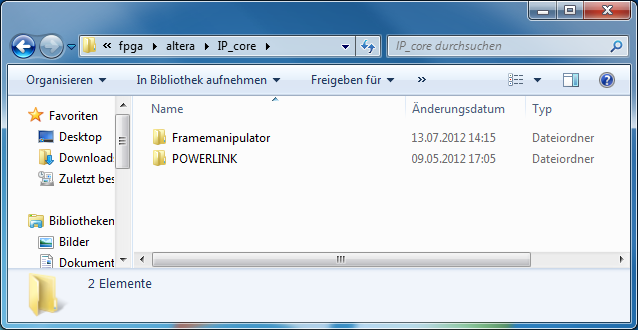


Figure 10: IP\_core Folder

**2.** Selected for the „*IP Search Path*“ by opening the *Options* window.

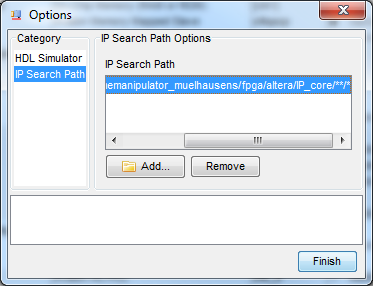


Figure 11: IP Search Path of the SOPC-Builder

**3.** Add a new Framemanipulator-component with “*TaskCount*=32”. This variable is for the maximal number of tasks/manipulations per test. (Normally a test doesn’t require more than ten manipulations)

The size of the frame data storage is defined by the generic *gBytesOfTheFrameBuffer*. A value of e.g. 1600 is converted with the two’s complement to a memory of 2048 bytes. This space is usable for saving about 34 different frames with a payload of 60 Bytes.

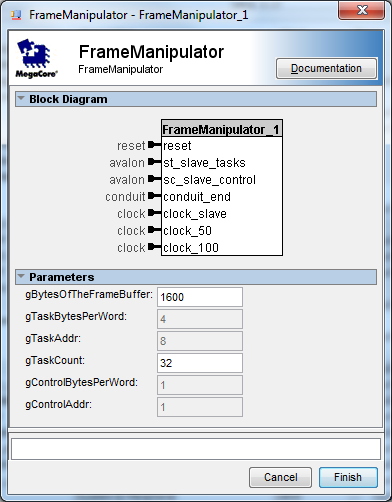


Figure 12: Adding a new Framemanipulator

**4.** Connect the two Avalon-Slaves to the *pcp\_cpu data\_master* and assign their clock to the one of the processor. The Framemanipulator itself uses the 50MHz clock:

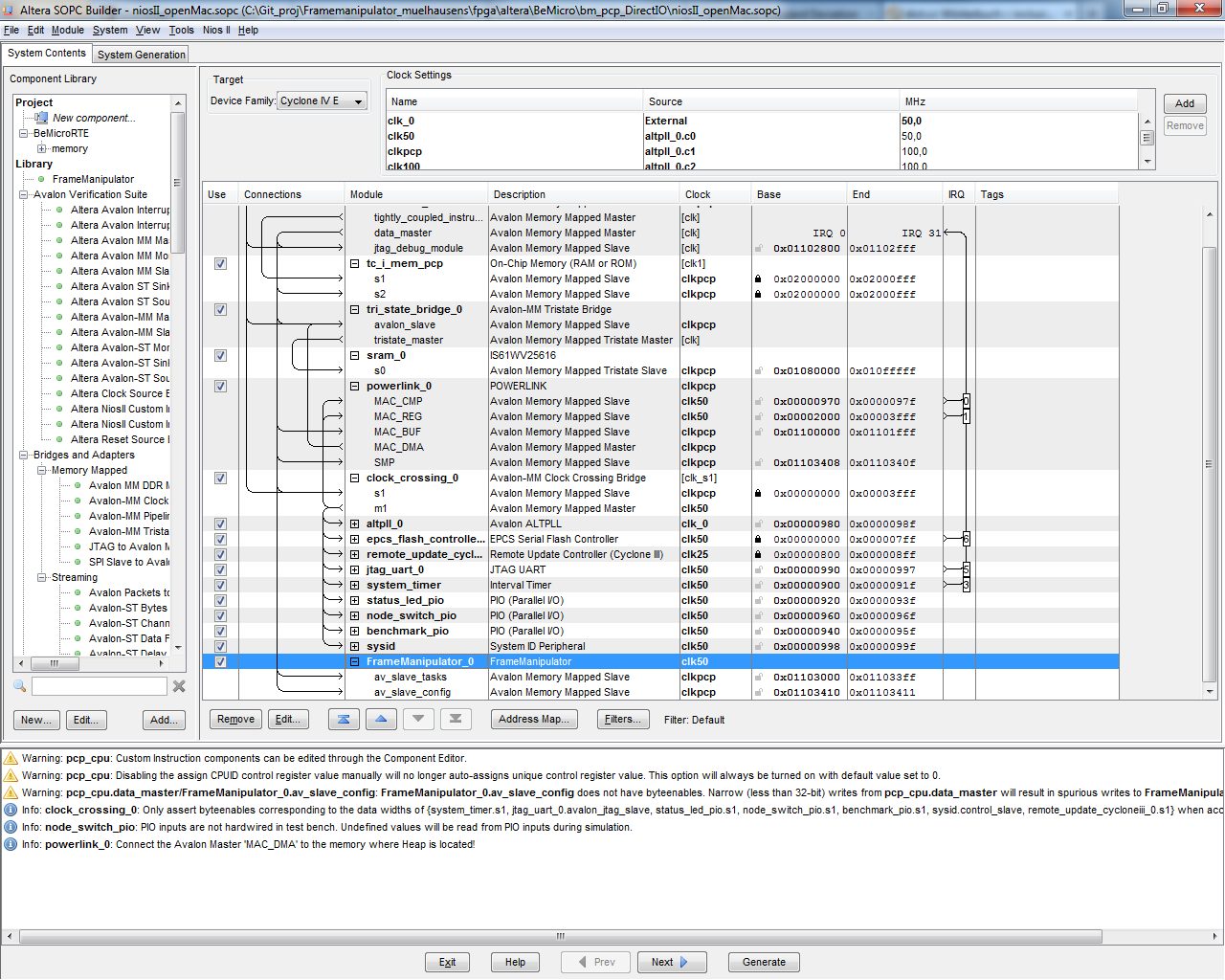


Figure 13: Framemanipulator SOPC Connection

*Av\_slave\_tasks* is an internal memory for the different manipulations of the frames. *Av\_slave\_control* is a register for different operations like starting the test.

**5.** The SOPC-Project can be generated after hitting “*System/Auto-Assign Base Addresses*”.

## Modification of the Schematic File

Update the SOPC-Block by doing a right click on it.

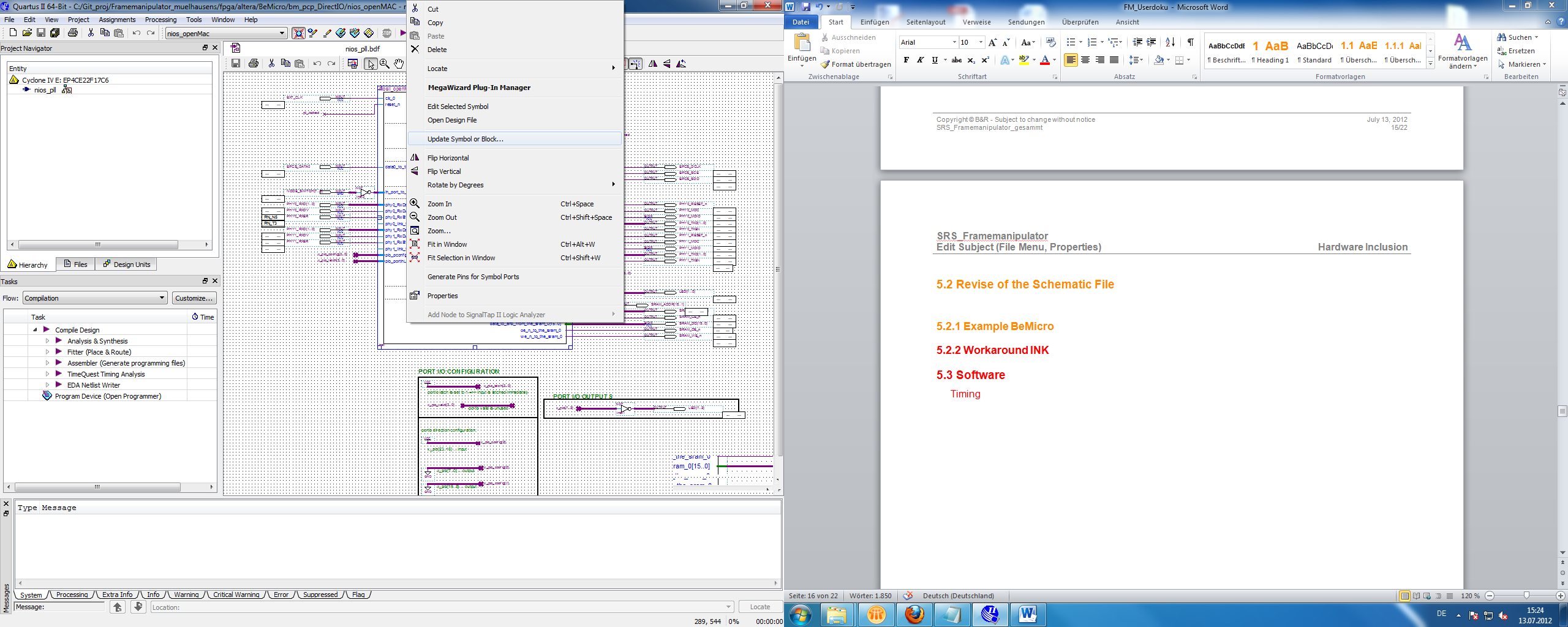


Figure 14: Update SOPC-Block in the Schematic File

The component will increase and could cause some errors in the current wiring, which has to be fixed at first.

