

XMC4800

EtherCAT APP SSC

Slave Example

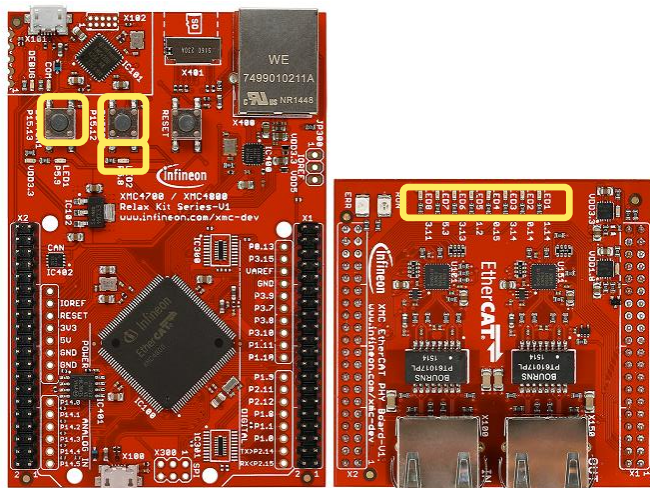
Getting Started V2.2



- 1 Overview and Requirements
- 2 Setup
- 3 Defining the interface of EtherCAT slave node
- 4 Generating Slave Stack Code and ESI file
- 5 Implementation of the application
- 6 How to test – using TwinCAT2 as host
- 7 How to test – using TwinCAT3 as host

- 1 Overview and Requirements
- 2 Setup
- 3 Defining the interface of EtherCAT slave node
- 4 Generating Slave Stack Code and ESI file
- 5 Implementation of the application
- 6 How to test – using TwinCAT2 as host
- 7 How to test – using TwinCAT3 as host

Overview



This example demonstrates the implementation of an EtherCAT slave node using the Beckhoff SSC Tool to generate the slave stack code for „XMC4800 Relax EtherCAT Kit“.

While reviewing this example you will see in output direction the EtherCAT master controlling the 8 LEDs on the „XMC EtherCAT PHY Board“ and dimming LED2 of the Relax Kit. In input direction you will monitor inside the master device the status of the buttons available on the Relax Kit. You will observe inside the source code how to modify the mapping of the data structures to the I/Os for your own evaluations and testing. Furthermore you will learn how to modify the data structures and generate a slave stack code which fits to your needs. In this example we will demonstrate how easy it is to setup a proper EtherCAT communication by using the EtherCAT APP.

Requirements



XMC4800 Relax EtherCAT Kit



RJ45 Ethernet Cable



Windows Laptop installed

- DAVE v4 (Version4.1.4 or higher)
- TwinCAT2 or TwinCAT3 Master PLC
- Slave Stack Code Tool



Micro USB Cable (Debugger connector)

Requirements - free downloads



TwinCAT2 (30 day trial; 32bit Windows only)

Link: [Download TwinCAT2](#)

or



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 \(Version 4\)](#)



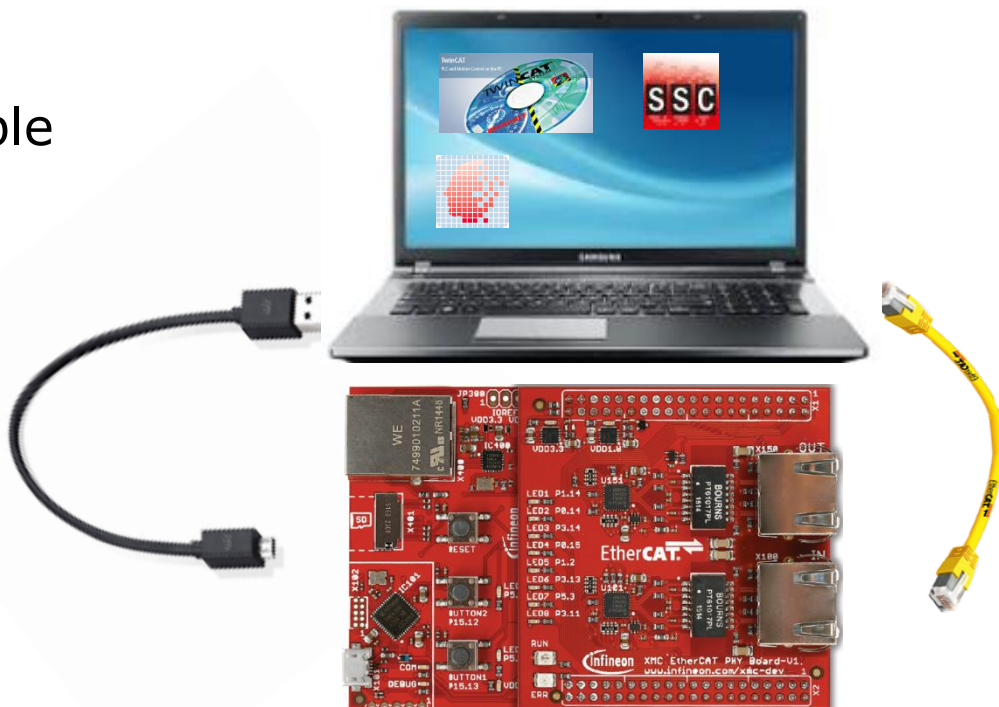
EtherCAT Slave Stack Code Tool
(ETG membership obligatory)

Link: [Slave Stack Code Tool](#)

- 1 Overview and Requirements
- 2 **Setup**
- 3 Defining the interface of EtherCAT slave node
- 4 Generating Slave Stack Code and ESI file
- 5 Implementation of the application
- 6 How to test – using TwinCAT2 as host
- 7 How to test – using TwinCAT3 as host

Setup – Hardware

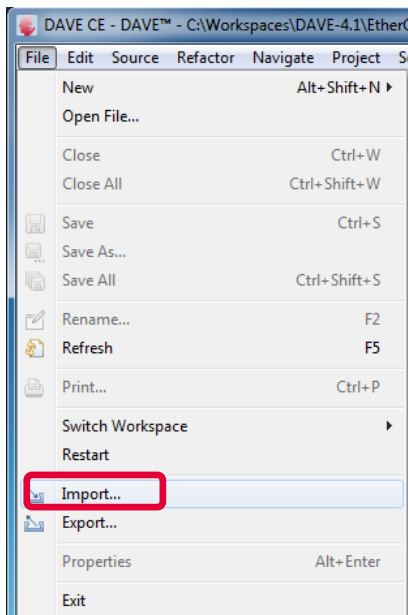
Micro USB cable
Debugger
connected to
X101 debug
connector



