XMC4800 EtherCAT

One cycle Client2Client Communication

1. Overview and Requirements
2. Limitations
3. Setup
4. How to test – using TwinCAT3 as host
5. Further readings

1 Overview and Requirements

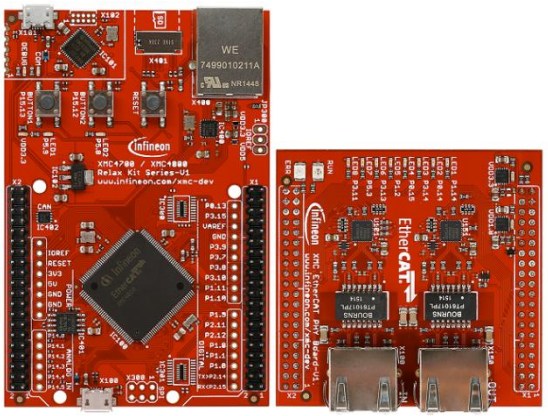
2 Limitations

3 Setup

4 How to test – using TwinCAT3 as host

5 Further readings

Overview

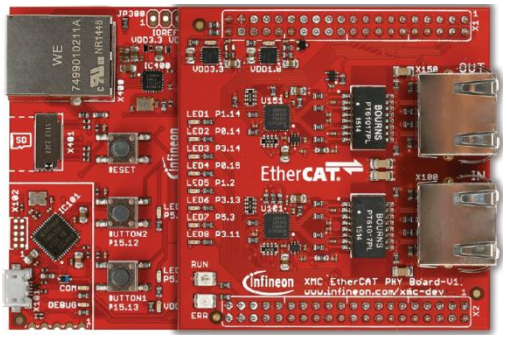


Like with every real time Industrial Ethernet system, one device (the master) has to be in charge of the network management and organise the Medium Access Control. However, with EtherCAT also direct slave-to-slave communication is supported in two ways:

- topology dependent within one communication cycle
- topology independent within two cycles.

Since EtherCAT is faster than competing systems, slave-to-slave communication using two cycles is also faster.
This documentation demonstrates the one cycle slave to slave communication from one EtherCAT™ slave node downstream (topology dependent) to another EtherCAT™ slave node within one bus cycle. This example here is based on the basic example described inside „Getting Started - XMC4800_Relax_EtherCat_APP_Slave_SSC Example_V2.2.pdf“. Before you proceed with this documentation, make sure you have a solid understanding of this documentation first.

Requirements



2 x XMC4800 Relax EtherCAT Kit

2 x RJ45 Ethernet Cable

Windows Laptop installed

- DAVE v4 (Version4.1.4 or higher)
- TwinCAT3 (v3.1.4020.32) Master PLC
- Slave Stack Code Tool

2 x Micro USB Cable (Debugger connector and Power supply)

Requirements - free downloads

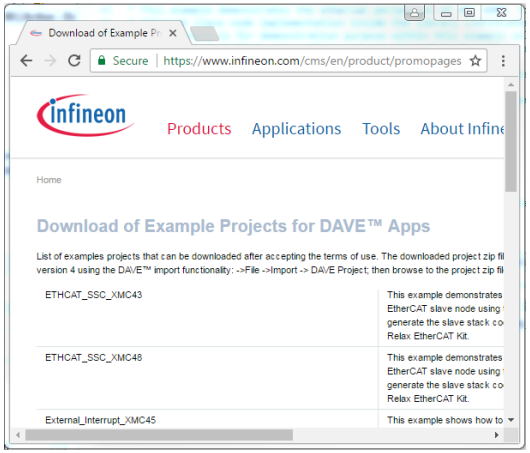
TwinCAT3 (no trial period; usability limited; 32bit and 64bit Windows)
Link: Download TwinCAT3

ATTENTION: According our experience TwinCAT is best compatible with Intel™ ethernet chipset.
For details on compatibility with your hardware, additional driver and general installation support please get into contact with your local BECKHOFF support.

Requirements - free downloads

DAVE (v4.1.4 or higher)
Link: Download DAVE (Version 4)

EtherCAT Slave Stack Code Tool
(ETG membership obligatory)
Link: Slave Stack Code Tool



ETHCAT_SSC_XMC48
Infineons basic EtherCAT example
Link: Download Examples

1 Overview and Requirements

2 Limitations

3 Setup

4 How to test – using TwinCAT3 as host

5 Further readings

Limitations

- One cycle slave to slave communication is basically a matter of network configuration by the master.
- Depending on the master used, this configuration might be limited and special slave node implementations needed.
- The demonstration described here, was tested using TwinCAT3 v3.1.4020.32 and limitations result from this.
- Limitations on the EtherCAT™ slave nodes for this demonstration:

Sending slave:
1. Only input PDO data allowed
2. Input PDO data structure must match output PDO data structure of receiving slave

Receiving slave:
1. Only output PDO data allowed
2. Output PDO data structure must match input PDO data structure of sending slave