The following rewiring depends on the used PHYs.

RMII PHYs have an internal FIFO for synchronization and use a data width of two with a clock rate of 50MHz.

MIIs have a data width of four with a clock rate of 25MHz. The FIFO for synchronization has to be implemented on the FPGA.

The Framemanipulator is built for development boards with RMII PHYs, like the BeMicro RTE. Thus, other boards with MIIs need a small workaround like in *chapter 5.2.2*.

After these changes, the project can be compiled and uploaded to the FPGA.

### Example RMII: BeMicro

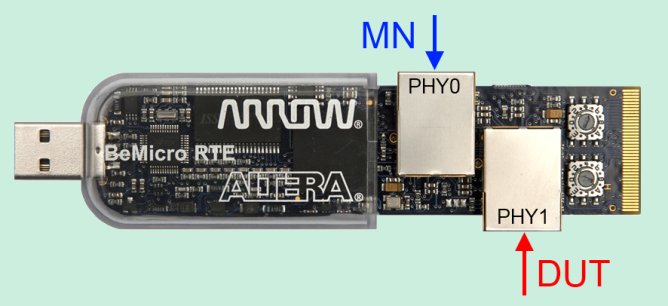


Figure 15: Connection of the BeMicro Stick

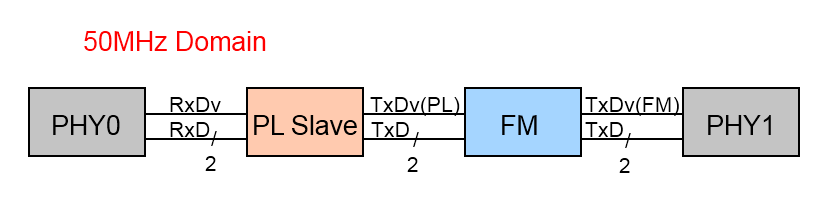


Figure 16: RMII Workflow of PL Frames from MN to DUT

Like shown in *Figure 15* and *Figure 16,* the Framemanipulator is connected in series to the outgoing Tx-Frames of the POWERLINK slave and PHY1 with the DUT. Therefore we have to cut the *TxData* and the *TxDataEnable* signal from the CN and connect it to the Framemanipulator output.

D-FF can also be added with a falling edge of a 100MHz clock for better timing results of the RMII.

The rewiring is depicted in *Figure 17*:

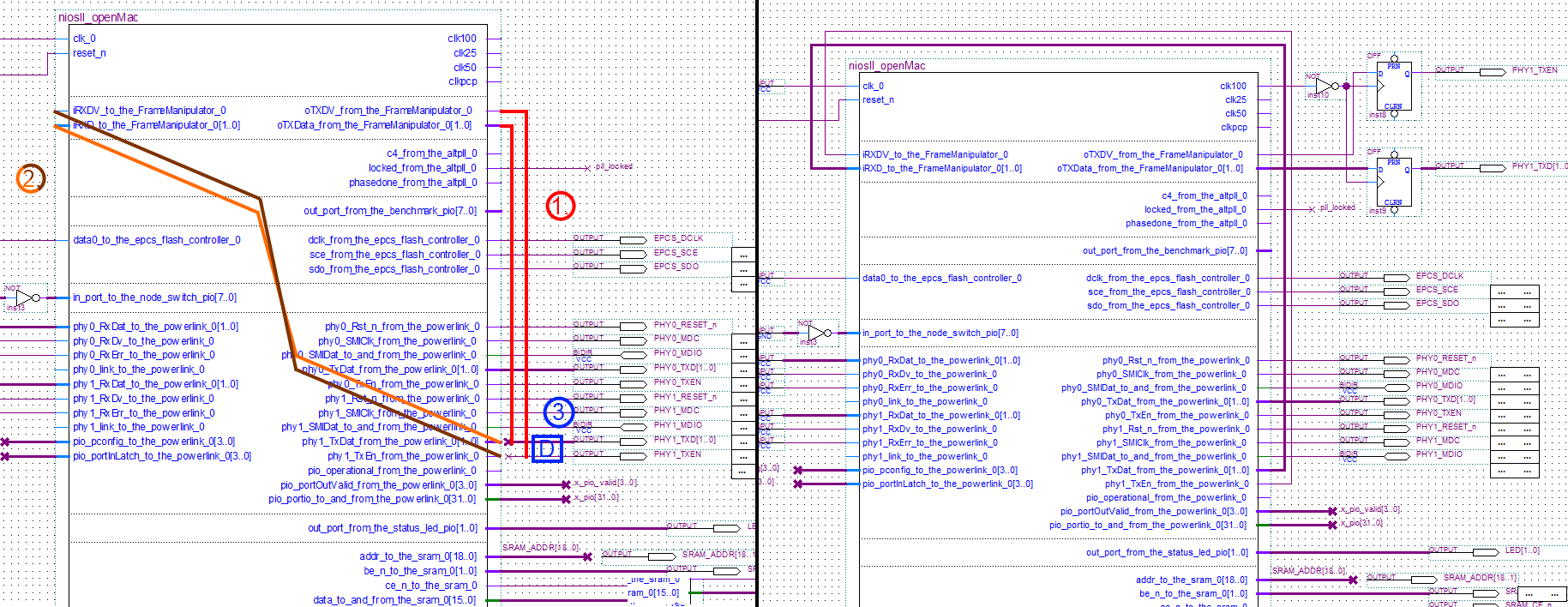


Figure 17: Rewiring RMII

**Rewiring steps:**

1. Remove TX-Data and TX-Data-Valid from PHY1 and connect them to the Framemanipulator output.
2. Connect the opened output of the PL-Slave to the Framemanipulator input
3. Add D-FF triggered on the falling edge of a 100MHz clock

### Workaround MII: INK-Bord

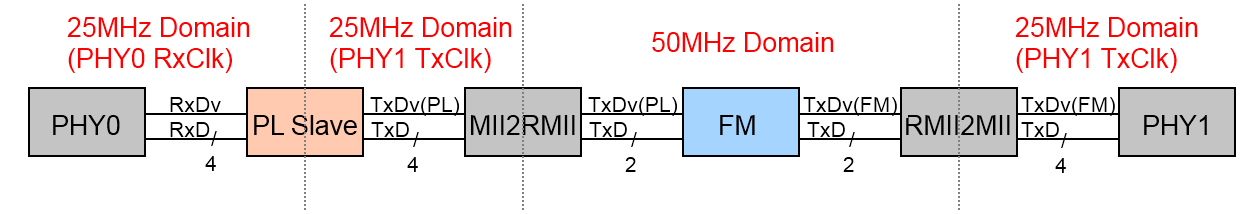


Figure 18: MII Workflow of PL Frames from MN to DUT with Workaround

The Workaround for Development-Boards with MII PHYs (like the INK-Board) is done by converting and synchronizing the frame to the 50MHz clock domain and back to the 25MHz of the MII PHYs. The workflow with the different components is depicted above.

Implement the *rmii2mii.vhd* converter of the POWERLINK IP-core and rewire the schematic:

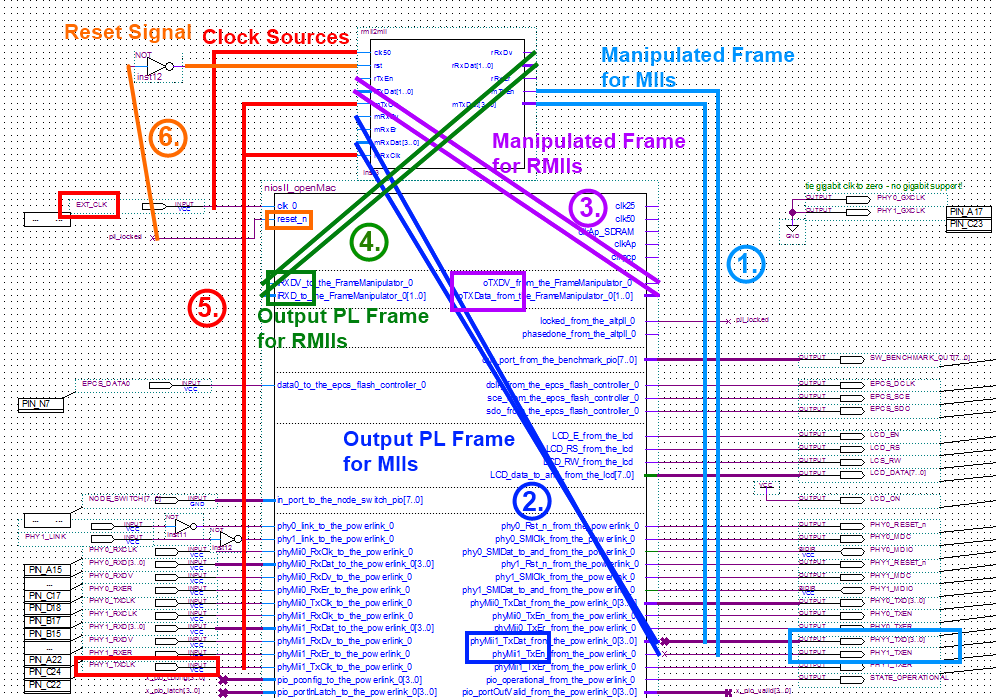


Figure 19: Rewiring the Schematic with rmii2mii Converter - 1

**Rewiring steps:**

1. Connect TX-Data and TX-Data-Valid signal of PHY1 to the corresponding rmii2mii output
2. Rewire the PL-Slave output to the mRxDat and mRxDv input of the rmii2mii-component
3. Connect the FM-output to the rTxDat and rTxEn input of the rmii2mii converter
4. Wire the remaining data output of the rmii2mii to the FM input
5. Support the converter with the PHY1 clock and a 50MHz clock source
6. Connect it to the reset (or like shown to an inverted reset\_n)