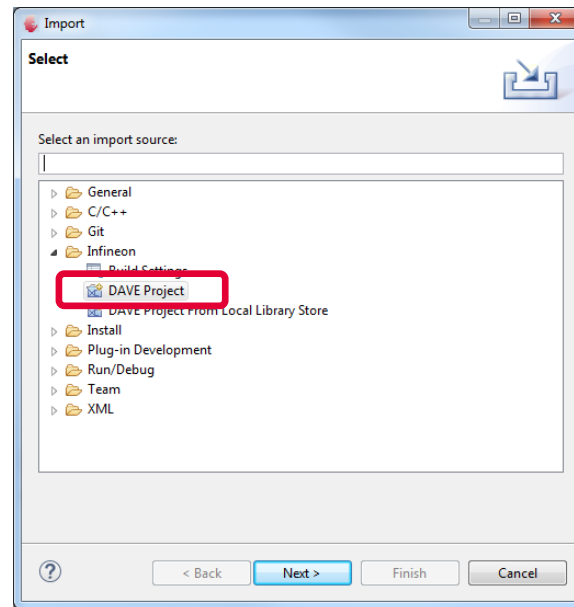
Ethernet Cable
connected to
IN-port

Setup – Import example project into DAVE

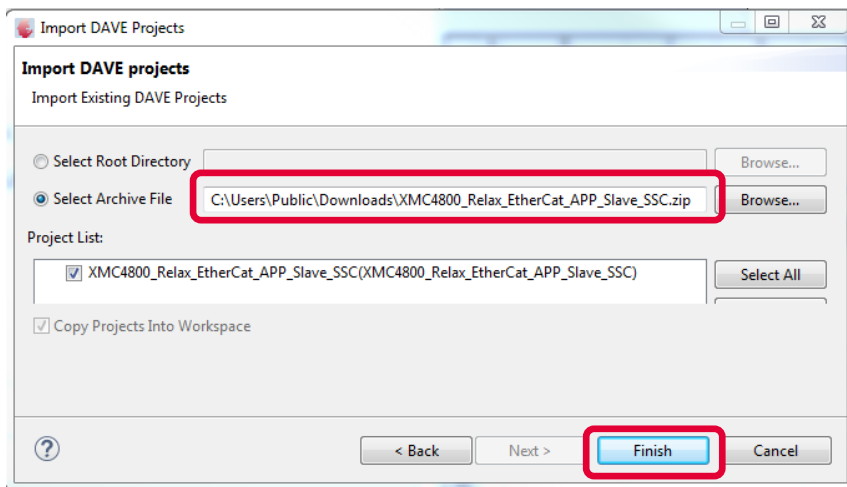
1



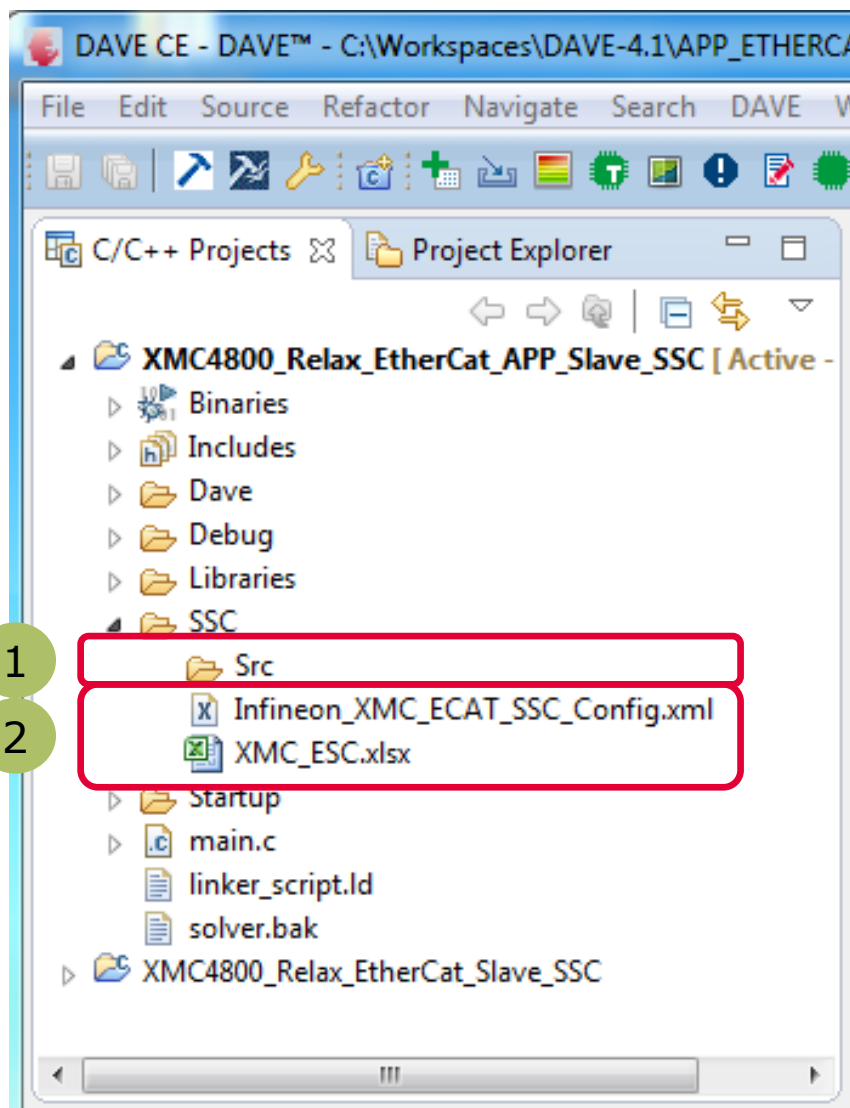
2



3



Setup – Import example project into DAVE



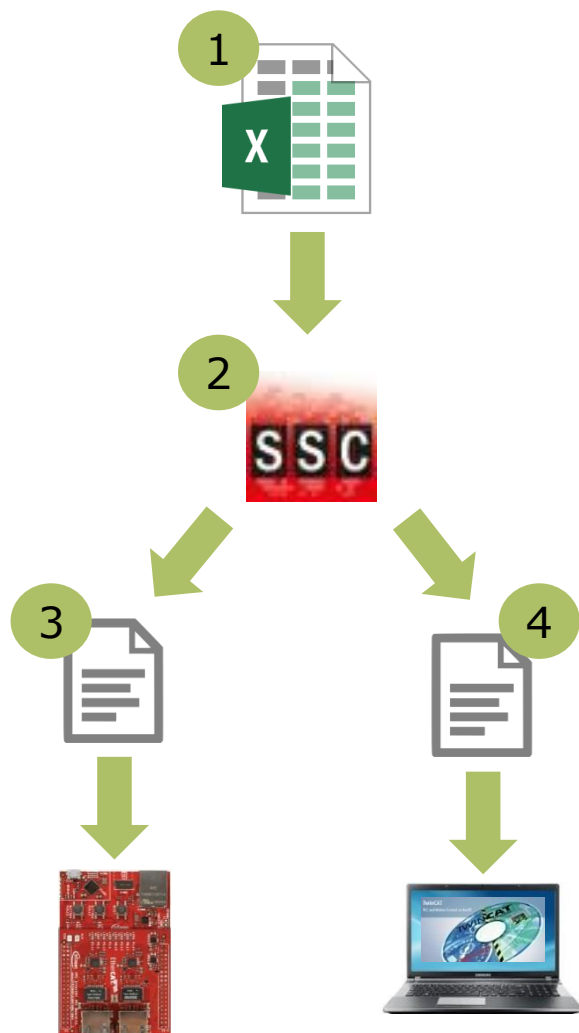
After the project import you will find this project folder structure.

- 1 The project is nearly complete for building, it only misses the EtherCAT slave stack code. For these files the Src folder has been already prepared.
- 2 The EtherCAT slave stack code for the XMC4800 can be generated by configuration files. These configuration files are included in the project already.

The following slides show in detail how to define your EtherCAT slave node interface and to generate the slave stack code.

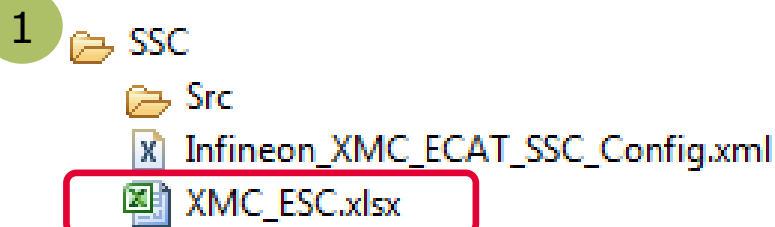
- 1 Overview and Requirements
- 2 Setup
- 3 Defining the interface of EtherCAT slave node
- 4 Generating Slave Stack Code and ESI file
- 5 Implementation of the application
- 6 How to test – using TwinCAT2 as host
- 7 How to test – using TwinCAT3 as host

The flow to define the EtherCAT slave node interface



- 1 Take the Excel Worksheet provided inside the example project to define your EtherCAT slave node interface.
- 2 The Beckhoff SSC-tool uses the excel sheet as an input to generate the output-files.
- 3 The generated EtherCAT slave stack code does apply for the XMC4800.
- 4 The generated **E**therCAT **S**lave **I**nformation file (ESI) does apply for the EtherCAT host. There the relevant interface information about the slave is stored.

Defining the interface of EtherCAT slave node



2

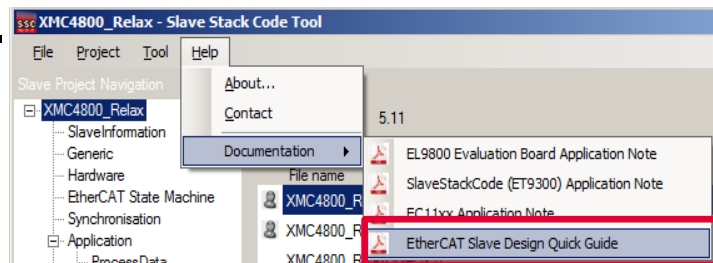
Index	ObjectCode	SI	DataType	Name
Input Data of the Module (0x6000 - 0x6FFF)				
W0x6nnx 0x6000	RECORD			IN_GENERIC
		0x01	UINT	IN_GEN_INT1
		0x02	UINT	IN_GEN_INT2
		0x03	UINT	IN_GEN_INT3
		0x04	UINT	IN_GEN_INT4
		0x05	BOOL	IN_GEN_Bit1
		0x06	BOOL	IN_GEN_Bit2
		0x07	BOOL	IN_GEN_Bit3
		0x08	BOOL	IN_GEN_Bit4
		0x09	BOOL	IN_GEN_Bit5
		0x0A	BOOL	IN_GEN_Bit6
		0x0B	BOOL	IN_GEN_Bit7
		0x0C	BOOL	IN_GEN_Bit8

Index	ObjectCode	SI	DataType	Name
Output Data of the Module (0x7000 - 0x7FFF)				
W0x7nnx 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 Check the content of the file. The data defined in both I/O directions is 4x16bit integers and 8x1bit booleans.

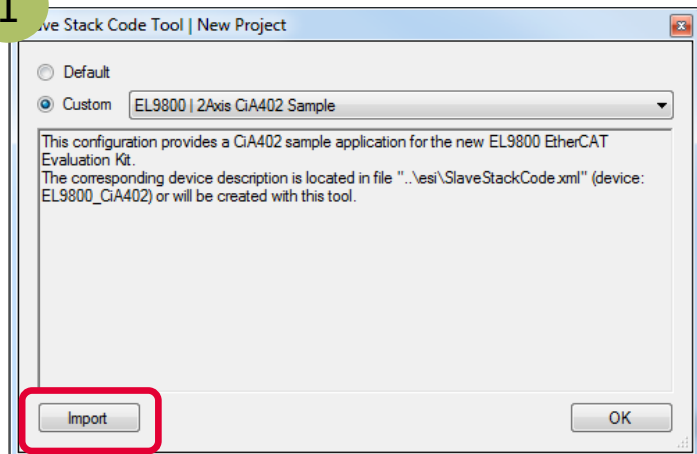
3 For further details on how to define your own interface you may want to follow the instructions inside *EtherCAT Slave Design Quick Guide.pdf* inside SSC tool.



- 1 Overview and Requirements
- 2 Setup
- 3 Defining the interface of EtherCAT slave node
- 4 **Generating Slave Stack Code and ESI file**
- 5 Implementation of the application
- 6 How to test – using TwinCAT2 as host
- 7 How to test – using TwinCAT3 as host

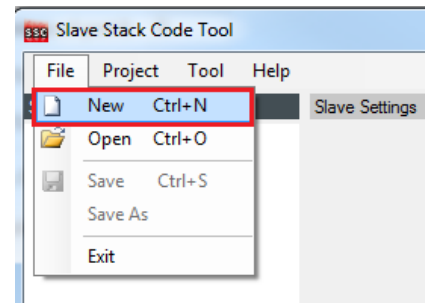
Generating Slave Stack Code and ESI file

1

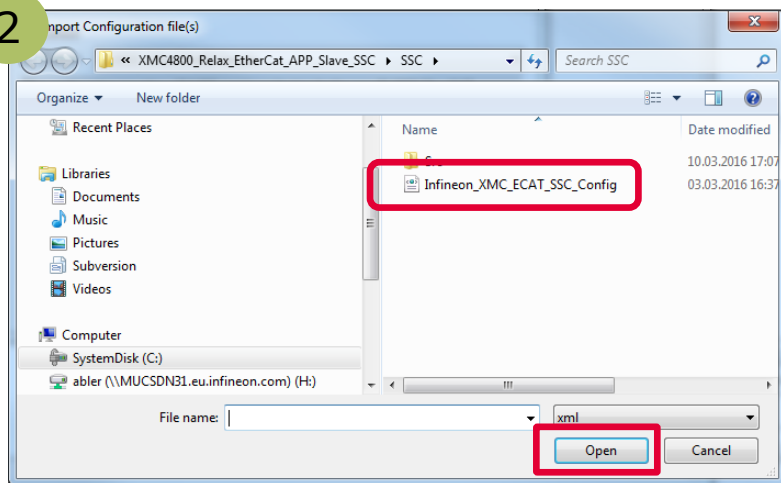


1

Start the **SSC** tool and create a new project **File >> New**



2

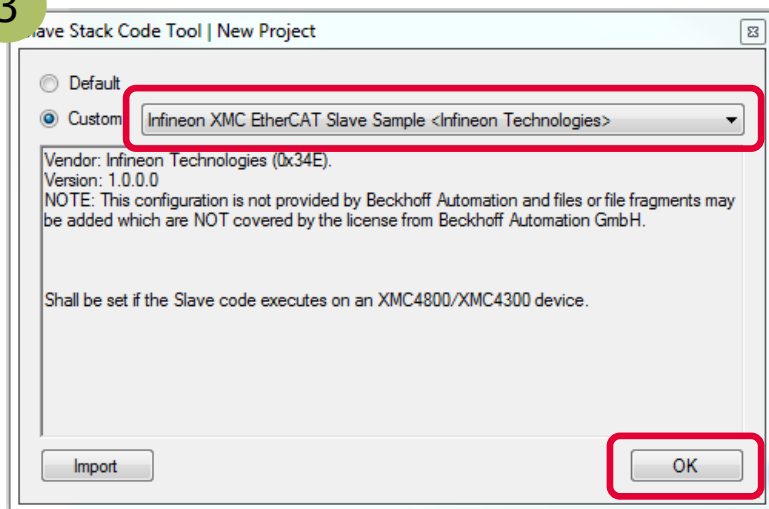


2

Select the configuration file which you find inside the example project.