1 Overview and Requirements

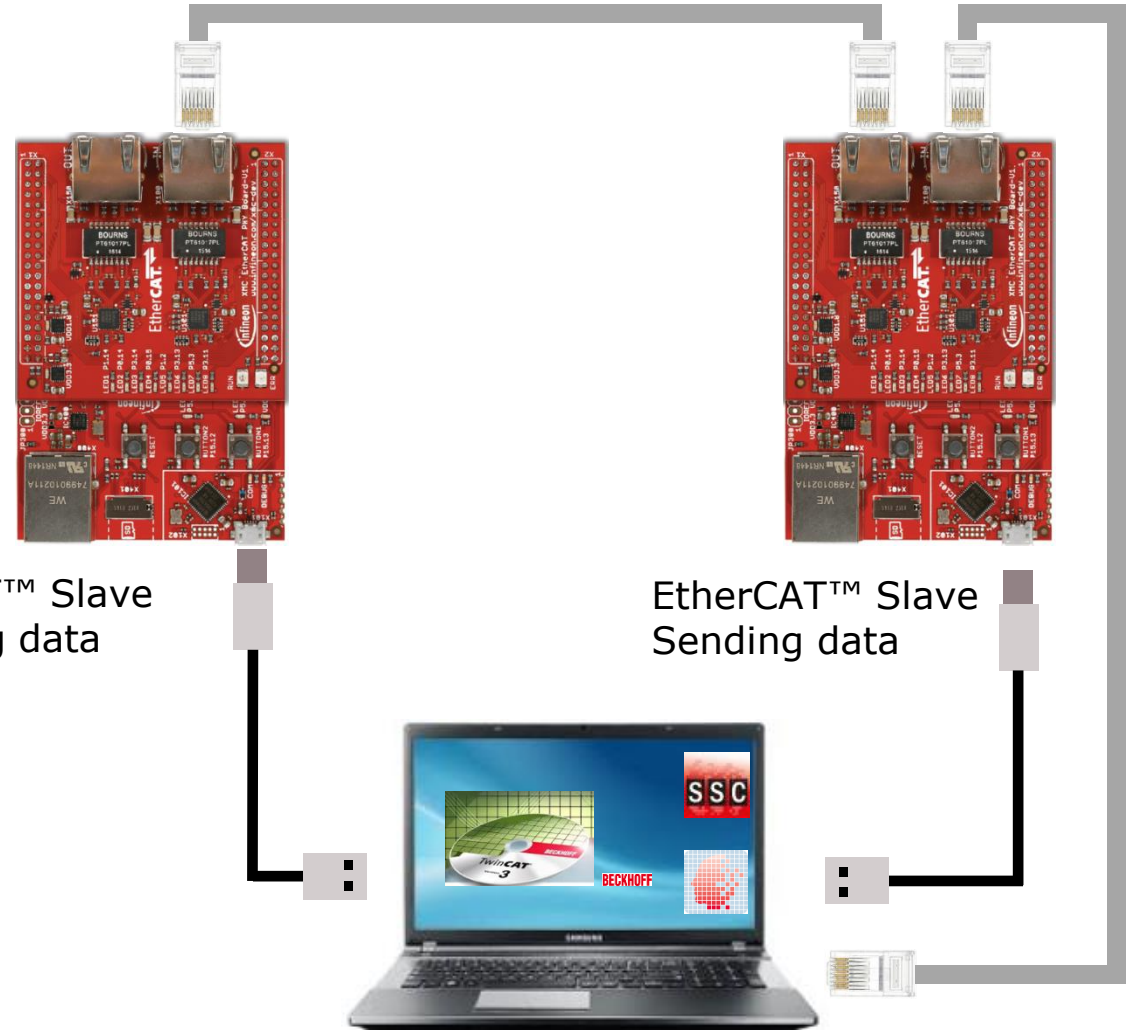
2 Limitations

3 Setup

4 How to test – using TwinCAT3 as host

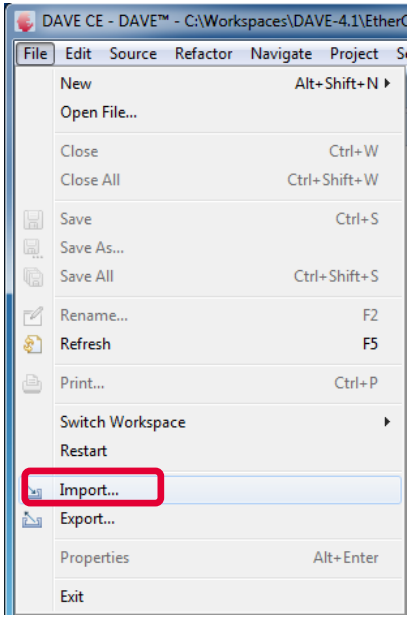
5 Further readings

Setup – Hardware

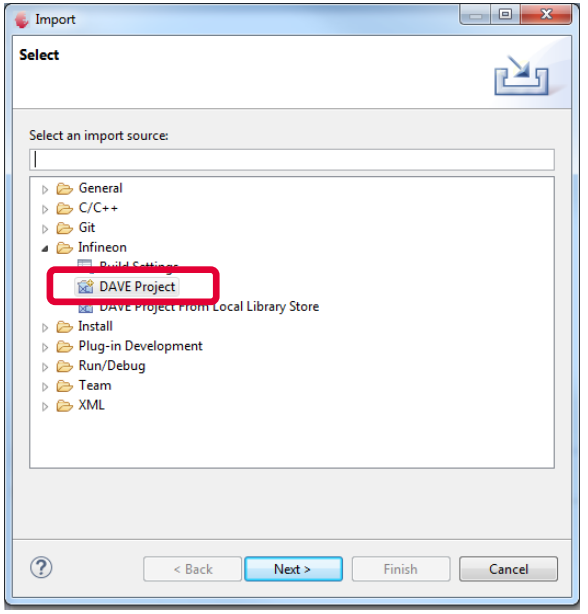


Setup – Import Basic Example (ETHCAT_SSC_XMC48)