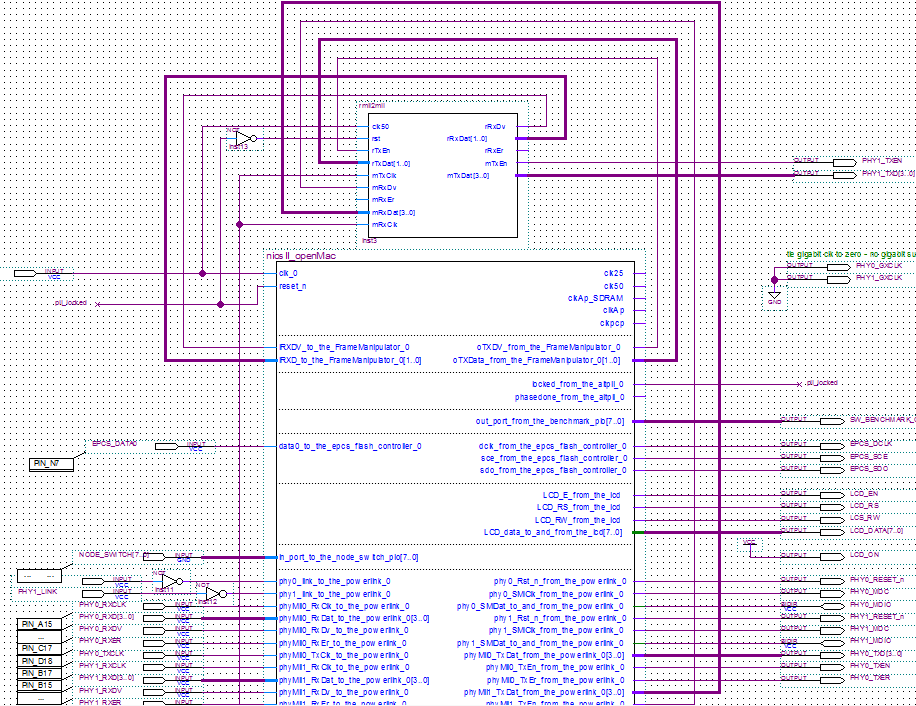


Figure 20: Rewiring the schematic with rmii2mii converter - 2

## Modification of the POWERLINK Software

The Framemanipulator will be configured with the POWERLINK software. Therefore some changes are necessary.

The Framemanipulator header- and c-source-files should also be copied to the project directory (here: “*\powerlink\pcp\_DirectIO”)*.

### Integration of the Framemanipulator Sources

The integration of the Framemanipulator.c to the BeMicro has to be done in the create-this-app of the Direct-IO project directory. It has to be added next to the other source files:

|  |
| --- |
| …  ###############################################################################  # source files  SRCFILES="main.c \  lcd.c \  Framemanipulator.c \  ${GENERIC\_DIR}/target/altera/fpgaCfg.c \  ${GENERIC\_DIR}/target/altera/fwUpdate.c \  ${GENERIC\_DIR}/target/altera/boot.c \  ${GENERIC\_DIR}/utils/crc32.c \  ${OBJDICTDIR}/Objdict.c \  ${EDRV\_DIR}/openmac/source/EdrvOpenMac.c \  ${EDRV\_DIR}/openmac/source/EplTimerSynck\_OpenMac.c \  ${EDRV\_DIR}/openmac/source/EplTgtTimeStamp\_OpenMac.c \  ${EDRV\_DIR}/openmac/source/omethlib.c \  ${EDRV\_DIR}/openmac/source/omethlibint.c \  ${POWERLINKDIR}/EplStack/EplDllk.c \  ${POWERLINKDIR}/EplStack/EplDllkCal.c \  ${POWERLINKDIR}/EplStack/EplDlluCal.c \  ${POWERLINKDIR}/EplStack/EplLedu.c \  ${POWERLINKDIR}/EplStack/EplEventk.c \  ${POWERLINKDIR}/EplStack/EplEventu.c \  ${POWERLINKDIR}/EplStack/EplNmtk.c \  ${POWERLINKDIR}/EplStack/EplNmtu.c \  ${POWERLINKDIR}/EplStack/EplNmtCnu.c \  ${POWERLINKDIR}/EplStack/EplPdok.c \  ${POWERLINKDIR}/EplStack/EplPdokCal.c \  ${POWERLINKDIR}/EplStack/EplPdou.c \  … |

### SDO-Transfer (objdict.h)

The settings of the different Framemanipulator tasks are done via SDO. We have to open the proper *objdict.h*. For Direct-IO designs it is located in the *\objDicts\Direct\_IO* folder.

The different objects for the tasks are configured in the *objdict.h* and linked to the callback *FMConfigObdAccess()*. Once the POWERLINK Slave receives the SDO-interrupt, this function is invoked and transfers the data to the Framemanipulator memory.

First we have to link the Framemanipulator-header at the beginning of the *objdict.h* to make the callback function known.

**#include** "Framemanipulator.h"

In this example, we created objects for up to 64 tasks. Since we defined the FM component of the SOPC-Builder with a “*TaskCount”* of 32, only 32 tasks are available. The number of available tasks can be read from the objects 3001h-3004h subindex 0.

The following objects have to be added to the *objdict.h* to the manufacturer specific area:

|  |
| --- |
| …  EPL\_OBD\_BEGIN\_PART\_MANUFACTURER ()  …  //Control Register (Operations and Status)  EPL\_OBD\_BEGIN\_INDEX\_RAM(0x3000, 0x03, NULL)  EPL\_OBD\_SUBINDEX\_RAM\_VAR(0x3000, 0x00, kEplObdTypUInt8, kEplObdAccConst, tEplObdUnsigned8, Number\_Entries, 0x02)  EPL\_OBD\_SUBINDEX\_RAM\_USERDEF(0x3000, 0x01, kEplObdTypUInt8, kEplObdAccVPRW, tEplObdUnsigned8, FM\_Operations, 0x00)  EPL\_OBD\_SUBINDEX\_RAM\_USERDEF(0x3000, 0x02, kEplObdTypUInt8, kEplObdAccVPR, tEplObdUnsigned8, FM\_Status, 0x00)  EPL\_OBD\_END\_INDEX(0x3000)  //Obj 1 => Manipulation Data 1  EPL\_OBD\_BEGIN\_INDEX\_RAM(0x3001, 0x41, FMConfigObdAccess) //64+1 Entry=41h  EPL\_OBD\_SUBINDEX\_RAM\_VAR(0x3001, 0x00, kEplObdTypUInt8, kEplObdAccSRW, tEplObdUnsigned8, Number\_Mani\_1, 0x40) //64  EPL\_OBD\_SUBINDEX\_RAM\_VAR(0x3001, 0x01, kEplObdTypUInt64, kEplObdAccSRW, tEplObdUnsigned64, Mani1\_Task, 0x00LL)  EPL\_OBD\_SUBINDEX\_RAM\_VAR(0x3001, 0x02, kEplObdTypUInt64, kEplObdAccSRW, tEplObdUnsigned64, Mani1\_Task, 0x00LL)  …  EPL\_OBD\_SUBINDEX\_RAM\_VAR(0x3001, 0x3F, kEplObdTypUInt64, kEplObdAccSRW, tEplObdUnsigned64, Mani1\_Task, 0x00LL)  EPL\_OBD\_SUBINDEX\_RAM\_VAR(0x3001, 0x40, kEplObdTypUInt64, kEplObdAccSRW, tEplObdUnsigned64, Mani1\_Task, 0x00LL)  EPL\_OBD\_END\_INDEX(0x3001)  //Obj 2 => Manipulation Data 2  EPL\_OBD\_BEGIN\_INDEX\_RAM(0x3002, 0x41, FMConfigObdAccess)  EPL\_OBD\_SUBINDEX\_RAM\_VAR(0x3002, 0x00, kEplObdTypUInt8, kEplObdAccSRW, tEplObdUnsigned8, Number\_Mani\_2, 0x40)  EPL\_OBD\_SUBINDEX\_RAM\_VAR(0x3002, 0x01, kEplObdTypUInt64, kEplObdAccSRW, tEplObdUnsigned64, Mani2\_Task, 0x00LL)  …  EPL\_OBD\_SUBINDEX\_RAM\_VAR(0x3002, 0x40, kEplObdTypUInt64, kEplObdAccSRW, tEplObdUnsigned64, Mani2\_Task, 0x00LL)  EPL\_OBD\_END\_INDEX(0x3002)  //Obj 3 => Framedata  EPL\_OBD\_BEGIN\_INDEX\_RAM(0x3003, 0x41, FMConfigObdAccess)  EPL\_OBD\_SUBINDEX\_RAM\_VAR(0x3003, 0x00, kEplObdTypUInt8, kEplObdAccSRW, tEplObdUnsigned8, Number\_Framedata, 0x40)  EPL\_OBD\_SUBINDEX\_RAM\_VAR(0x3003, 0x01, kEplObdTypUInt64, kEplObdAccSRW, tEplObdUnsigned64, Framedata\_Task, 0x00LL)  …  EPL\_OBD\_SUBINDEX\_RAM\_VAR(0x3003, 0x40, kEplObdTypUInt64, kEplObdAccSRW, tEplObdUnsigned64, Framedata\_Task, 0x00LL)  EPL\_OBD\_END\_INDEX(0x3003)  //Obj 4 => Framemask  EPL\_OBD\_BEGIN\_INDEX\_RAM(0x3004, 0x41, FMConfigObdAccess)  EPL\_OBD\_SUBINDEX\_RAM\_VAR(0x3004, 0x00, kEplObdTypUInt8, kEplObdAccSRW, tEplObdUnsigned8, Number\_Framemask, 0x40)  EPL\_OBD\_SUBINDEX\_RAM\_VAR(0x3004, 0x01, kEplObdTypUInt64, kEplObdAccSRW, tEplObdUnsigned64, Framemask\_Task, 0x00LL)  …  EPL\_OBD\_SUBINDEX\_RAM\_VAR(0x3004, 0x40, kEplObdTypUInt64, kEplObdAccSRW, tEplObdUnsigned64, Framemask\_Task, 0x00LL)  EPL\_OBD\_END\_INDEX(0x3004)  …  EPL\_OBD\_END\_PART ()  … |