Generating Slave Stack Code and ESI file

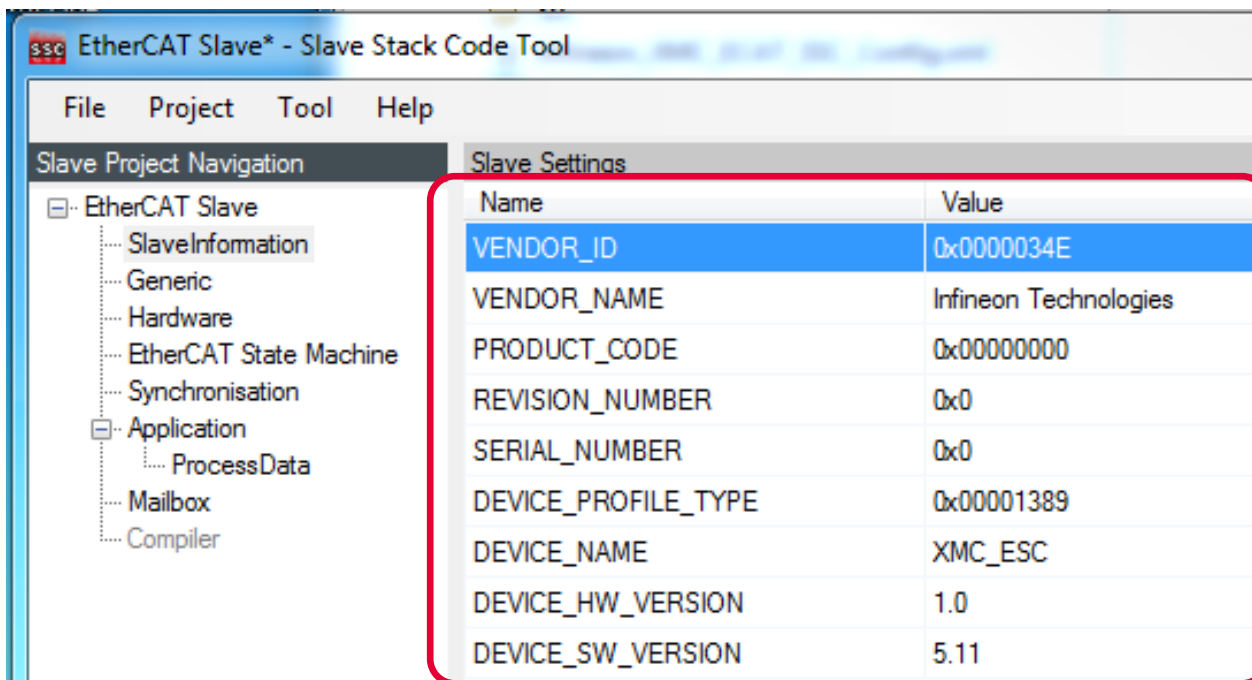
3



3

Select the Infineon device inside the drop down list and confirm with the OK button. Your project will be created.

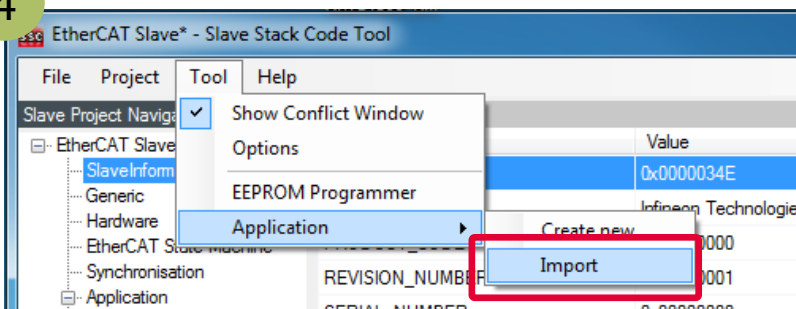
Generating Slave Stack Code and ESI file



- › Check the settings inside SlaveInformation: vendor ID, vendor name, product ID and product code are customer specific and are used by the host to identify the slave.
- › Define revision number, serial number, device name, HW/SW version according to your needs.
- › The vendor ID/name and product code assigned to infineon may be used for evaluation purpose only. For productive purpose your own vendor ID/name assigned by the EtherCAT Technology Group is obligatory.

Generating Slave Stack Code and ESI file

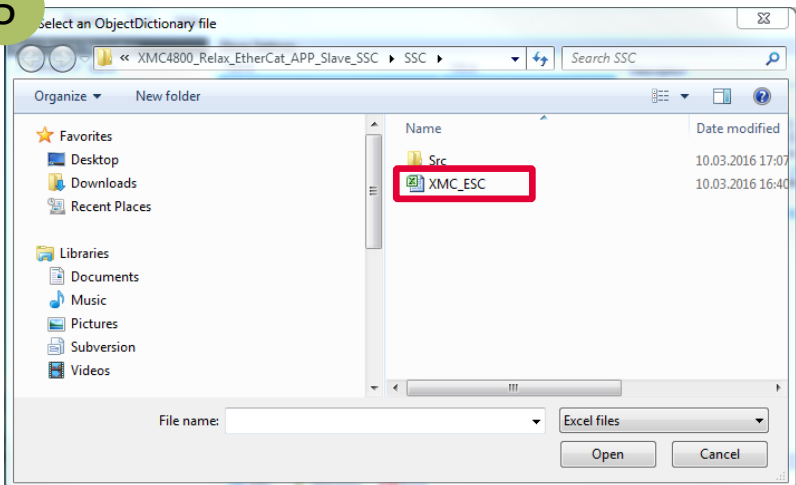
4



4

Import the EXCEL-sheet which defines the interface of your EtherCAT node.

5

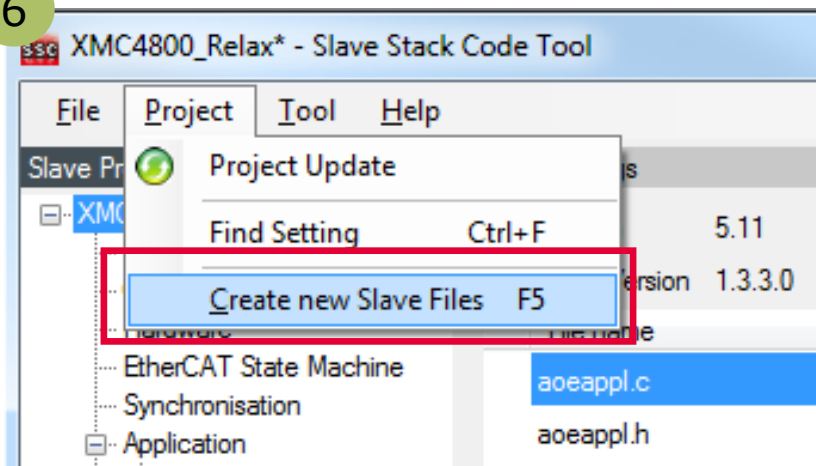


5

Select the EXCEL-file provided inside the example project.