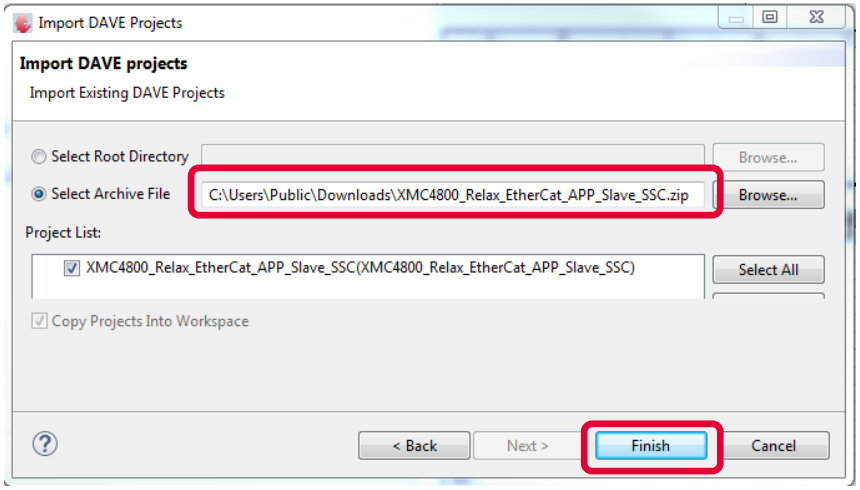
1



2



3

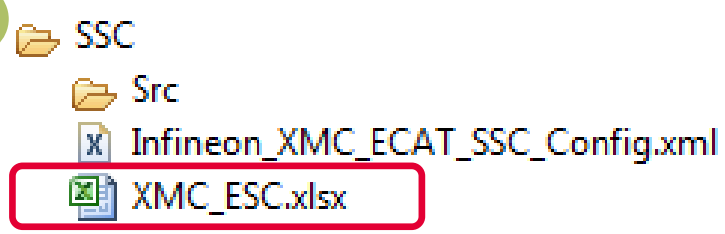


Setup – Modify Basic Example

1. For this demonstration you need two different EtherCAT™ slave node implementations. One for the receiving and one for the sending slave.
2. First follow the documentation which comes with the basic example (ETHCAT_SSC_XMC48) and prepare it for compilation.
3. Duplicate the project.
4. The following documentation will describe the neccessary modifications of the basic example. One for the receiving and one for the sending slave.

Setup – Modify Basic Example Sending Slave (XMC_ESC_IN)

1



2

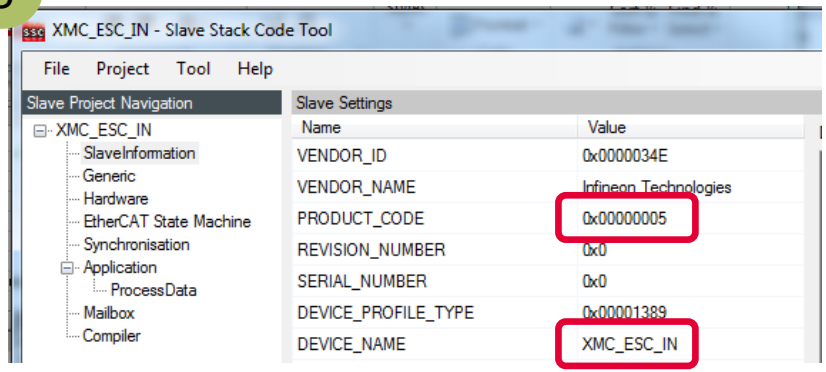
Index	ObjectCode	SI	DataType	Name
//0x6nnx	Input Data of the Module (0x6000 - 0x6FFF)			
0x6000	RECORD			IN_GENERIC
		0x01	UINT	IN_GEN_INT1
		0x02	UINT	IN_GEN_INT2
		0x03	UINT	IN_GEN_INT3
		0x04	UINT	IN_GEN_INT4
		0x05	BOOL	IN_GEN_Bit1
		0x06	BOOL	IN_GEN_Bit2
		0x07	BOOL	IN_GEN_Bit3
		0x08	BOOL	IN_GEN_Bit4
		0x09	BOOL	IN_GEN_Bit5
		0x0A	BOOL	IN_GEN_Bit6
		0x0B	BOOL	IN_GEN_Bit7
		0x0C	BOOL	IN_GEN_Bit8
//0x7nnx	Output Data of the Module (0x7000 - 0x7FFF)			

1 Double click on the excel file to open it.

2 Empty the data defined for the output direction and save the file.

Setup – Modify Basic Example
Sending Slave (XMC_ESC_IN)

3



3 Create a new SSC project and specify a unique product code and device name for your slave node.

4 Regenerate the SSC code using the XMC_ESC.xlsx file you just have modified before.

Setup – Modify Basic Example
Sending Slave (XMC_ESC_IN)

5

XMC_ESC.c

```
254 ////////////////////////////////////////////////////////////////////////////////
255 /**
256 \param      pData  pointer to input process data
257
258 \brief      This function will copies the inputs from the local memory to the ESC
259           to the hardware
260 *////////////////////////////////////////////////////////////////////////////////
261 void APPL_InputMapping(UINT16* pData)
262 {
263     memcpy(pData,&(((UINT16*)&IN_GENERIC0x6000)[1]),SIZEOF(IN_GENERIC0x6000)-2);
264 }
265
266 ////////////////////////////////////////////////////////////////////////////////
267 /**
268 \param      pData  pointer to output process data
269
270 \brief    This function will copies the outputs from the ESC memory to the local
271           to the hardware
272 *////////////////////////////////////////////////////////////////////////////////
273 void APPL_OutputMapping(UINT16* pData)
274 {
275     return;
276 }
277
278 ////////////////////////////////////////////////////////////////////////////////
279 /**
280 \brief    This function will called from the synchronisation ISR
281           or from the mainloop if no synchronisation is supported
282 ////////////////////////////////////////////////////////////////////////////////
283 void process_app(TOBJ6000 *IN_GENERIC);
284 void APPL_Application(void)
285 {
286     process_app(&IN_GENERIC0x6000);
287 }
```

5 Open the file XMC_ESC.c of the generated code for customization.

Implement APPL_InputMapping, APPL_OutputMapping and APPL_Application.