The 3000h control object is explained later on in chapter 7.1*Operations.*

We also created a pre-defined mapping for the PDO-transfer to object 3000h subindex 1 (PReq FM operations) and 2 (PRes FM status).

|  |
| --- |
| …  // Object 1600h: PDO\_RxMappParam\_00h\_AU64  EPL\_OBD\_BEGIN\_INDEX\_RAM(0x1600, 0x1A, EplPdouCbObdAccess)  EPL\_OBD\_SUBINDEX\_RAM\_VAR(0x1600, 0x00, kEplObdTypUInt8, kEplObdAccRW, tEplObdUnsigned8, NumberOfEntries, 0x01)  EPL\_OBD\_SUBINDEX\_RAM\_VAR(0x1600, 0x01, kEplObdTypUInt64, kEplObdAccRW, tEplObdUnsigned64, ObjectMapping, 0x0008000000013000)  EPL\_OBD\_SUBINDEX\_RAM\_VAR(0x1600, 0x02, kEplObdTypUInt64, kEplObdAccRW, tEplObdUnsigned64, ObjectMapping, 0x00LL)  …  // Object 1A00h: PDO\_TxMappParam\_00h\_AU64  EPL\_OBD\_BEGIN\_INDEX\_RAM(0x1A00, 0x1A, EplPdouCbObdAccess)  EPL\_OBD\_SUBINDEX\_RAM\_VAR(0x1A00, 0x00, kEplObdTypUInt8, kEplObdAccRW, tEplObdUnsigned8, NumberOfEntries, 0x01)  EPL\_OBD\_SUBINDEX\_RAM\_VAR(0x1A00, 0x01, kEplObdTypUInt64, kEplObdAccRW, tEplObdUnsigned64, ObjectMapping, 0x0008000000023000)  EPL\_OBD\_SUBINDEX\_RAM\_VAR(0x1A00, 0x02, kEplObdTypUInt64, kEplObdAccRW, tEplObdUnsigned64, ObjectMapping, 0x00LL)  … |

### VendorID and Product Code (EplCfg.h)

The Framemanipulator receives a VendorID and a product code to be recognized by the openConformance MN. These values are available in object 1018h subindex 1 and 2.

But instead of updating the object in the *objdict.h*, we use the *EplCfg.h* of the CNDK project folder:

|  |
| --- |
| …  // identification parameters  #define CONFIG\_IDENT\_PRODUCT\_CODE 0x00000064  #define CONFIG\_IDENT\_REVISION 0x10020  #define CONFIG\_IDENT\_VENDOR\_ID 0x0100006C  #define CONFIG\_IDENT\_SERIAL\_NUMBER 0  #define CONFIG\_IDENT\_DEVICE\_NAME "POWERLINK FM"  … |

### PDO-Transfer (main.c)

Operations like the start signal of the test are sent via PReq to the FM control register. The current status of the Framemanipulator is also available in the control register and transferred back to the MN via PRes. The variables with the data are passed on as input parameter to the function *FM\_PDO\_Transfer()*.

In our example, we are creating the variable *FM\_Control[]* which is defined at the beginning of the *main.c* via:

BYTE FM\_Control[2];

It is linked the corresponding object 3000h via *EplApiLinkObject* of the *EplApiGeneric.c*:

int openPowerlink(WORD wNodeId\_p):

|  |
| --- |
| …  ObdSize = **sizeof**(FM\_Control[0]);  uiVarEntries = 2;  EplRet = EplApiLinkObject(0x3000, Control, &uiVarEntries, &ObdSize, 0x01);  **if** (EplRet != *kEplSuccessful*)  {  printf("linking FM vars... error\n\n");  **goto** ExitShutdown;  }  … |

We use *FM\_PDO\_Transfer()* in the cyclic function *AppCbSync():*

|  |
| --- |
| …  FM\_PDO\_Transfer(FM\_Control[FM\_ControlReg\_Operation],&FM\_Control[FM\_ControlReg\_Status]);  … |

To use this function we also have to include the header-file to the *main.c*:

**#include** "Framemanipulator.h"

After this last step, CNDK-projects can be compiled with executing the *rebuild.bat*. The program is transferred to the Nios 2 softcore via *run.bat*.

### XDC-File

We also update the .xdd/.xdc-file of the *\objDicts\Direct\_IO* folder for the POWERLINK Master by adding the Framemanipulator objects:

|  |
| --- |
| <!-- Manufacturer Specific Profile Area (0x2000 - 0x5FFF): may freely be used by the device manufacturer -->  …  <!-- FrameManipulator Area -->  <Object index="3000" name="FM\_Control\_AU8" objectType="8">  <SubObject subIndex="00" name="Number\_of\_Entries\_U8" objectType="7" dataType="0005" accessType="const" defaultValue="2" PDOmapping="no"/>  <SubObject subIndex="01" name="FM\_Operation\_U8" objectType="7" dataType="0005" accessType="rw" PDOmapping="RPDO" defaultValue="0x00"/>  <SubObject subIndex="02" name="FM\_Status\_U8" objectType="7" dataType="0005" accessType="ro" PDOmapping="TPDO" defaultValue="0x00"/>  </Object>    <!-- FrameManipulator Object -->  <Object index="3001" name="FM\_TaskConfig\_Settings1\_AU64" objectType="8">  <SubObject subIndex="00" name="Number\_Settings1\_U8" objectType="7" dataType="0005" accessType="const" defaultValue="64" PDOmapping="no"/>  <SubObject subIndex="01" name="Settings1\_U64" objectType="7" dataType="001B" accessType="rw" PDOmapping="no" defaultValue="0x0000000000000000"/>  <SubObject subIndex="02" name="Settings1\_U64" objectType="7" dataType="001B" accessType="rw" PDOmapping="no" defaultValue="0x0000000000000000"/>  <SubObject subIndex="03" name="Settings1\_U64" objectType="7" dataType="001B" accessType="rw" PDOmapping="no" defaultValue="0x0000000000000000"/>  …  <SubObject subIndex="3F" name="Settings1\_U64" objectType="7" dataType="001B" accessType="rw" PDOmapping="no" defaultValue="0x0000000000000000"/>  <SubObject subIndex="40" name="Settings1\_U64" objectType="7" dataType="001B" accessType="rw" PDOmapping="no" defaultValue="0x0000000000000000"/>  </Object>  <!-- FrameManipulator Object -->  <Object index="3002" name="FM\_ TaskConfig\_Settings\_AU64" objectType="8">  <SubObject subIndex="00" name="Number\_ Settings2\_U8" objectType="7" dataType="0005" accessType="const" defaultValue="64" PDOmapping="no"/>  <SubObject subIndex="01" name=" Settings2\_U64" objectType="7" dataType="001B" accessType="rw" PDOmapping="no" defaultValue="0x0000000000000000"/>  …  <SubObject subIndex="40" name=" Settings2\_U64" objectType="7" dataType="001B" accessType="rw" PDOmapping="no" defaultValue="0x0000000000000000"/>  </Object>  <!-- FrameManipulator Object -->  <Object index="3003" name="FM\_TaskConfig\_FrameData\_AU64" objectType="8">  <SubObject subIndex="00" name="Number\_FrameData\_U8" objectType="7" dataType="0005" accessType="const" defaultValue="64" PDOmapping="no"/>  <SubObject subIndex="01" name="FrameData\_U64" objectType="7" dataType="001B" accessType="rw" PDOmapping="no" defaultValue="0x0000000000000000"/>  …  <SubObject subIndex="40" name="FrameData\_U64" objectType="7" dataType="001B" accessType="rw" PDOmapping="no" defaultValue="0x0000000000000000"/>  </Object>    <!-- FrameManipulator Object -->  <Object index="3004" name="FM\_TaskConfig\_FrameMask\_AU64" objectType="8">  <SubObject subIndex="00" name="Number\_FrameMask\_U8" objectType="7" dataType="0005" accessType="const" defaultValue="64" PDOmapping="no"/>  <SubObject subIndex="01" name="FrameMask\_U64" objectType="7" dataType="001B" accessType="rw" PDOmapping="no" defaultValue="0x0000000000000000"/>  …  <SubObject subIndex="40" name="FrameMask\_U64" objectType="7" dataType="001B" accessType="rw" PDOmapping="no" defaultValue="0x0000000000000000"/>  </Object>  … |

We changed the default values object 1600h and 1800h subindex 0 and 1 to simplify the fixed PDO-mapping:

|  |
| --- |
| …  <Object index**="1600"** name**="PDO\_RxMappParam\_00h\_AU64"** objectType**="8">**  <SubObject subIndex**="00"** name**="NumberOfEntries"** objectType**="7"** dataType**="0005"** accessType**="rw"** PDOmapping**="no"** defaultValue**="1"/>**  <SubObject subIndex**="01"** name**="ObjectMapping"** objectType**="7"** dataType**="001B"** accessType**="rw**" PDOmapping**="no"** defaultValue**="0x0008000000013000"/>**  <SubObject subIndex**="02"** name**="ObjectMapping"** objectType**="7"** dataType**="001B"** accessType**="rw"** PDOmapping**="no"** defaultValue**="0x0000000000000000"/>**  …  <Object index**="1A00"** name**="PDO\_TxMappParam\_00h\_AU64"** objectType**="8">**  <SubObject subIndex**="00"** name**="NumberOfEntries"** objectType**="7"** dataType**="0005"** accessType**="rw"** PDOmapping**="no"** defaultValue**="1"/>**  <SubObject subIndex**="01"** name**="ObjectMapping"** objectType**="7"** dataType**="001B"** accessType**="rw"** PDOmapping**="no"** defaultValue**="0x0008000000023000"/>**  … |