Generating Slave Stack Code and ESI file

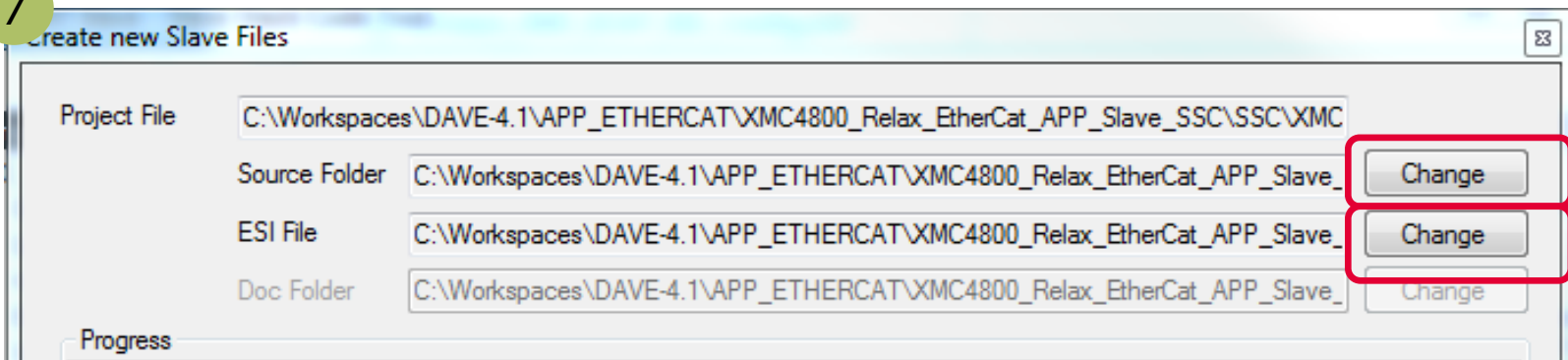
6



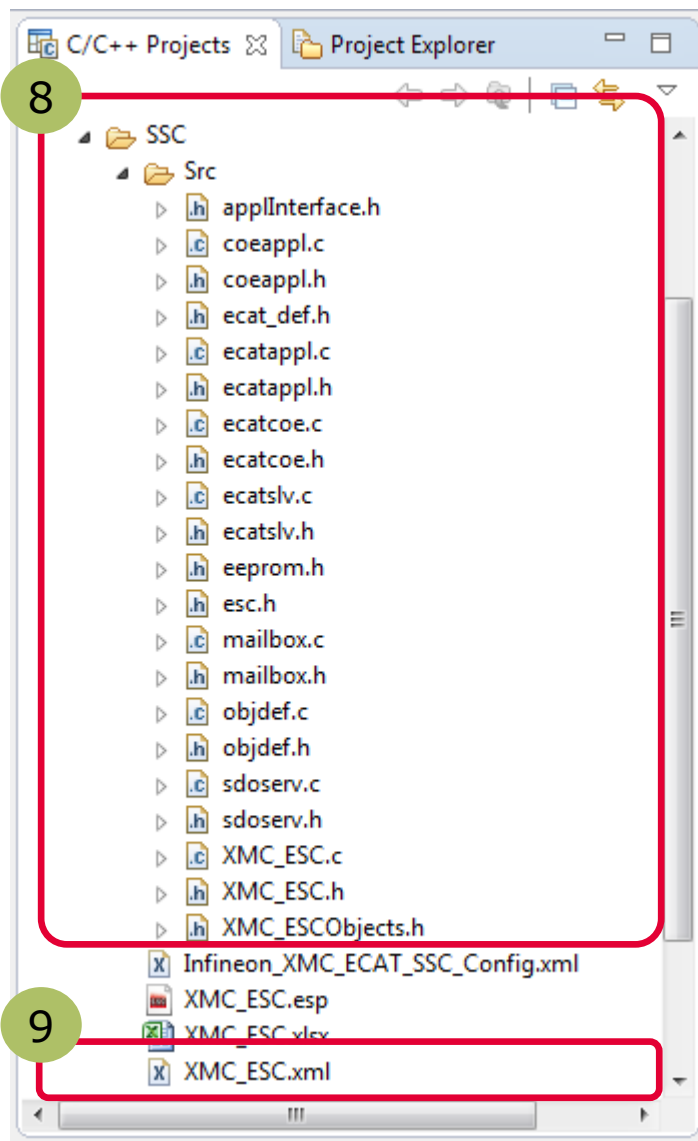
6 Click on **Project** >> **Create new Slave Files** to start file generation.

7 In this step the destination folder for the EtherCAT Slave Stack Code and the ESI file can be adapted. For this example it is recommended to take the default settings.

7

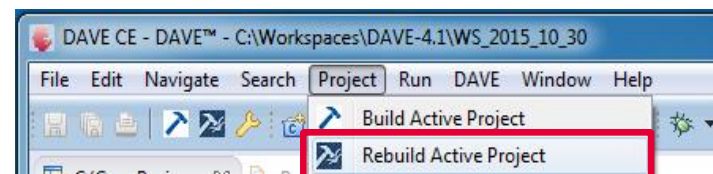


Find and use your result



After the generation process the respective files are inside the project space:

- 8 Check the availability of the generated slave stack code
- 9 Check the availability of the ESI-file and download to the host by these 3 steps:
 1. Stop TwinCAT System Manager
 2. Copy the ESI file to resp. destination for TwinCAT2:
C:\TwinCAT\Io\EtherCAT
for TwinCAT3:
C:\TwinCAT\3.1\Config\Io\EtherCAT
 3. Restart TwinCAT System Manager to start re-work of the device description cache.
- 10 Rebuild the DAVE project with the new files.



- 1 Overview and Requirements
- 2 Setup
- 3 Defining the interface of EtherCAT slave node
- 4 Generating Slave Stack Code and ESI file
- 5 **Implementation of the application**
- 6 How to test – using TwinCAT2 as host
- 7 How to test – using TwinCAT3 as host

Copy data from/to local data to/from ESC memory

Inside the generated file *XMC_ESC.c* the link to your application must be implemented. Modify the source code accordingly which copies the application data to/from ESC memory to the local application memory:

Originally generated code:

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/**
 \param   pData pointer to input process data

 \brief   This function will copies the inputs from the local memory to the ESC memory
 to the hardware
 *////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void APPL_InputMapping(UINT16* pData)
{
 #if WIN32
 #pragma message ("Warning: Implement input (Slave -> Master) mapping")
 #else
 #warning "Implement input (Slave -> Master) mapping"
 #endif
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/**
 \param   pData pointer to output process data

 \brief   This function will copies the outputs from the ESC memory to the local memory
 to the hardware
 *////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void APPL_OutputMapping(UINT16* pData)
{
 #if WIN32
 #pragma message ("Warning: Implement output (Master -> Slave) mapping")
 #else
 #warning "Implement output (Master -> Slave) mapping"
 #endif
}
```



Modified code:

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/**
 \param   pData pointer to input process data

 \brief   This function will copies the inputs from the local memory to the ESC memory
 to the hardware
 *////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void APPL_InputMapping(UINT16* pData)
{
 memcpy(pData, &((UINT16 *) &IN_GENERIC0x6000)[1], sizeof(IN_GENERIC0x6000)-2);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/**
 \param   pData pointer to output process data

 \brief   This function will copies the outputs from the ESC memory to the local memory
 to the hardware
 *////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void APPL_OutputMapping(UINT16* pData)
{
 memcpy(&((UINT16 *) &OUT_GENERIC0x7000)[1], pData, sizeof(OUT_GENERIC0x7000)-2);
}
```



Implement application specific slave node behaviour

Inside the generated file *XMC_ESC.c* file the function *APPL_Application* is implemented. This function implements the application specific code to handle input and output...

A) ... from mainloop or

B) ... if synchronisation is active from ISR

Inside *main.c* of the example, the function

*void process_app(TOBJ7000 *OUT_GENERIC, TOBJ6000 *IN_GENERIC);*

implements the mapping of the input/output data to buttons and LEDs. Therefore please modify the function *APPL_Application* to call *process_app* in the following way:

Originally generated code:

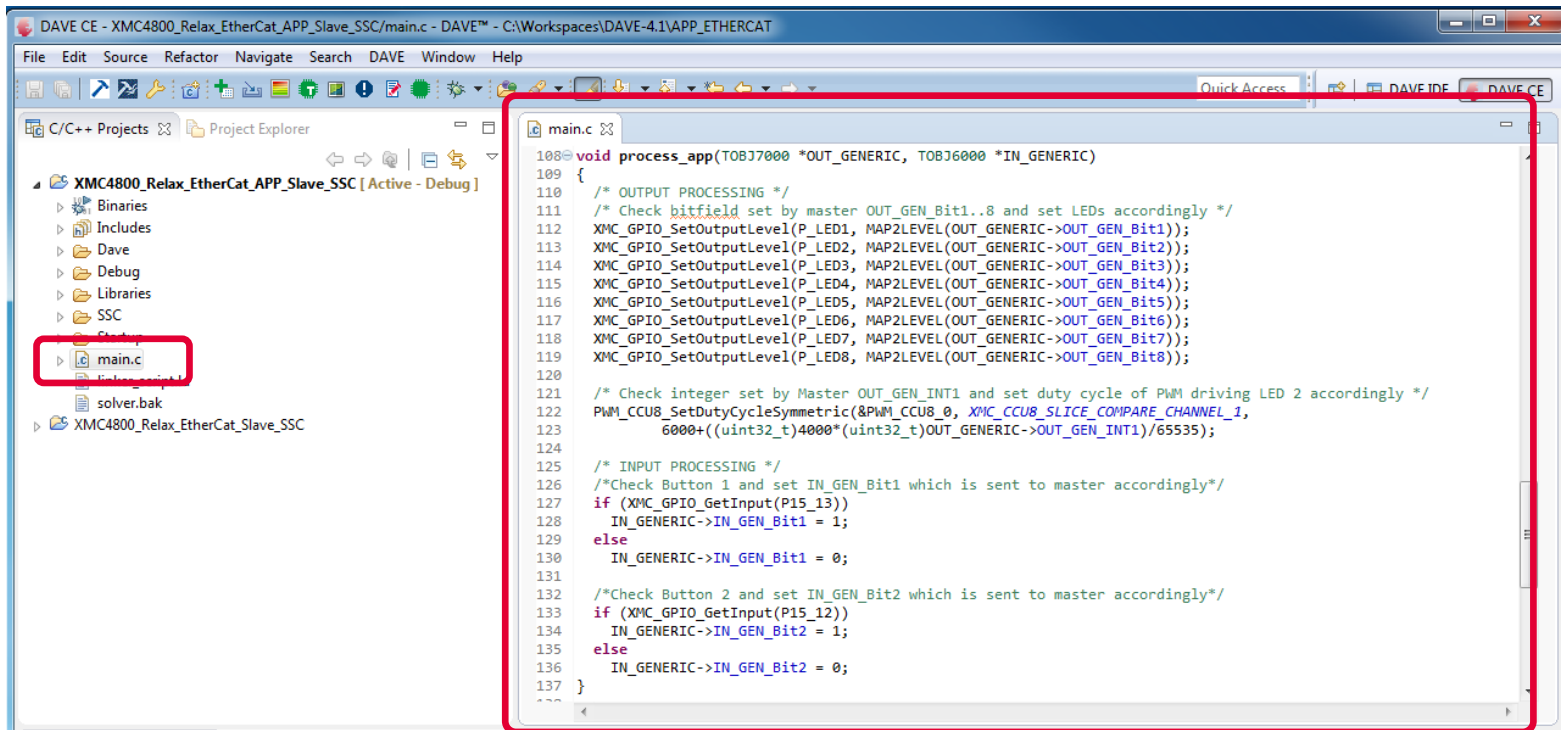
```
////////////////////////////////////  
/**  
\brief This function will called from the synchronisation ISR  
or from the mainloop if no synchronisation is supported  
*////////////////////////////////////  
void APPL_Application(void)  
{  
#if WIN32  
#pragma message ("Warning: Implement the slave application")  
#else  
#warning "Implement the slave application."  
#endif  
}
```



Modified code:

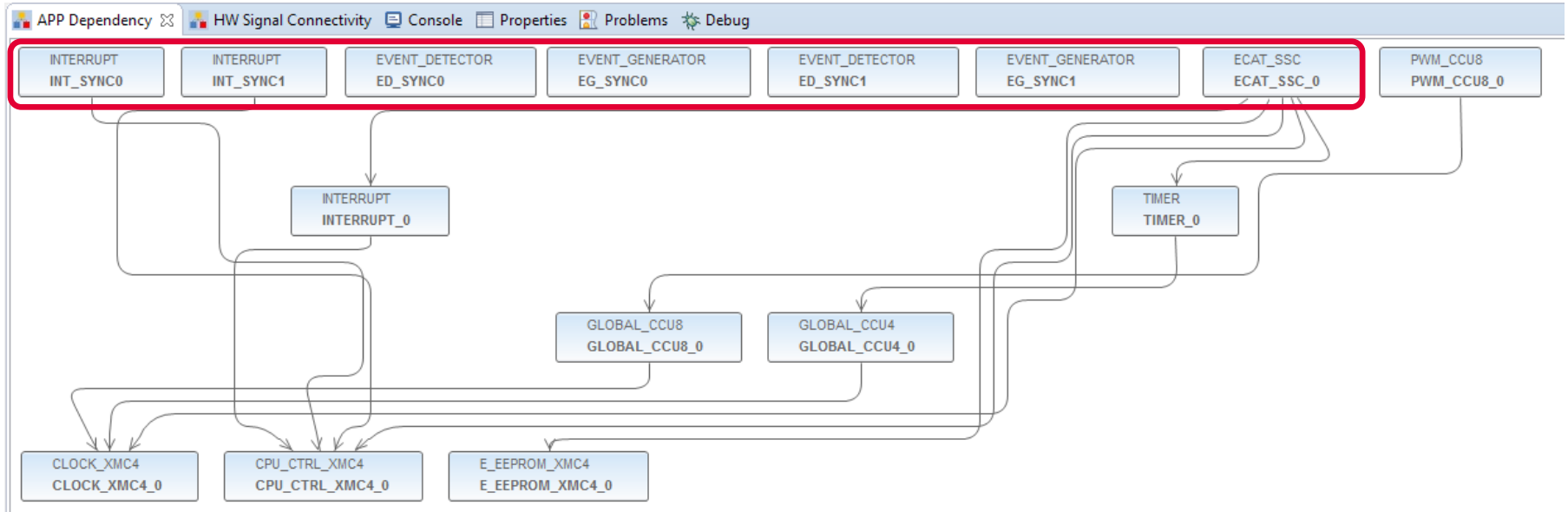
```
////////////////////////////////////  
/**  
\brief This function will called from the synchronisation ISR  
or from the mainloop if no synchronisation is supported  
*////////////////////////////////////  
void process_app(TOBJ7000 *OUT_GENERIC, TOBJ6000 *IN_GENERIC);  
void APPL_Application(void)  
{  
process_app(&OUT_GENERIC0x7000, &IN_GENERIC0x6000);  
}
```


Description – process of input and output



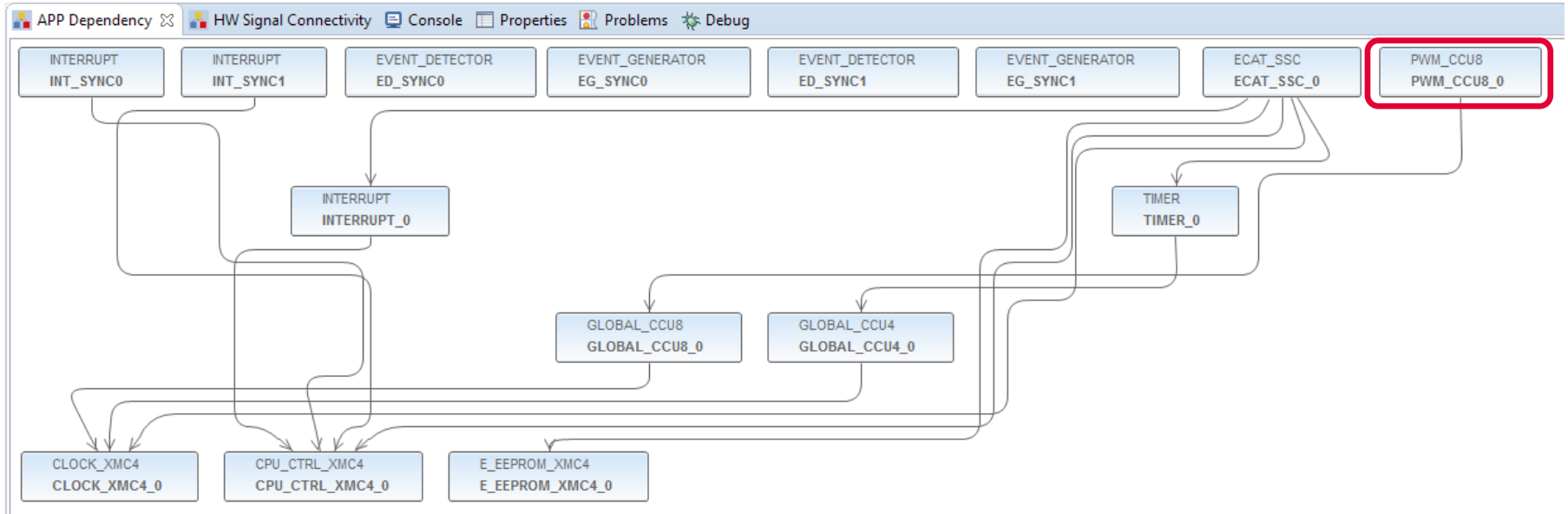
```
108 void process_app(TOB37000 *OUT_GENERIC, TOB36000 *IN_GENERIC)
109 {
110     /* OUTPUT PROCESSING */
111     /* Check bitfield set by master OUT_GEN_Bit1..8 and set LEDs accordingly */
112     XMC_GPIO_SetOutputLevel(P_LED1, MAP2LEVEL(OUT_GENERIC->OUT_GEN_Bit1));
113     XMC_GPIO_SetOutputLevel(P_LED2, 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     /* Check integer set by Master OUT_GEN_INT1 and set duty cycle of PWM driving LED 2 accordingly */
122     PWM_CCUB_SetDutyCycleSymmetric(&PWM_CCUB_0, XMC_CCUB_SLICE_COMPARE_CHANNEL_1,
123     6000+((uint32_t)4000*(uint32_t)OUT_GENERIC->OUT_GEN_INT1)/65535);
124
125     /* INPUT PROCESSING */
126     /*Check Button 1 and set IN_GEN_Bit1 which is sent to master accordingly*/
127     if (XMC_GPIO_GetInput(P15_13))
128         IN_GENERIC->IN_GEN_Bit1 = 1;
129     else
130         IN_GENERIC->IN_GEN_Bit1 = 0;
131
132     /*Check Button 2 and set IN_GEN_Bit2 which is sent to master accordingly*/
133     if (XMC_GPIO_GetInput(P15_12))
134         IN_GENERIC->IN_GEN_Bit2 = 1;
135     else
136         IN_GENERIC->IN_GEN_Bit2 = 0;
137 }
```

Within the slave stack code the function `process_app` is called. This `process_app` function process the binary output data (master->slave) to set the LED1 to LED8 level on the XMC EtherCAT PHY Board. The integer output data is used to set the duty cycle of the dimmable LED2. The states of the buttons are checked and propagated to the input data (slave->master).



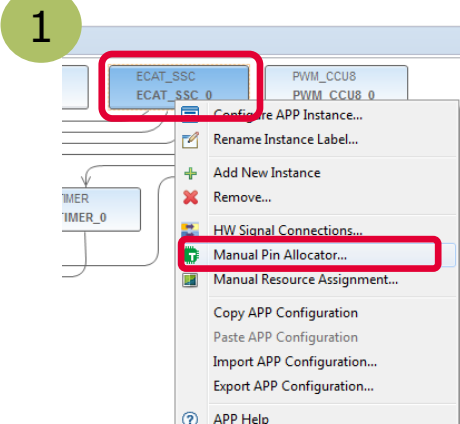
The ECAT_SSC APP assigns the system resources (automatically done by DAVE by using the respective lower level apps) and pins (by manual configuration) to setup a proper EtherCAT communication. The EVENT_DETECTOR, EVENT_GENERATOR and INTERRUPT APPs are used inside this example to connect the sync_out_0 and sync_out_1 of the ECAT_SSC APP to the interrupt service routines of the SSC-stack.

Description – Overview on used APPs

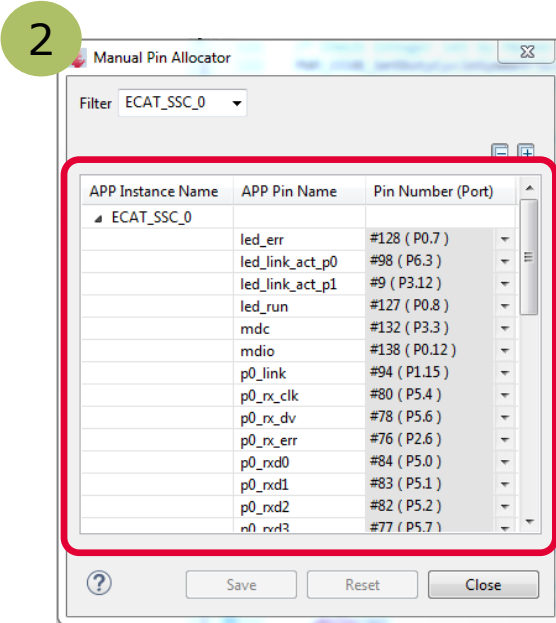


PWM_CCU8 APP is used to control the dimming level of the LED2 on your Relax Kit and to assign the PWM output to the respective pin.

Description – EtherCat ports and physical connection

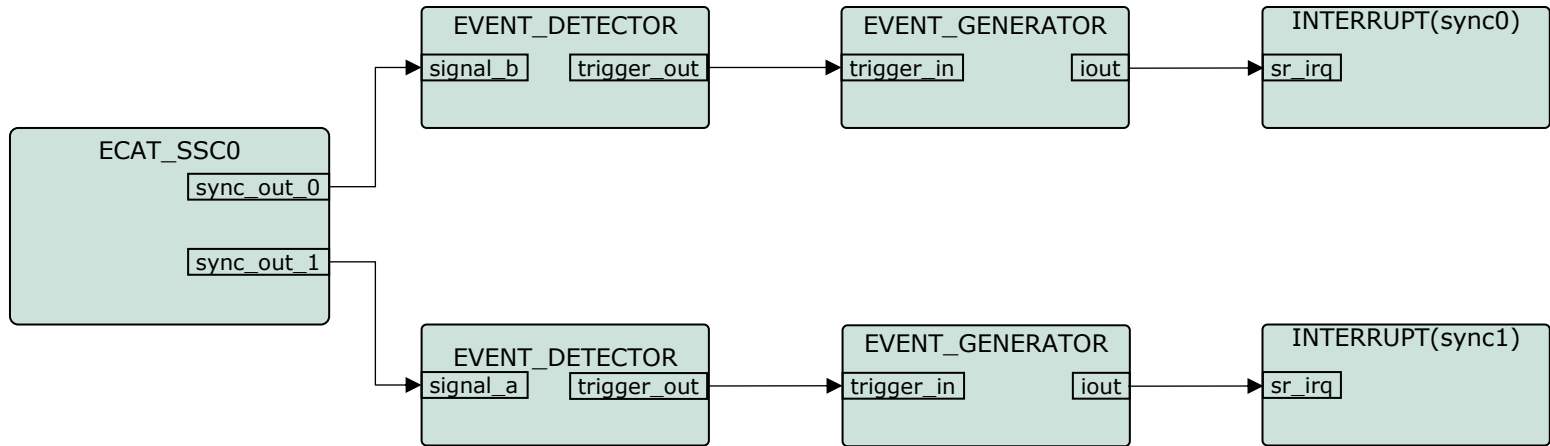


1 Right click on the ECAT_SSC APP. From the context menu select „Manual Pin Allocator“ to open the pin allocation for the EtherCAT module.