Setup – Modify Basic Example
Sending Slave (XMC_ESC_IN)

6

main.c

```
108 void process_app(TOBJ6000 *IN_GENERIC)
109 {
110     static uint8_t slowdown=0;
111     static uint8_t count=0;
112
113     slowdown++;
114
115     if (slowdown==100)
116     {
117         count++;
118         slowdown=0;
119     }
120     IN_GENERIC->IN_GEN_Bit1=count&0x1;
121     IN_GENERIC->IN_GEN_Bit2=(count>>1)&0x1;
122     IN_GENERIC->IN_GEN_Bit3=(count>>2)&0x1;
123     IN_GENERIC->IN_GEN_Bit4=(count>>3)&0x1;
124     IN_GENERIC->IN_GEN_Bit5=(count>>4)&0x1;
125     IN_GENERIC->IN_GEN_Bit6=(count>>5)&0x1;
126     IN_GENERIC->IN_GEN_Bit7=(count>>6)&0x1;
127     IN_GENERIC->IN_GEN_Bit8=(count>>7)&0x1;
128
129 }
130
```

6 Open the main.c file. Inside the function process_app create some input data for sending it to the receiving node.

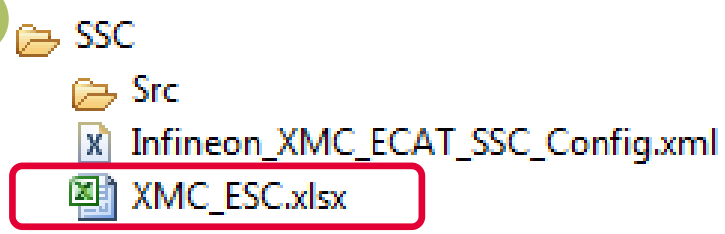
Here a 8-bit counter is mapped to the eight binary inputs.

7 Erase the flash of the sending EtherCAT™ node (XMC4800 Relax Kit)

8 Build and download the software to the sending EtherCAT™ node.

Setup – Modify Basic Example
Receiving Slave (XMC_ESC_OUT)

1



2

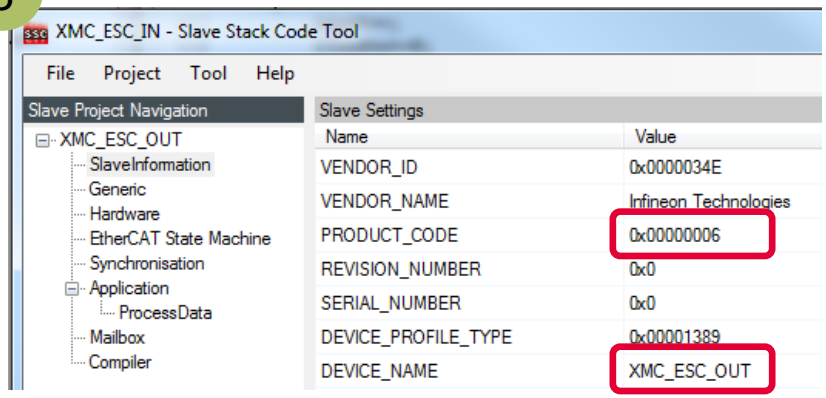
dex	ObjectCode	SI	DataType	Name
//0x6nnx	Input Data of the Module (0x6000 - 0x6FFF)			
//0x7nnx	Output Data of the Module (0x7000 - 0x7FFF)			
0x7000	RECORD			OUT_GENERIC
		0x01	UINT	OUT_GEN_INT1
		0x02	UINT	OUT_GEN_INT2
		0x03	UINT	OUT_GEN_INT3
		0x04	UINT	OUT_GEN_INT4
		0x05	BOOL	OUT_GEN_Bit1
		0x06	BOOL	OUT_GEN_Bit2
		0x07	BOOL	OUT_GEN_Bit3
		0x08	BOOL	OUT_GEN_Bit4
		0x09	BOOL	OUT_GEN_Bit5
		0x0A	BOOL	OUT_GEN_Bit6
		0x0B	BOOL	OUT_GEN_Bit7
		0x0C	BOOL	OUT_GEN_Bit8

1 Double click on the excel file to open it.

2 Empty the data defined for the input direction and save the file.

Setup – Modify Basic Example
Receiving Slave (XMC_ESC_OUT)

3



3 Create a new SSC project and specify a unique product code and device name for your slave node.

4 Regenerate the SSC code using the XMC_ESC.xlsx file you just have modified before.

Setup – Modify Basic Example
Receiving Slave (XMC_ESC_OUT)

5

XMC_ESC.c

```
254 ////////////////////////////////////////////////////////////////////////////////
255 /**
256 \param      pData  pointer to input process data
257
258 \brief      This function will copies the inputs from the local memory to the ESC me
259             to the hardware
260 *////////////////////////////////////////////////////////////////////////////////
261 void APPL_InputMapping(UINT16* pData)
262 {
263 return;
264 }
265
266 ////////////////////////////////////////////////////////////////////////////////
267 /**
268 \param      pData  pointer to output process data
269
270 \brief    This function will copies the outputs from the ESC memory to the local mer
271             to the hardware
272 *////////////////////////////////////////////////////////////////////////////////
273 void APPL_OutputMapping(UINT16* pData)
274 {
275     memcpy(&(((UINT16*)&OUT_GENERIC0x7000)[1]),pData,SIZEOF(OUT_GENERIC0x7000)-2);
276 }
277
278 ////////////////////////////////////////////////////////////////////////////////
279 /**
280 \brief    This function will called from the synchronisation ISR
281             or from the mainloop if no synchronisation is supported
282 *////////////////////////////////////////////////////////////////////////////////
283 void process_app(TOBJ7000 *OUT_GENERIC);
284 void APPL_Application(void)
285 {
286     process_app(&OUT_GENERIC0x7000);
287 }
```

5 Open the file XMC_ESC.c of the generated code for customization.

Implement APPL_InputMapping, APPL_OutputMapping and APPL_Application.