We also update the different entries of the product name, as well as the object 1018h with the VendorID and product code:

|  |
| --- |
| …  <ProfileName>**POWERLINK FM01 CN device profile**</ProfileName>  …  <productName>**POWERLINK\_Framemanipulator**</productName>  …  fileName="**00000000\_POWERLINK\_CN\_FM01.xdc**"  …  <Object index**="1018"** name**="NMT\_IdentityObject\_REC"** objectType**="9">**  <SubObject subIndex**="00"** name**="NumberOfEntries"** objectType**="7"** dataType**="0005"** accessType**="const"** PDOmapping**="no"** defaultValue**="4"/>**  <SubObject subIndex**="01"** name**="VendorId\_U32"** objectType**="7"** dataType**="0007"** accessType**="const"** PDOmapping**="no**" defaultValue**="0x0100006C"/>**  <SubObject subIndex**="02"** name**="ProductCode\_U32"** objectType**="7"** dataType**="0007"** accessType**="const"** PDOmapping**="no"** defaultValue**="0x00000064"/>**  <SubObject subIndex**="03"** name**="RevisionNo\_U32"** objectType**="7"** dataType**="0007"** accessType**="const"** PDOmapping**="no"** defaultValue**="0x00010007"/>**  <SubObject subIndex**="04"** name**="SerialNo\_U32"** objectType**="7"** dataType**="0007"** accessType**="const"** PDOmapping**="no"** defaultValue**="0x00000000"/>**  </Object>  … |

## Timing Characteristics

Timing is an important point. Large delay or jitter could cause an unforeseen crash or other hitches of the DUT or the Framemanipulator itself.

The following chapters discuss the timing of the Framemanipulator itself and with the POWERLINK Slave of the used CDNK-design.

### Delay of the Start-Signal

The start signal of a series of tests and other operations of the Framemanipulator are done via PDO. Incoming PDOs are processed in the **same** POWERLINK cycle.

The shortest cycle time depends on the cyclic processes. A too short cycle time causes a timing error and causes the CN to leave the operational state.

After receiving the start signal, the test cycle is going to start in the following cycle:

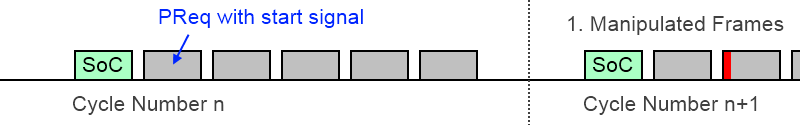


Figure 21: Start of Tests after receiving PReq with Start Signal

### Delay of the writing the Task Objects

The data of the task objects are sent via SDO. These are stored on an internal shared buffer and processed later on.

The callback-function of the Framemanipulator objects is marked by the benchmark function and can be traced with SignalTap.

Table 4: Used Commands for the Benchmark Function

|  |  |
| --- | --- |
| **#include** "Benchmark.h" | Inclusion of the benchmark-function |
| BENCHMARK\_SET(7); | Activate Bit number seven |
| BENCHMARK\_RESET(7); | Deactivate Bit number seven |

The first trace is done with a cycle time of 600µs:

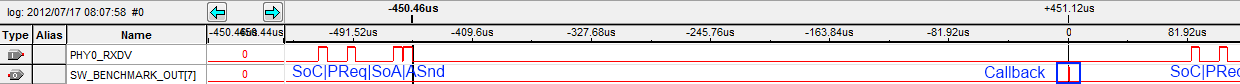


Figure 22: SignalTap Trace; 600µs; Delay of Callback Executing

The peak of the callback appears in the same cycle after about 450µs.

The next trace is done with a shorter cycle time of 400µs:

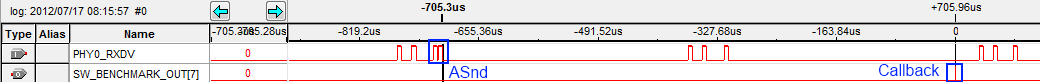


Figure 23: SignalTap Trace; 400µs; Delay of Callback Executing

The callback is processed in the following cycle, with about 700µs delay after receiving the ASnd.

Shorter cycle times or more complex functions of the CN could create an overflow of the shared buffer or a timing violation.

The callback delay of one or more cycles doesn’t affect the function of the Framemanipulator. The callback is always processed before the SDO acknowledge is sent to the Master Node.

### Delay Framemanipulator IP\_core

The delay of the Framemanipulator can be divided into collecting the data for the frame selection and the actual process time for comparing the filter objects with the header and staring the new frame.

The collection of all important Bytes is a constant value of 2460ns. It is a result of 8Byte Preamble, 22 Bytes of Ethernet-, POWERLINK-Header and other Flags and another 60ns process delay.

The process time depends on the number of used tasks (not the maximum number of possible tasks) plus 140ns of editing the frame. The delay is constant for the test cycles and is only changing during the rewriting of the task objects.

This delay can be calculated with:

Thus, a series of tests with ten manipulations leads to a total delay of 2,8µs without jitter. That’s about the half of the whole size of a SoC-frame.

### Delay of the Whole System

To receive the whole delay of the manipulated frames, we also have to add the delay of the POWERLINK Slave and its openHUB.

First we are measuring the delay of the standard RMII-BeMicro design:

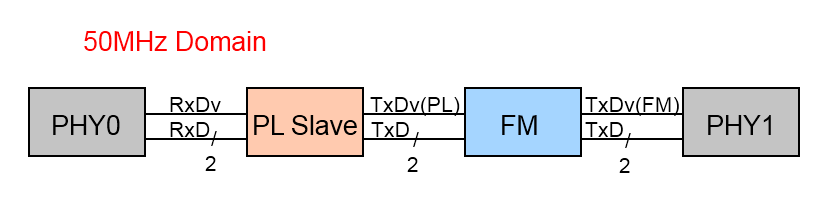


Figure 24: RMII Workflow of PL Frames from MN to DUT

We are using SignalTap and are measuring the time at the in- and output of the SOPC-component:

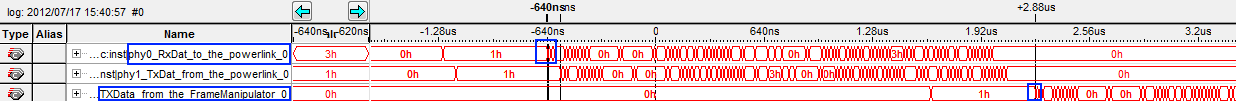


Figure 25: SignalTap; Incoming and Outgoing Frame; RMII; Delay of 2.88µs

The additional delay of the openHUB of the POWERLINK-Slave increases the delay to 2,88µs ().

We are also testing the workaround of the INK-Board with the MIIs with its additional converters:

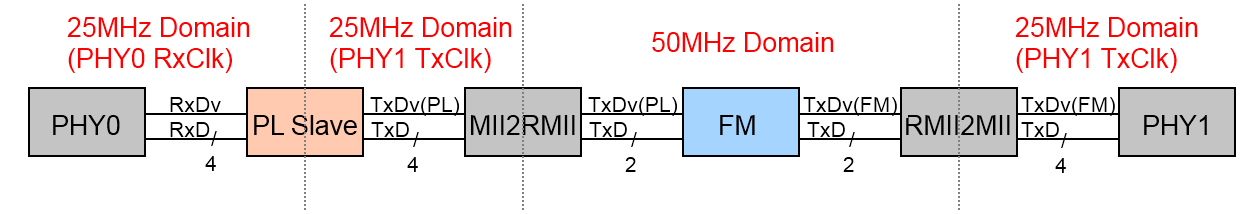


Figure 26: MII Workflow of PL Frames from MN to DUT

We are also tracing the delay of the external *mii2rmii* converter and the internal openHUB:

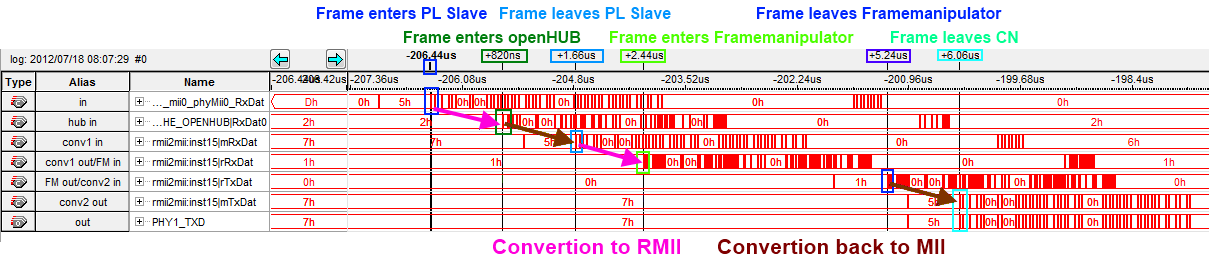


Figure 27: SignalTap; Incoming and Outgoing Frame; MII; Delay of 6.06µs

The delay of the MII design with workaround is more than twice as high as the one of the RMII design. The frames are converted four times (two for the PL-Slave and two for the FM) with a delay of about 0,8µs each.