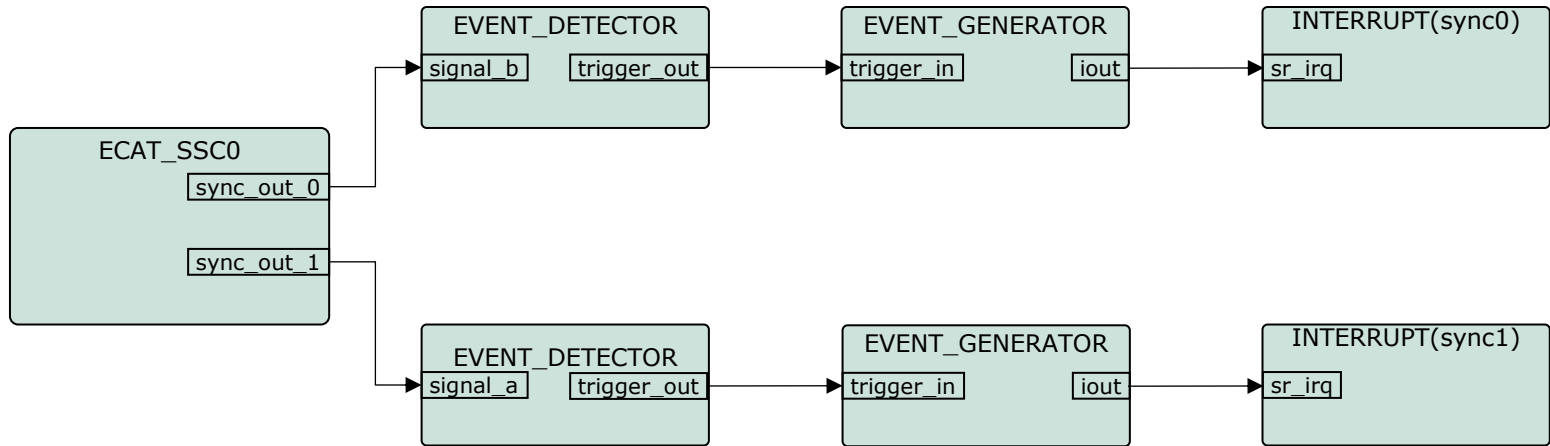
2 Inside Manual Pin Allocator you can configure the EtherCAT ports for your application. For the example provided, the configuration fits to the XMC4800 Relax EtherCAT Kit.

Description – Distributed clock support



For distributed clock support, the sync0 and sync1 signals coming from the ethercat peripheral are used to trigger interrupts. Inside the interrupt service routines the respective API functions of the SSC protocol stack are called.

Description – Overview on propagating the sync0 and sync1 signals to ISR

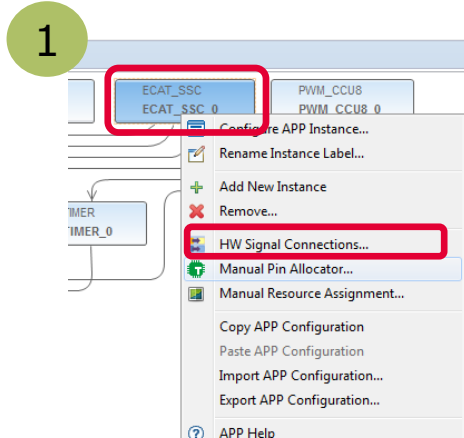


EVENT_DETECTOR and EVENT_GENERATOR APPs are instances of the event request unit (ERU) peripheral. Inside this example the ERU is used to propagate the signals sync0 and sync1 to the interrupt service routines.

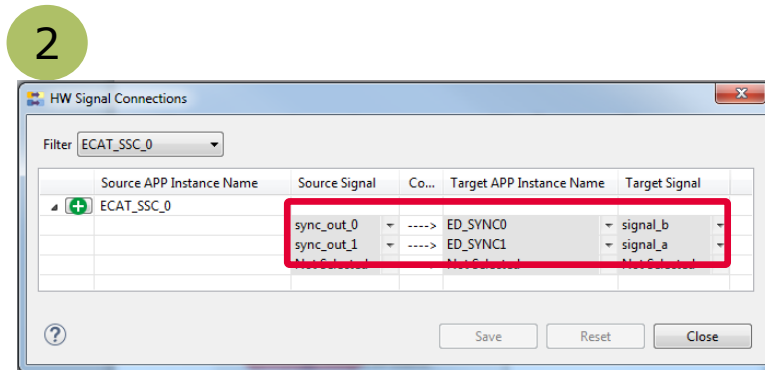
Please see next slides how to setup this configuration inside DAVE™.

ATTENTION: With the same approach sync0 and sync1 signals can also be connected to other resources. For example: ADC, ports and timers.

Description – DAVE™ settings for distributed clock support

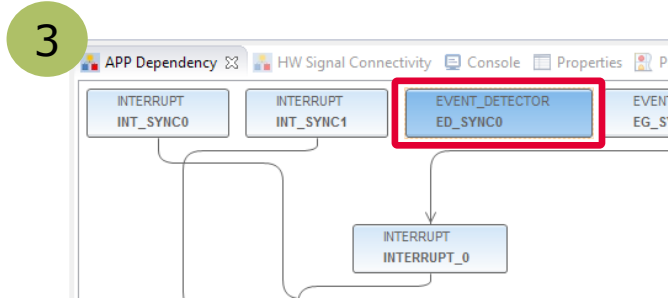


1 Right click on the ECAT_SSC APP. From the context menu select „HW Signal Connections “ to open the HW Signal Connection dialog of the ECAT_SSC APP.

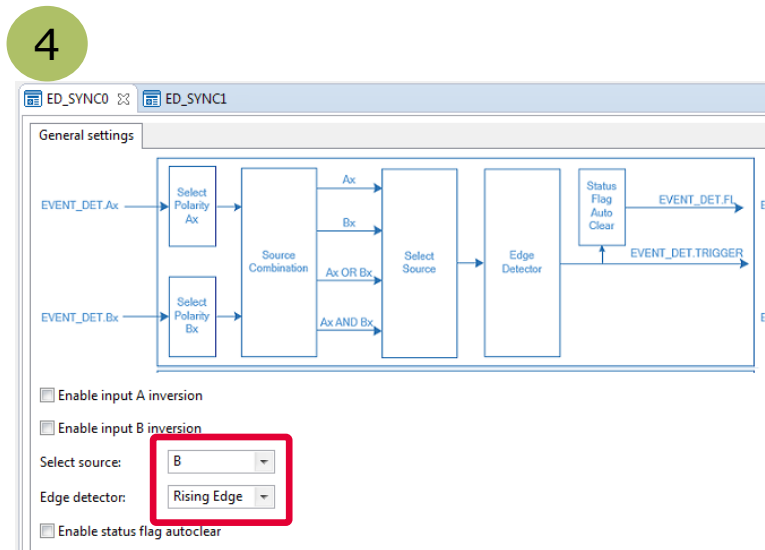


2 Connect the sync_out_0 and sync_out_1 signal to the a/b input of the event detection units.

Description – DAVE™ settings for distributed clock support

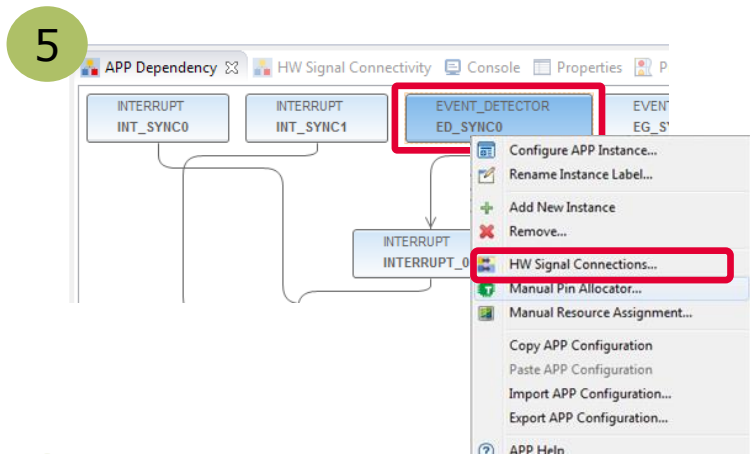


3 Double click on the EVENT_DETECTOR APP for SYNC0 and EVENT_DETECTOR APP for SYNC1.

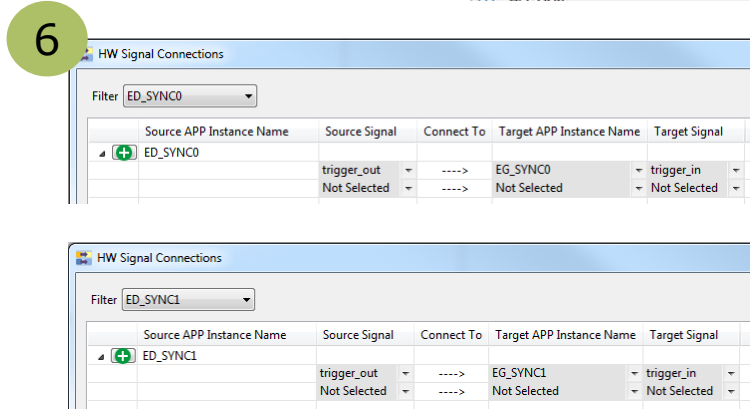


4 Select the respective source signal („A“ for SYNC0 and „B“ for SYNC1) and edge detection „Rising Edge“.

Description – DAVE™ settings for distributed clock support

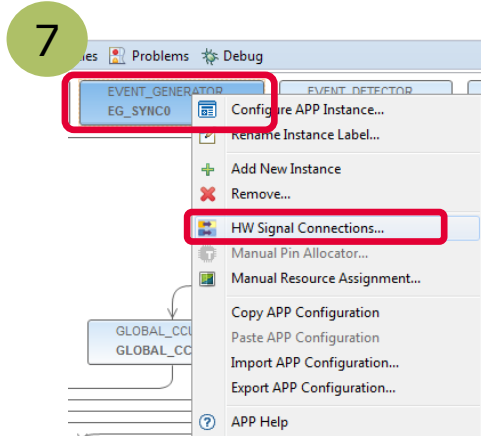


5 Right click on the EVENT_DETECTOR APP for SYNC0 and SYNC1. From the context menu select „HW Signal Connections “ to open the HW Signal Connection dialog of the ECAT_SSC APP.