Setup – Modify Basic Example
Receiving Slave (XMC_ESC_OUT)

6

main.c

```
108 void process_app(TOBJ7000 *OUT_GENERIC)
109 {
110   /* OUTPUT PROCESSING */
111   /* Check bitfield set by master OUT_GEN_Bit1..8 and set LEDs accordingly */
112   XMC_GPIO_SetOutputLevel(P_LED1, MAP2LEVEL(OUT_GENERIC->OUT_GEN_Bit1));
113   XMC_GPIO_SetOutputLevel(P_LED2, MAP2LEVEL(OUT_GENERIC->OUT_GEN_Bit2));
114   XMC_GPIO_SetOutputLevel(P_LED3, MAP2LEVEL(OUT_GENERIC->OUT_GEN_Bit3));
115   XMC_GPIO_SetOutputLevel(P_LED4, MAP2LEVEL(OUT_GENERIC->OUT_GEN_Bit4));
116   XMC_GPIO_SetOutputLevel(P_LED5, MAP2LEVEL(OUT_GENERIC->OUT_GEN_Bit5));
117   XMC_GPIO_SetOutputLevel(P_LED6, MAP2LEVEL(OUT_GENERIC->OUT_GEN_Bit6));
118   XMC_GPIO_SetOutputLevel(P_LED7, MAP2LEVEL(OUT_GENERIC->OUT_GEN_Bit7));
119   XMC_GPIO_SetOutputLevel(P_LED8, MAP2LEVEL(OUT_GENERIC->OUT_GEN_Bit8));
120
121 }
```

6 Open the main.c file. Inside the function process_app map the eight binary outputs this EtherCAT node receives to the available LEDs on the XMC4800 Relax Kit.

7 Erase the flash of the receiving EtherCAT™ node (XMC4800 Relax Kit)

8 Build and download the software to the sending EtherCAT™ node (XMC4800 Relax Kit).

1 Overview and Requirements

2 Limitations

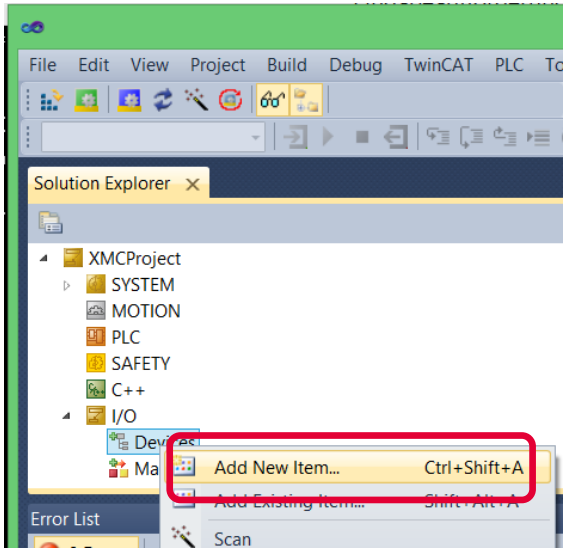
3 Setup

4 How to test – using TwinCAT3 as host

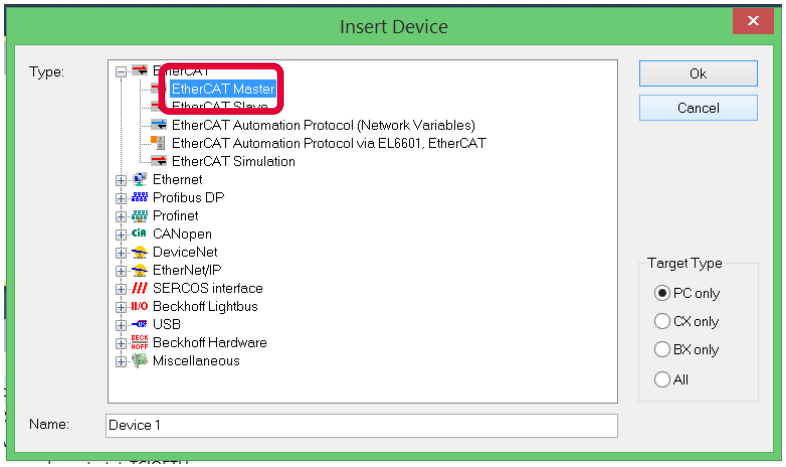
5 Further readings

How to test – start the TwinCAT 3 master and scan the bus

1



2



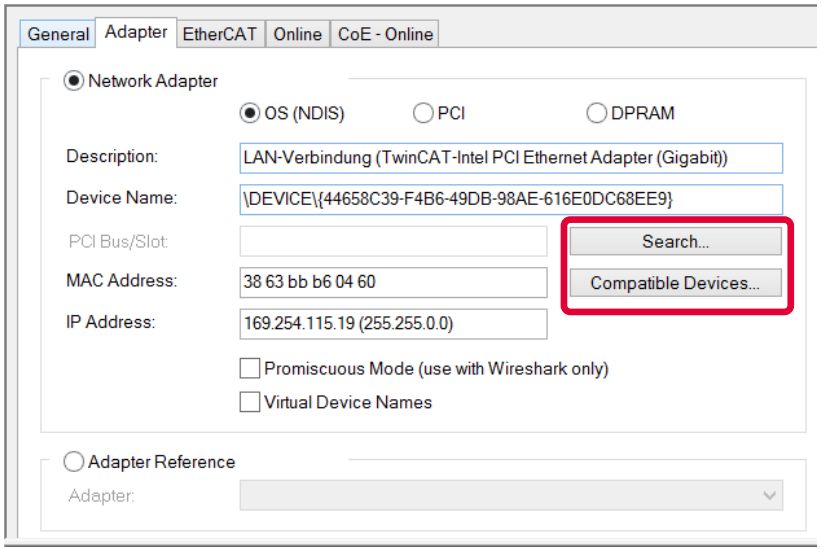
ACTIONS

After starting the TwinCAT System Manager from windows start menu:

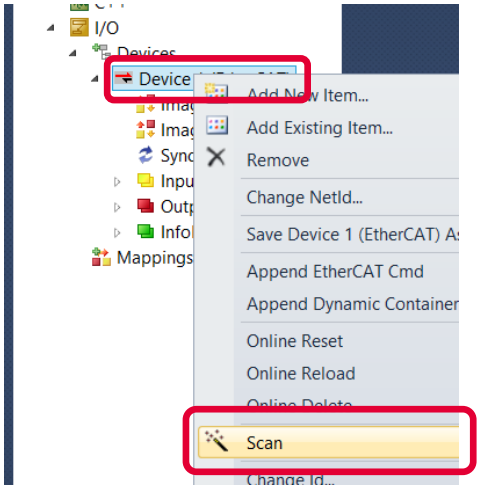
1. Right Click I/O-Devices and select „Add New Item…“
2. Create an EtherCAT master device by double click

How to test – start the TwinCAT 3 master and scan the bus

3



4



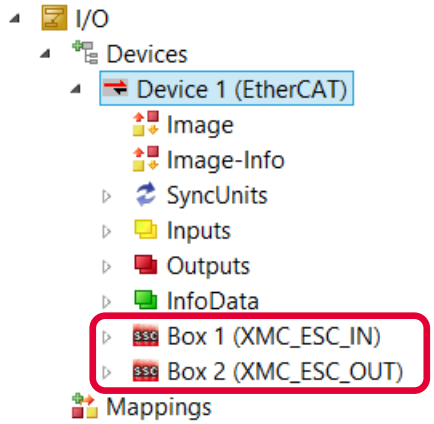
ACTIONS

3 Select the network adapter you want to use (search and select).
Application hint:
In case the device is not found please install the respective device driver by following the instructions given by TwinCAT through the „Compatible Devices…“ button.

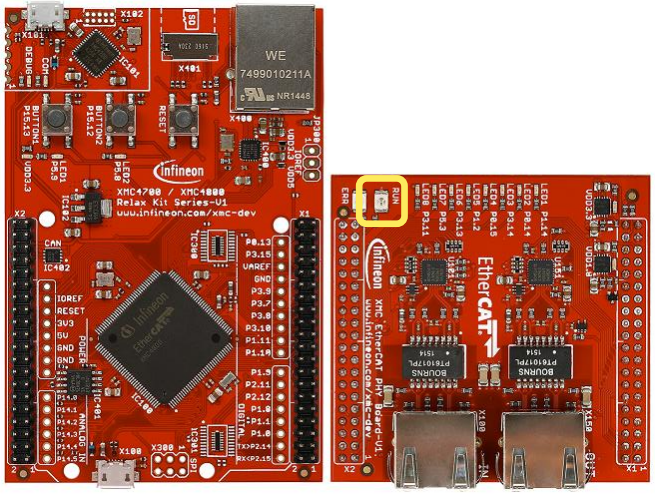
4 Right Click EtherCAT master and select „Scan Boxes…“

How to test – start the TwinCAT 3 master and scan the bus

1



2



OBSERVATIONS

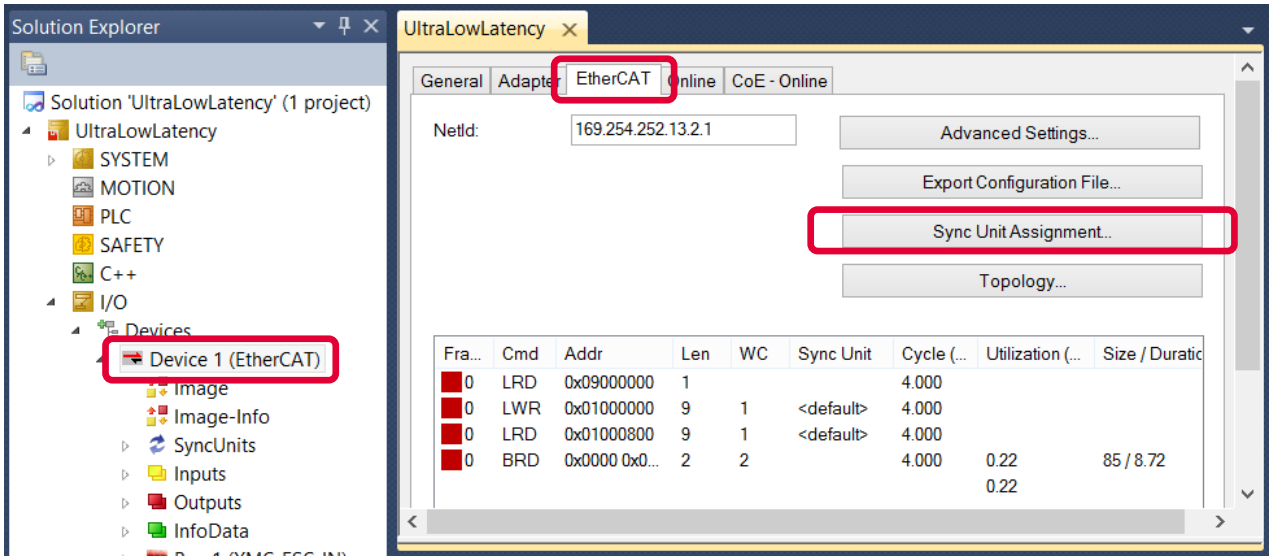
1 Both slaves appear as a node on the EtherCAT master bus. Take a note that the topology is mandatory. So Box1 must be the sending slave (XMC_ESC_IN) and Box 2 must be the receiving slave (XMC_ESC_OUT).