()

# POWERLINK Network Configuration

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 created in chapter *5.3.5 XDC-File*:

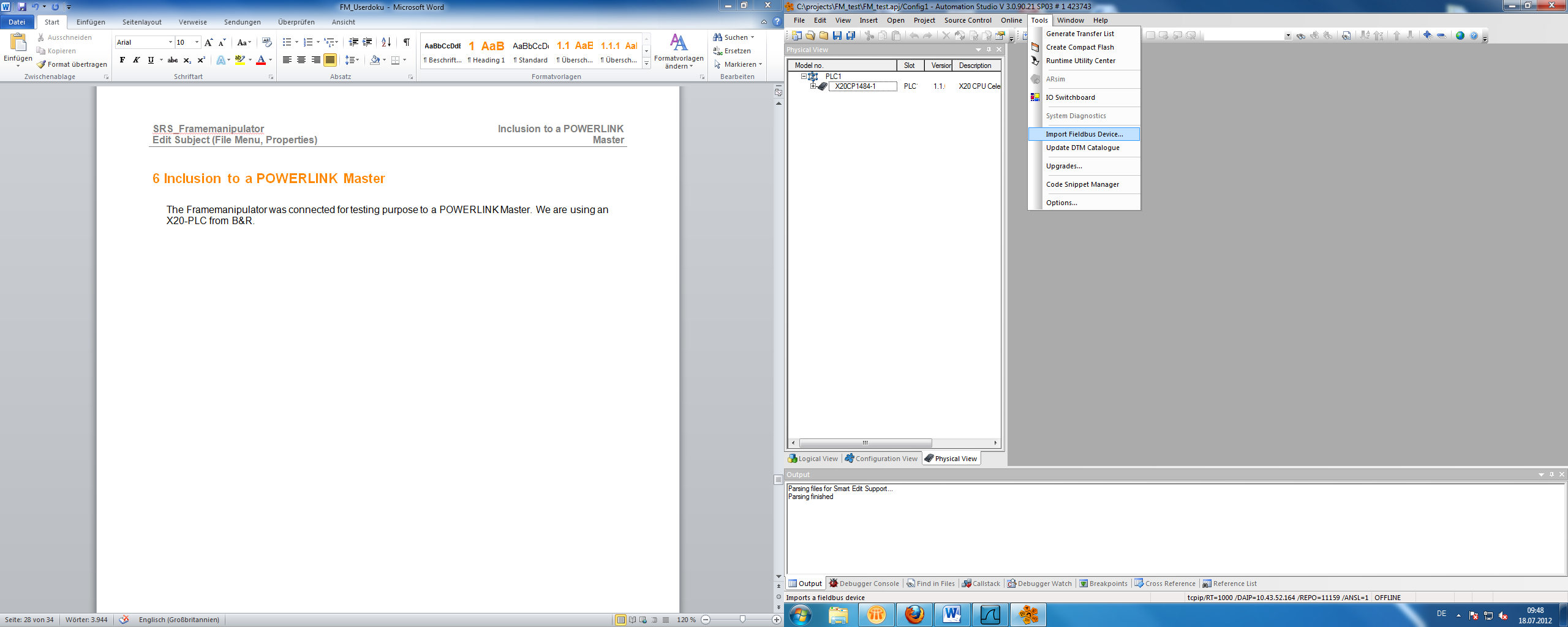


Figure 28: Automation Studio; Import Fieldbus Device

**2.** Go to the X20-PLC and select “Open POWERLINK” via right click to insert our CN with the Framemanipulator with e.g. NodeID 1 (as well as our DUT):

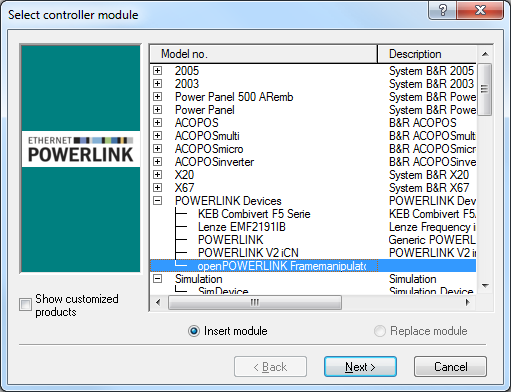


Figure 29: Selection of the CN with the Framemanipulator

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

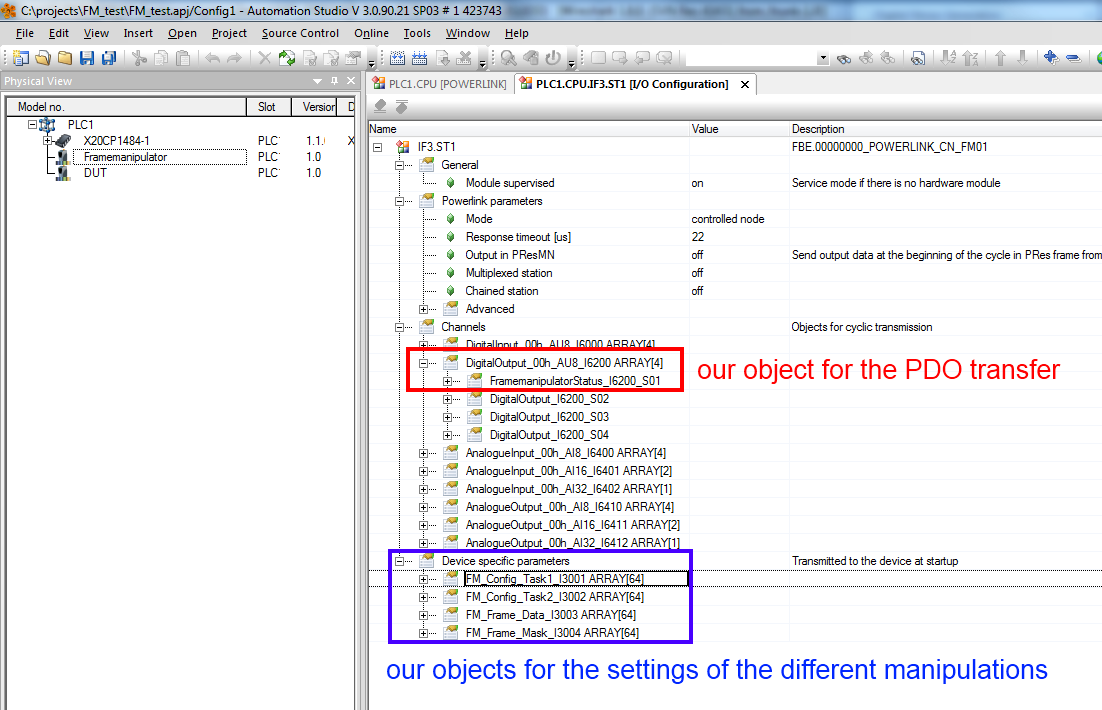


Figure 30: Automation Studio: I/O Configuration of the Framemanipulator CN

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.

# Framemanipulator Control

The Framemanipulator control is the interface to the openConformance software. It is connected to object 3000h and transfers the data via PDO.

The controlling is done via different operations of subindex 1. The resulting states and errors are collected in 3000h subindex 2.

## Operations

The different operations of the Framemanipulator are sent via PReq to object 3000h/1.

The following ones are currently available:

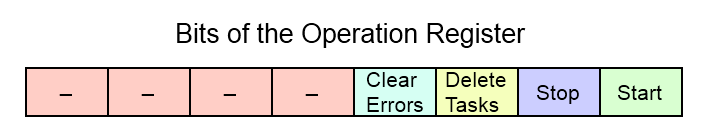


Figure 31: FM Operation Register

**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 7.2) 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 3000h/2.

The following ones are currently available:

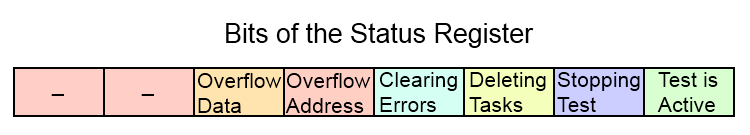


Figure 32: FM Status Register

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

*How to calculate the buffer sizes:*

The dependences of the two buffers are described with the following example:

* The SOPC generic *gSizeFrameBuffer* is defined by a value 1600 (like in chapter 5.1).
* Consequently, the data-buffer obtains a size of 2048 Byte (next value of the two’s complement)
* A data buffer filled with frames of the minimum size of 60Bytes contains up to 34 different frames (2048/60=34,...)
* Hence, the size of the Address-Buffer is specified to fit in the 34 frames => size for 64 frames

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 33.*

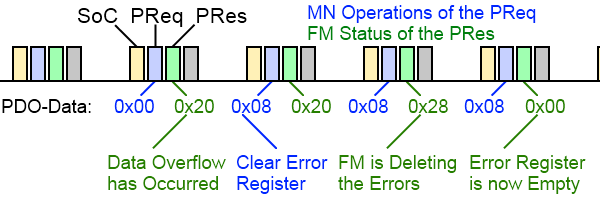


Figure 33: Clear Error Operation

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:

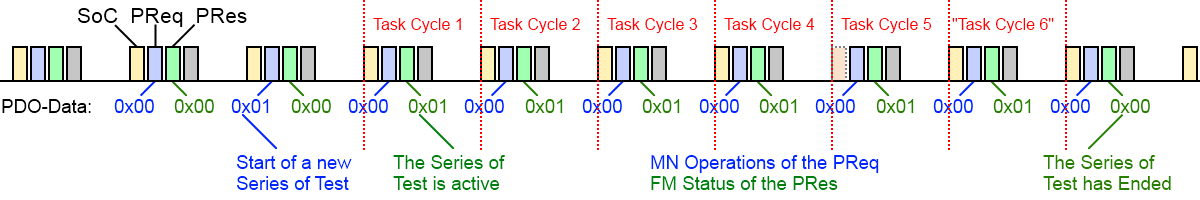


Figure 34: Start Task Operation with dropping the fifth SoC

After receiving the positive edge of the start flag, the Framemanipulator starts a series of test and responses with setting the corresponding bit.

The 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

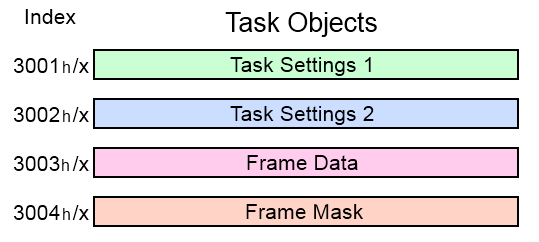


Figure 35: Objects of the Manipulation Tasks

The different manipulation tasks are configured by the subindices of the objects 3001h – 3004h. Their data content is shown in *Figure 35*.

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 3001h-3004h.

E.g. the first task has its settings in 3001h/1 and 3002h/1. The configuration of its frame-filter is contained in 3003h/1 and its corresponding mask in 3004h/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.