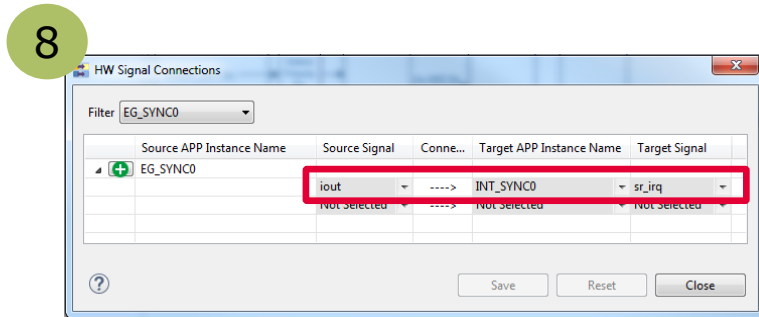


6 Connect the trigger_out signals of the event detection units to the trigger_in signals of the event generation units.

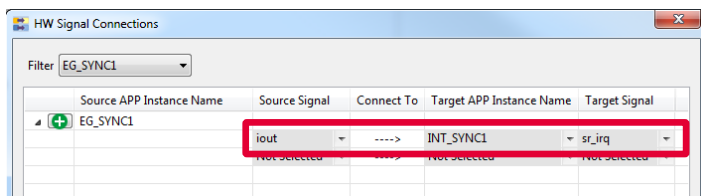
Description – DAVE™ settings for distributed clock support



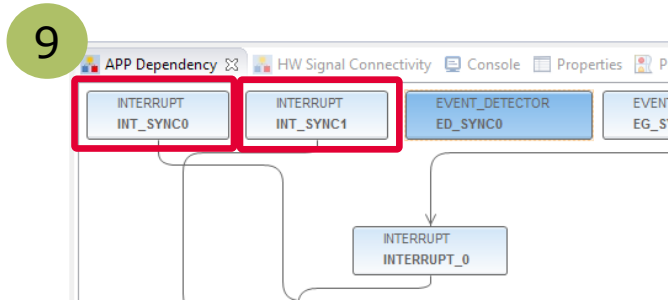
7 Right click on EVENT_GENERATOR for sync0 and sync1. From the context menu select „HW Signal Connections “ to open the HW Signal Connection dialog of the EVENT_GENERATOR APP.



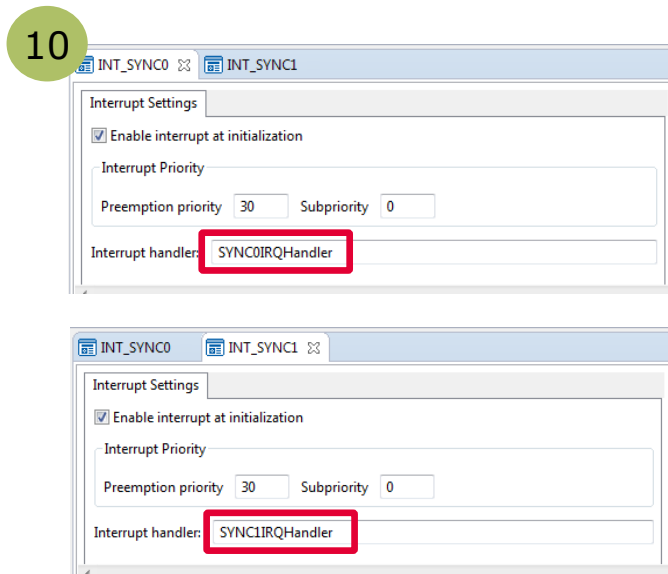
8 Connect the iout of the EVENT_GENERATOR APP for sync0 to INTERRUPT APP of sync0. Proceed respectively for sync1.



Description – DAVE™ settings for distributed clock support

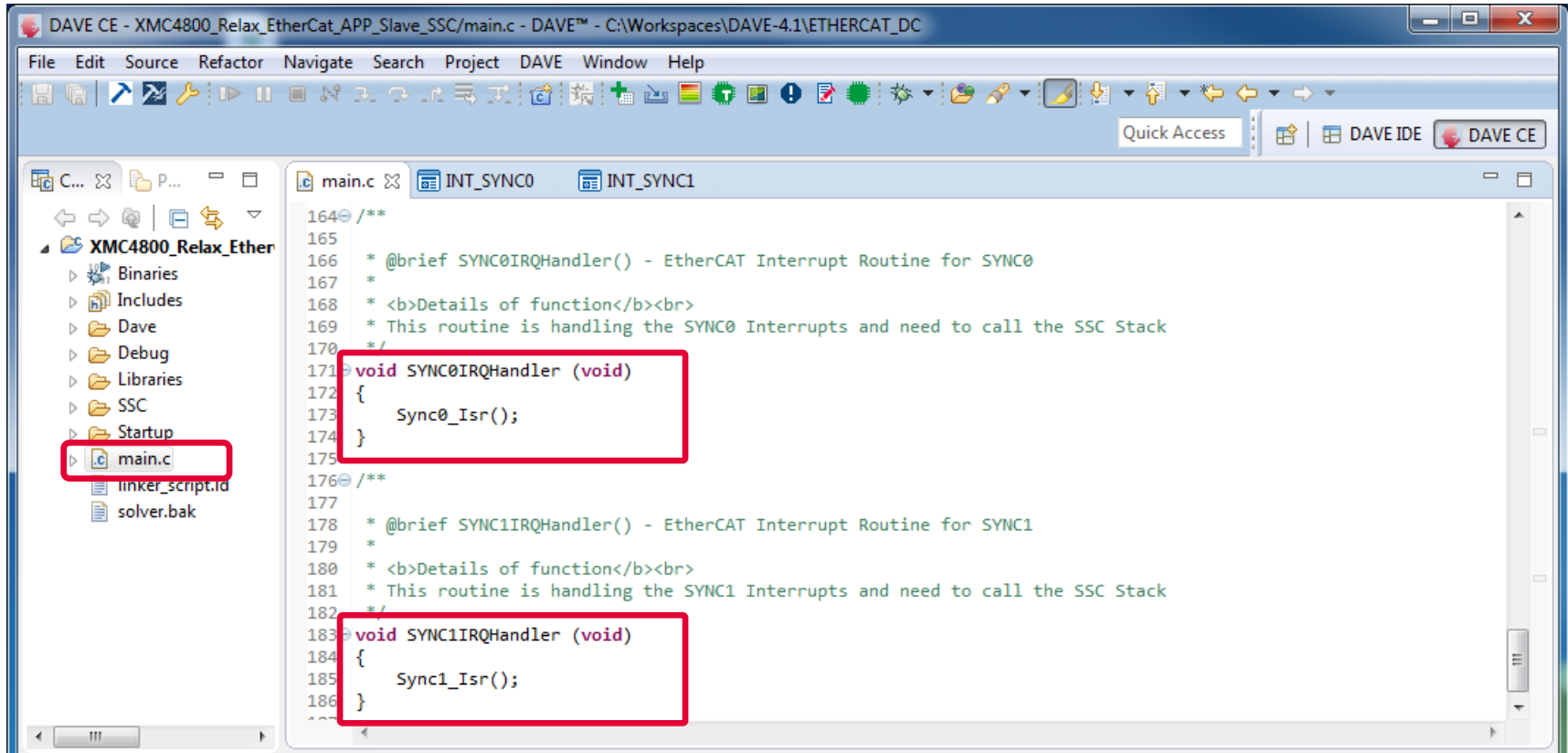


9 Double click on the INTERRUPT APP for sync0 and INTERRUPT for sync1.



10 Set the interrupt service routine for sync0 and sync1 inside the configuration of the respective INTERRUPT APP.

Description – DAVE™ settings for distributed clock support



```
DAVE CE - XMC4800_Relax_EtherCat_APP_Slave_SSC/main.c - DAVE™ - C:\Workspaces\DAVE-4.1\ETHERCAT_DC
File Edit Source Refactor Navigate Search Project DAVE Window Help
Quick Access DAVE IDE DAVE CE
main.c INT_SYNC0 INT_SYNC1
164 /**
165
166 * @brief SYNC0IRQHandler() - EtherCAT Interrupt Routine for SYNC0
167 *
168 * <b>Details of function</b><br>
169 * This routine is handling the SYNC0 Interrupts and need to call the SSC Stack
170 */
171 void SYNC0IRQHandler (void)
172 {
173     Sync0_Isr();
174 }
175
176 /**
177
178 * @brief SYNC1IRQHandler() - EtherCAT Interrupt Routine for SYNC1
179 *
180 * <b>Details of function</b><br>
181 * This routine is handling the SYNC1 Interrupts and need to call the SSC Stack
182 */
183 void SYNC1IRQHandler (void)
184 {
185     Sync1_Isr();
186 }
```

Inside main() the interrupt handlers for sync0 and sync1 are implemented. The implementation is calling the respective functions of the SSC protocol stack.

Description – Setup PWM for dimmable LED2 on P5.8

The screenshot shows the configuration for the PWM_CCU8_0 application. The 'General Settings' tab is active, showing the following parameters:

- Clock frequency [MHz]: 144
- Counting mode: Edge Aligned
- Compare mode: Symmetric
- Start during initialization: (highlighted with a red box)
- Single-shot mode:
- PWM resolution [nsec]: 100
- Actual PWM resolution [nsec]: 55.555556
- Prescaler: 3
- Period register: 0x8C9F
- Frequency [Hz]: 500 (highlighted with a red box)
- Actual frequency [Hz]: 500
- Channel 1 duty cycle [%]: 50 (highlighted with a red box)
- Compare 1: 0x64

The 'HW Signal Connectivity' diagram below shows the hardware connections. The PWM_CCU8_0 application is highlighted with a red box. It is connected to the TIMER_0 and INTERRUPT_0 blocks. The TIMER_0 block is connected to the GLOBAL_CCU4_0 and GLOBAL_CCU8_0 blocks. The INTERRUPT_0 block is connected to the GLOBAL_CCU8_0 block. The GLOBAL_CCU4_0 block is connected to the CLOCK_XMC4_0, CPU_CTRL_XMC4_0, and E_EEPROM_XMC4_0 blocks. The GLOBAL_CCU8_0 block is connected to the CPU_CTRL_XMC4_0 and E_EEPROM_XMC4_0 blocks.