2 The RUN-LEDs of both slave nodes are flashing indicating PREOP-state

How to test – Link sync managers and PDO data of sending and receiving node

ACTION

1

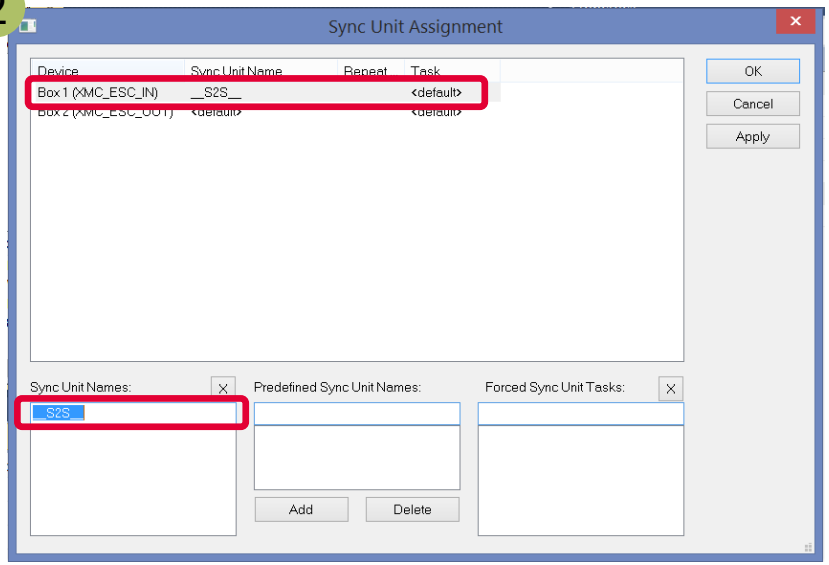


1 Open window to interconnect the sync managers of the receiving and sending slave

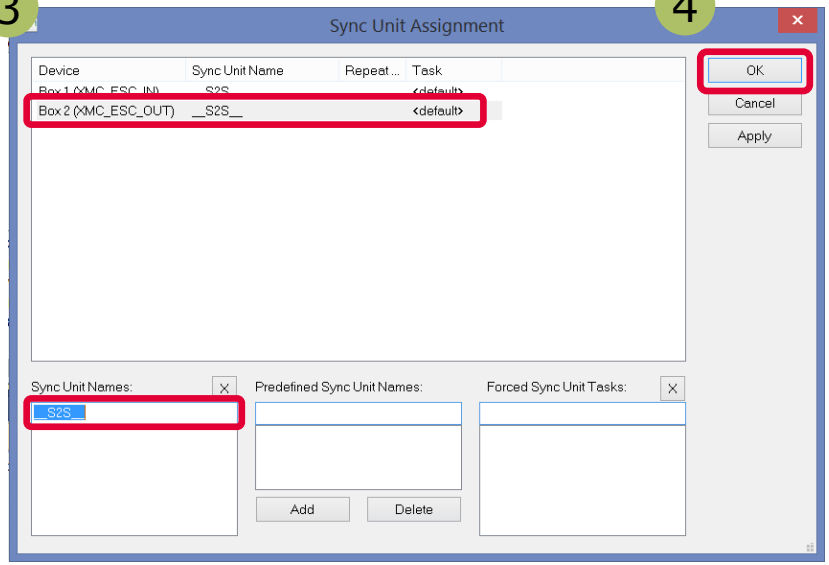
How to test – Link sync managers and PDO data of sending and receiving node

ACTION

2



3 4



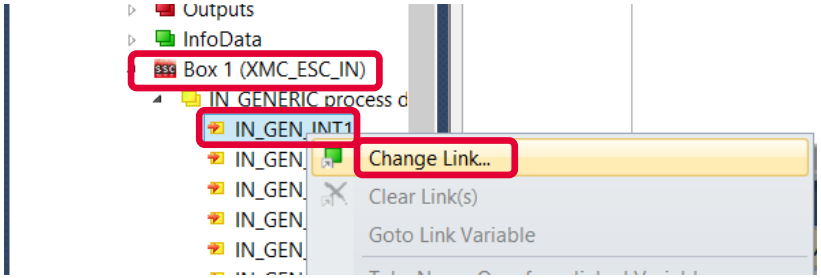
2 Assign the Sync Unit Name „__S2S__“ to the sending slave