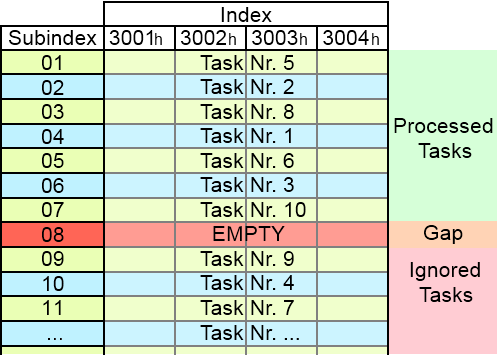


Figure 36: Example of the used Task Objects

Unused tasks can be deactivated by writing a cycle-number of 0 to the corresponding subobject of object 3001h.

## Task Settings

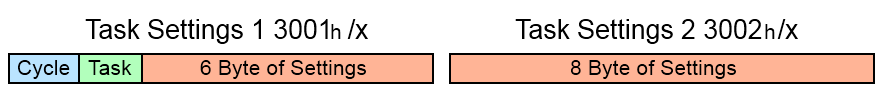


Figure 37: Basic Structure of the Task Setting Objects

The basic structure of the setting objects 3001h and 3002h is shown in *Figure 37*. The first byte of object 3001h 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:

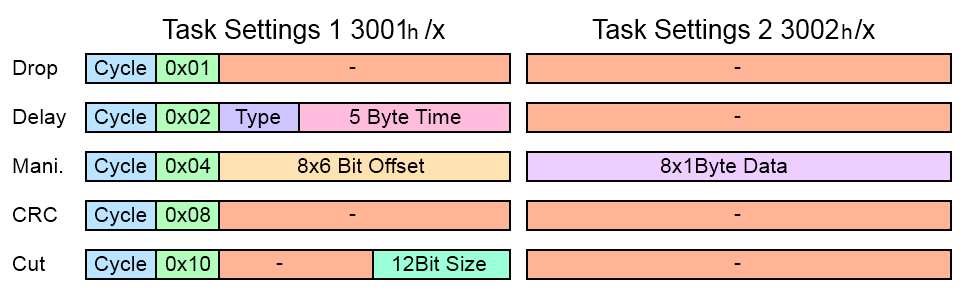


Figure 38: Whole Structure of the Task Setting Objects

### Dropping Frames

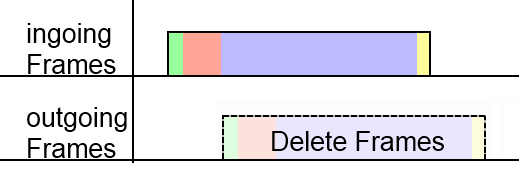


Figure 39: Task: Dropping Frames

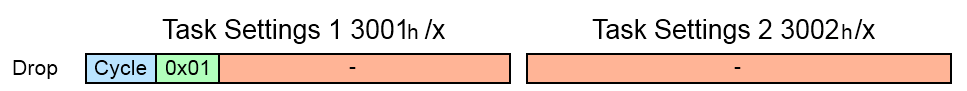


Figure 40: Task Setting of Dropping Frames

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

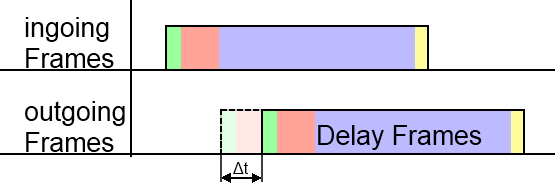


Figure 41: Task: Delaying Frames

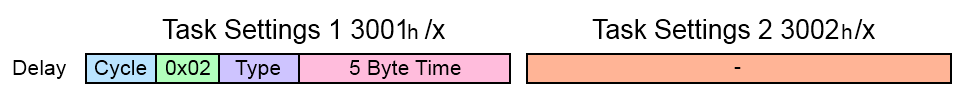


Figure 42: Task Setting of Delaying Frames

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 43* is delayed:

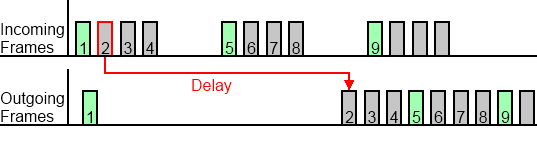
****

Figure 43: 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:

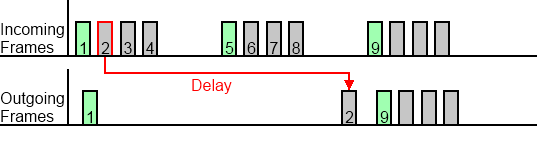


Figure 44: Delaying Frame 2; Dropping of all other Frames during the Delay

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

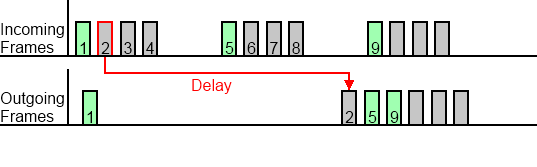


Figure 45: Delaying Frame 2; Storing of the Green Coloured SoC-Frames like Frame 5

The number of storable frames depends on the variable *gBytesOfTheFrameBuffer*, which is adjustable in the component of the SOPC-Builder design (like shown in chapter 5.1).

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

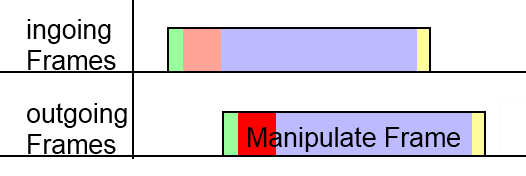


Figure 46: Task: Manipulating the Headers

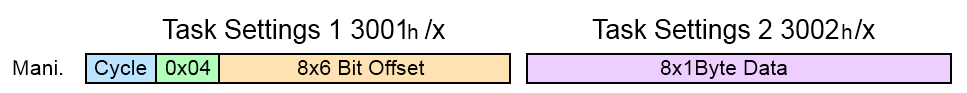


Figure 47: Task Setting of Manipulating the Headers

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 6Bit offset, and so on…

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

### CRC Distortion

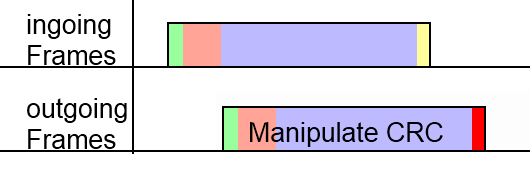


Figure 48: Task: CRC Distortion

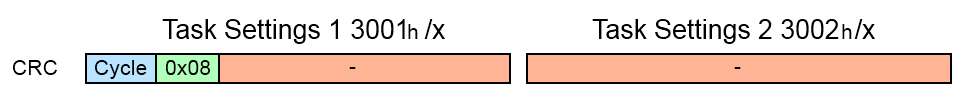


Figure 49: Task Setting of CRC Distortion

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

### Truncate the Frame

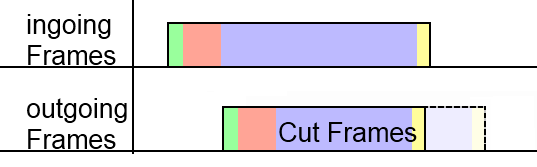


Figure 50: Task: Truncate the Frame

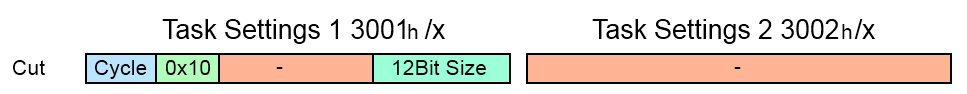


Figure 51: Task Setting of Frame Truncation

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

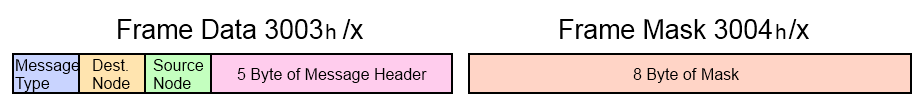


Figure 52: Basic Structure of the Responsible Objects for Filtering the Frames

The basic structure of the frame data object is shown in *Figure 52*. The Framemanipulator possesses an internal filter in hardware for the Ethertype. It only allows POWERLINK- (88ABh) and IP- (0800h, 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):

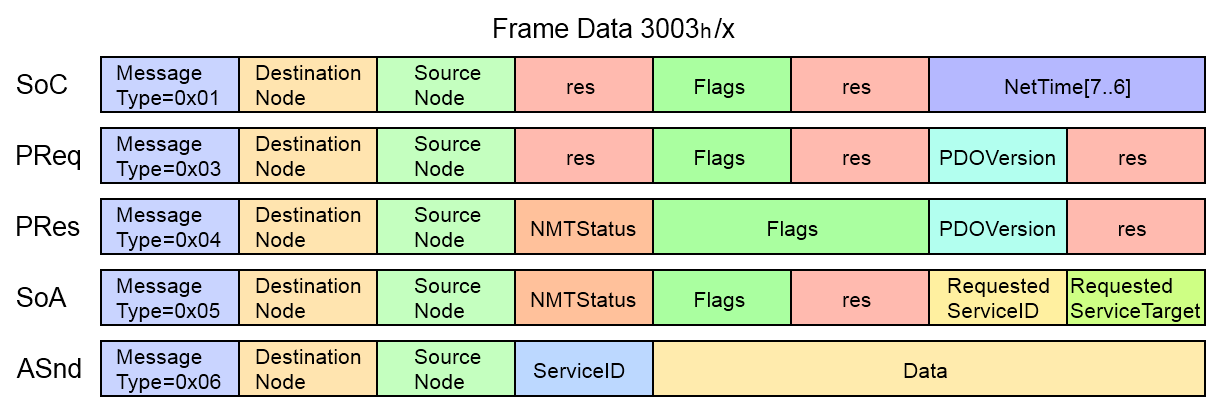


Figure 53: Filter Object Structure of the Different Messages

The mask of object 3004h 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

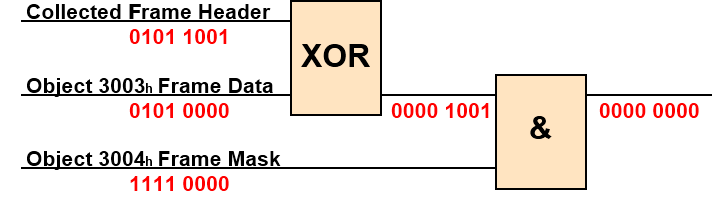


Figure 54: Filter Example: Data Conformance

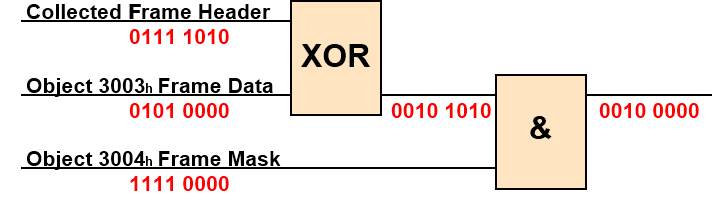


Figure 55: Filter Example: Data Conformance not fulfilled

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 tasks 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:

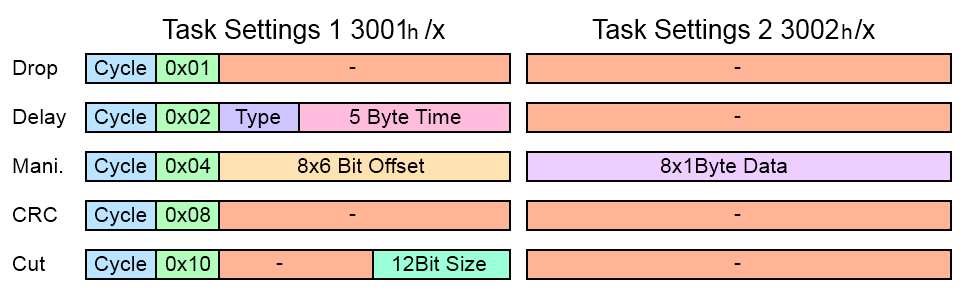


Figure 56: Task Setting Objects Structure

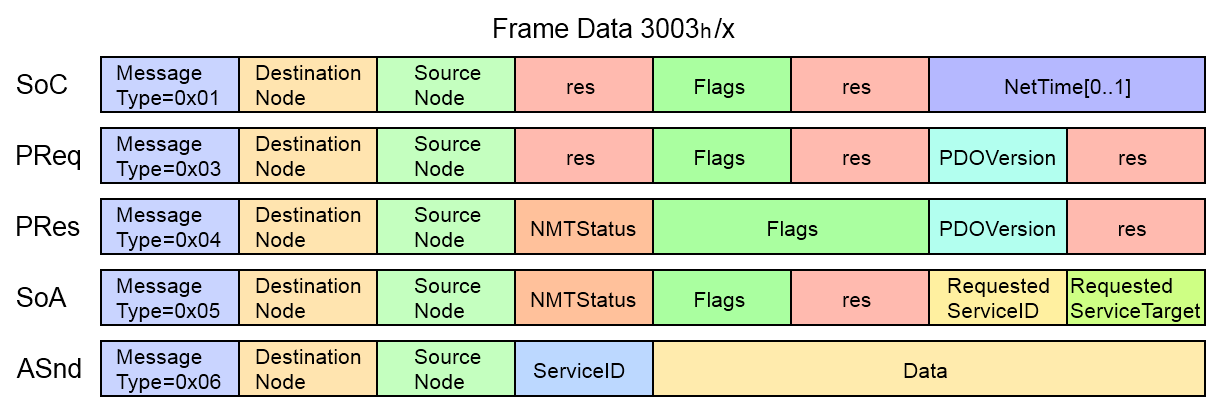


Figure 57: Frame Data Object Structure

### Drop SoC of the First POWERLINK Cycle

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

* **3001h/x Task Setting 1:**
  + Cycle 1 => 0x01 to the first Byte
  + Dropping => 0x01 to the second Byte
* **3002h/x Task Setting 2:**
  + Dropping => empty
* **3003h/x Frame Data:**
  + SoC => 0x01 to the first Byte
* **3004h/x Frame Mask:**
  + Activate Message-Type => 0xFF to the first Byte

|  |  |
| --- | --- |
| Object | Data |
| 3001h/x | 0x0101000000000000 |
| 3002h/x | 0x0000000000000000 |
| 3003h/x | 0x0100000000000000 |
| 3004h/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):

* **3001h/x Task Setting 1:**
  + Cycle 3 => 0x03 to the first Byte
  + Cut => 0x10 to the second Byte
  + Length=>60=> 0x3C to the last Byte
* **3002h/x Task Setting 2:**
  + Cut => empty
* **3003h/x Frame Data:**
  + PRes => 0x04 to the first Byte
  + Source ID =5 => 0x05 to the third Byte
* **3004h/x Frame Mask:**
  + Activate Message-Type => 0xFF to the first Byte
  + Activate Source ID => 0xFF to the third Byte

|  |  |
| --- | --- |
| Object | Data |
| 3001h/x | 0x031000000000003C |
| 3002h/x | 0x0000000000000000 |
| 3003h/x | 0x0400050000000000 |
| 3004h/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.

* **3001h/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
* **3002h/x Task Setting 2:**
  + New Data 1=7 => 0x07 to the first Byte
* **3003h/x Frame Data:**
  + ASnd => 0x06 to the first Byte
  + ServiceID =5 => 0x05 to the fourth Byte
* **3004h/x Frame Mask:**
  + Activate Message-Type => 0xFF to the first Byte
  + Activate ServiceID => 0xFF to the fourth Byte

|  |  |
| --- | --- |
| Object | Data |
| 3001h/x | 0xFF04480000000000 |
| 3002h/x | 0x0700000000000000 |
| 3003h/x | 0x0600000500000000 |
| 3004h/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 5

Figure 2: Ethernet and POWERLINK Header 6

Figure 3: Structure of the POWERLINK Frames 6

Figure 4: Tasks Pass and Delete Frames 7

Figure 5: Tasks Manipulate Frame Header and CRC 7

Figure 6: Tasks Delay and Truncate Frames 7

Figure 7: POWERLINK Slave with Framemanipulator 10

Figure 8: BeMicro RTE Stick [Source: Arroweurope] 11

Figure 9: DE2-115 Development and Education Board [Source: Altera] 11

Figure 10: IP\_core Folder 12

Figure 11: IP Search Path of the SOPC-Builder 12

Figure 12: Adding a new Framemanipulator 13

Figure 13: Framemanipulator SOPC Connection 13

Figure 14: Update SOPC-Block in the Schematic File 14

Figure 15: Connection of the BeMicro Stick 15

Figure 16: RMII Workflow of PL Frames from MN to DUT 15

Figure 17: Rewiring RMII 15

Figure 18: MII Workflow of PL Frames from MN to DUT with Workaround 16

Figure 19: Rewiring the Schematic with rmii2mii Converter - 1 16

Figure 20: Rewiring the schematic with rmii2mii converter - 2 17

Figure 21: Start of Tests after receiving PReq with Start Signal 24

Figure 22: SignalTap Trace; 600µs; Delay of Callback Executing 24

Figure 23: SignalTap Trace; 400µs; Delay of Callback Executing 25

Figure 24: RMII Workflow of PL Frames from MN to DUT 25

Figure 25: SignalTap; Incoming and Outgoing Frame; RMII; Delay of 2.88µs 26

Figure 26: MII Workflow of PL Frames from MN to DUT 26

Figure 27: SignalTap; Incoming and Outgoing Frame; MII; Delay of 6.06µs 26

Figure 28: Automation Studio; Import Fieldbus Device 27

Figure 29: Selection of the CN with the Framemanipulator 27

Figure 30: Automation Studio: I/O Configuration of the Framemanipulator CN 28

Figure 31: FM Operation Register 29

Figure 32: FM Status Register 30

Figure 33: Clear Error Operation 31

Figure 34: Start Task Operation with dropping the fifth SoC 31

Figure 35: Objects of the Manipulation Tasks 32

Figure 36: Example of the used Task Objects 32

Figure 37: Basic Structure of the Task Setting Objects 33

Figure 38: Whole Structure of the Task Setting Objects 33

Figure 39: Task: Dropping Frames 34

Figure 40: Task Setting of Dropping Frames 34

Figure 41: Task: Delaying Frames 34

Figure 42: Task Setting of Delaying Frames 34

Figure 43: Delaying Frame 2; Storing all other Frames until Frame 2 is Passed on 35

Figure 44: Delaying Frame 2; Dropping of all other Frames during the Delay 35

Figure 45: Delaying Frame 2; Storing of the Green Coloured SoC-Frames like Frame 5 35

Figure 46: Task: Manipulating the Headers 36

Figure 47: Task Setting of Manipulating the Headers 36

Figure 48: Task: CRC Distortion 37

Figure 49: Task Setting of CRC Distortion 37

Figure 50: Task: Truncate the Frame 37

Figure 51: Task Setting of Frame Truncation 37

Figure 52: Basic Structure of the Responsible Objects for Filtering the Frames 38

Figure 53: Filter Object Structure of the Different Messages 38

Figure 54: Filter Example: Data Conformance 38

Figure 55: Filter Example: Data Conformance not fulfilled 38

Figure 56: Task Setting Objects Structure 39

Figure 57: Frame Data Object Structure 39

# Table Index

Table 1: Versions 2

Table 2: Distribution 2

Table 3: Safety notices 2

Table 4: Used Commands for the Benchmark Function 24

# Index

D

Distribution 2

F

Figure Index 42

I

Introduction 5

L

Listing Index 45

S

Safety Notices 2

T

Table Index 44

Table of Contents 3

V

Versions 2