3 Assign the Sync Unit Name „__S2S__“ to the receiving slave

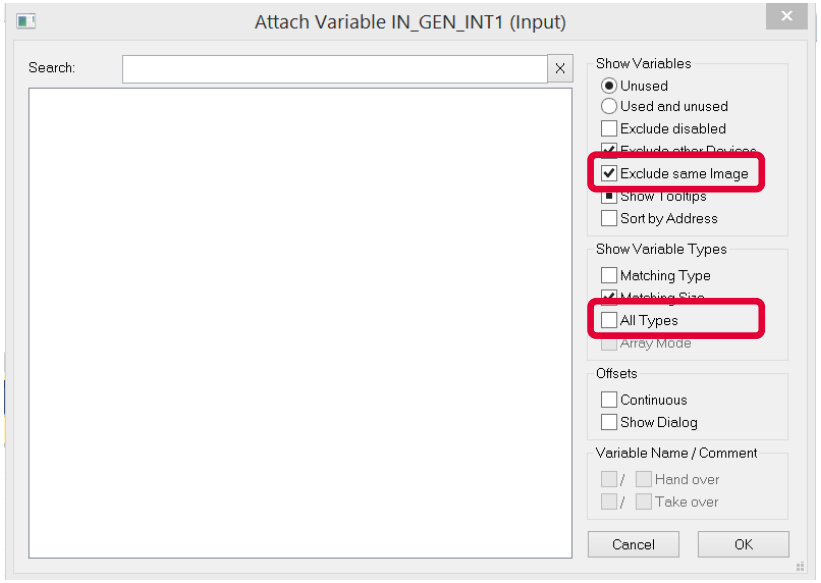
4 Confirm with OK button

How to test – Link sync managers and PDO data of sending and receiving node

5



6



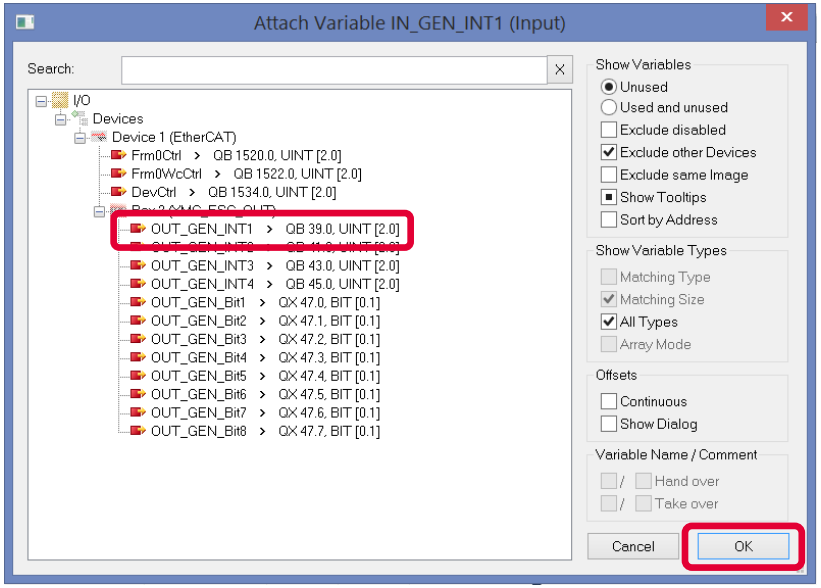
ACTIONS

5 Right Click the first PDO data item „IN_GEN_INT1“ of the sending slave and select „Change Link…“

6 Unselect „Exclude same Image“ and select „All Types“ to visualize the PDO data items of the receiving slave.

How to test – Link sync managers and PDO data of sending and receiving node

7



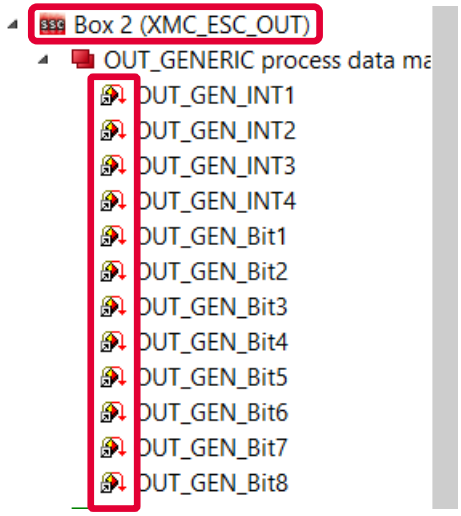
ACTIONS

7 Select the first data item „IN_GEN_INT1“ of the receiving slave and confirm with OK

8 Proceed with step 5 and 7 for all PDO data items of the sending slave. By this all PDO data of the sending slave is linked to the receiving slave.

How to test – Link sync managers and PDO data of sending and receiving node

1



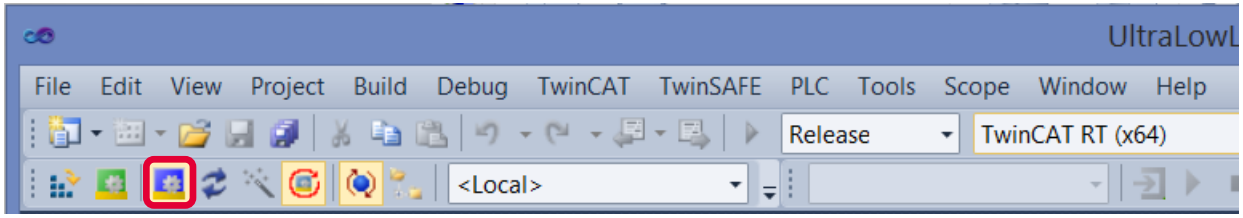
OBSERVATIONS

1 The icons of the PDO data items of the receiving slave now indicate the linkage to the sending slave.

How to test – Monitor the PDO data sent from the first node to the second node

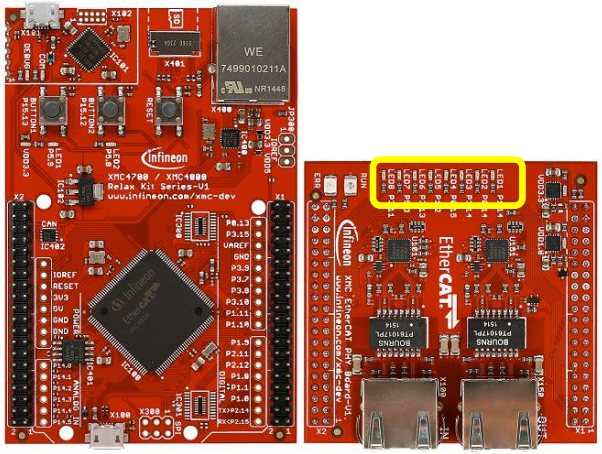
ACTIONS

Activate the configuration and confirm all further pop-ups until the network runs in free run mode.



OBSERVATION

The 8-bit counter implemented inside sending node is represented on the eight LEDs of the receiving node.



1 Overview and Requirements

2 Limitations

3 Setup

4 How to test – using TwinCAT3 as host

5 Further readings

Further readings

Some background on the EtherCAT™ protocol you can find here:

Moving up to Industrial Ethernet: The EtherCAT protocol

Part of your life. Part of tomorrow.

Infineon