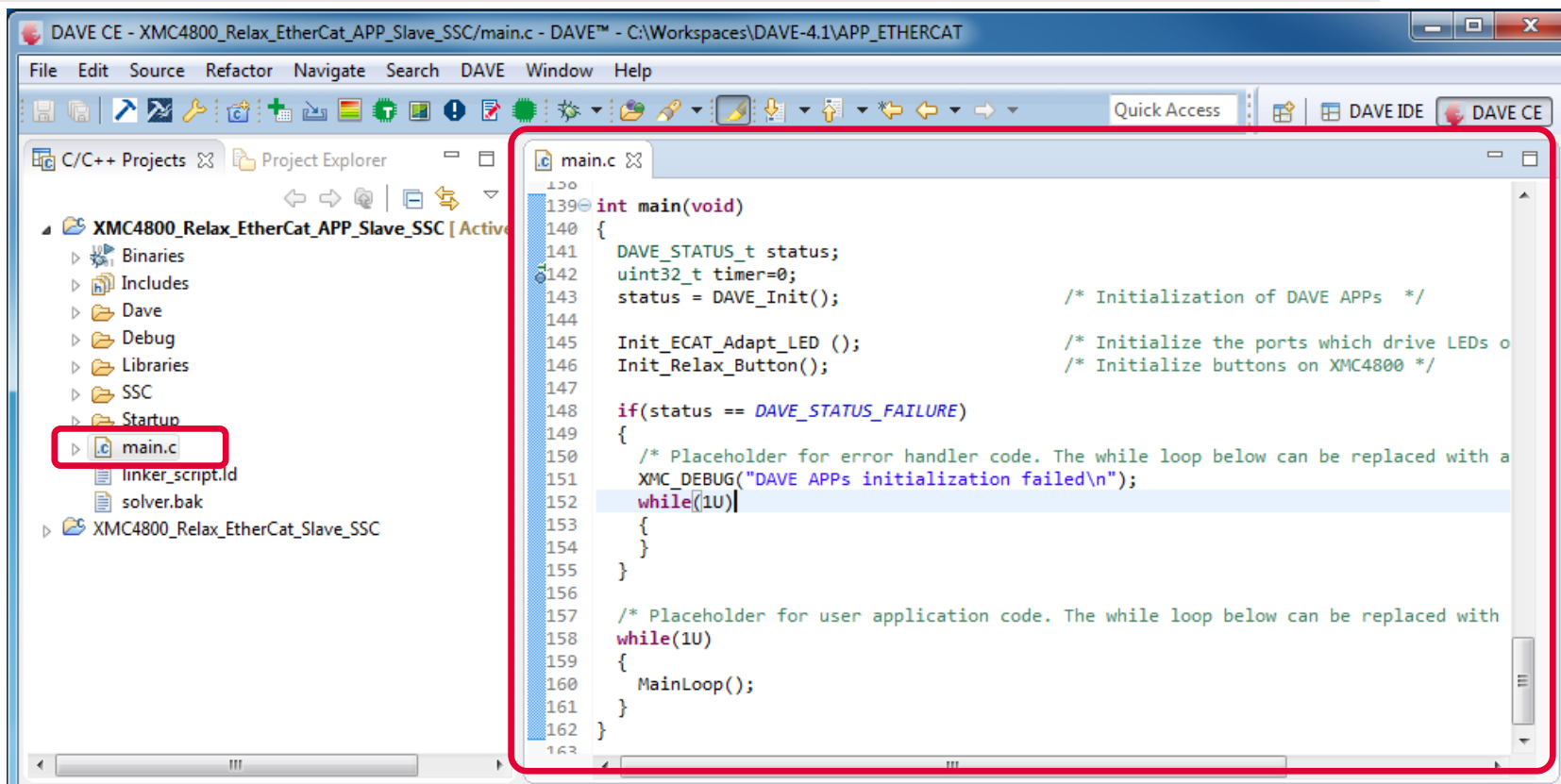
For driving the dimmable LED2 on the XMC4800 Relax EtherCAT Kit the DAVE App PWM_CCU8 is used. The PWM is set to start during initialization, frequency is 500Hz and the initial duty cycle is set to 50%. The pin P5.8 is allocated to channel1 of CCU8 inside the manual pin allocator (right-click PWM_CCU8 APP).

The screenshot shows the Manual Pin Allocator tool. The filter is set to PWM_CCU8_0. The table below shows the pin allocation for the PWM_CCU8_0 application instance.

APP Instance Name	APP Pin Name	Pin Number (Port)
PWM_CCU8_0	PWM_CCU8 CH1 Invert Out	#58 (P5.8) (highlighted with a red box)

Buttons: Save, Reset, Close

Description – initialization inside main.c



```

139 int main(void)
140 {
141     DAVE_STATUS_t status;
142     uint32_t timer=0;
143     status = DAVE_Init();           /* Initialization of DAVE APPs */
144
145     Init_ECAdapt_LED ();           /* Initialize the ports which drive LEDs o
146     Init_Relax_Button();           /* Initialize buttons on XMC4800 */
147
148     if(status == DAVE_STATUS_FAILURE)
149     {
150         /* Placeholder for error handler code. The while loop below can be replaced with a
151         XMC_DEBUG("DAVE APPs initialization failed\n");
152         while(1U)
153         {
154         }
155     }
156
157     /* Placeholder for user application code. The while loop below can be replaced with
158     while(1U)
159     {
160         MainLoop();
161     }
162 }
163
  
```

Inside main() DAVE and its APPs (PWM_CCU8, ECAT_SSC) are initialized. InitECAT_Adapt_LED() and Init_Relax-Button() are used to initialize the buttons and LED1 to 8 of the „XMC EtherCAT PHY Board“. Finally the MainLoop is called cyclically to process the state machine of the slave stack code.

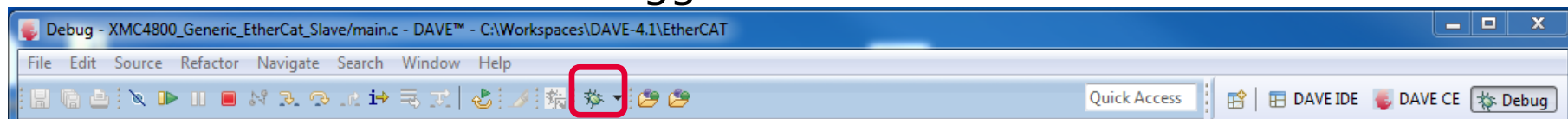
- 1 Overview and Requirements
- 2 Setup
- 3 Defining the interface of EtherCAT slave node
- 4 Generating Slave Stack Code and ESI file
- 5 Implementation of the application
- 6 How to test – using TwinCAT2 as host
- 7 How to test – using TwinCAT3 as host

How to test – start the slave to run

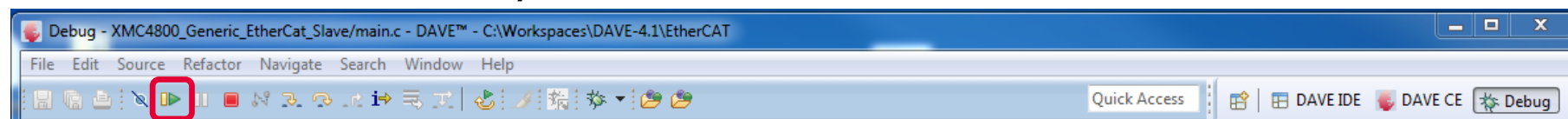


ACTIONS

1. Build and download the example application software to the XMC4800 and start the debugger



2. Start the software by the run button

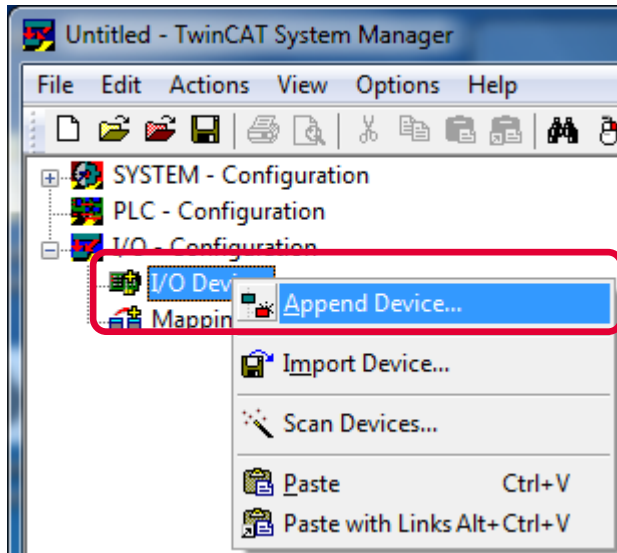


OBSERVATIONS

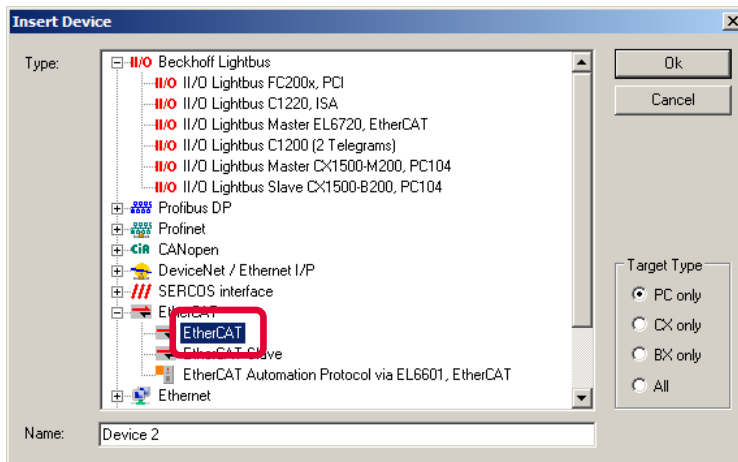
1. The ERR-LED on the "XMC EtherCAT PHY Board" will turn on and immediately turn off again.
2. The LED2 on the "XMC4800 Relax EtherCAT Kit" will remain turned on.

How to test – start the TwinCAT 2 master to run (1/4)

1



2



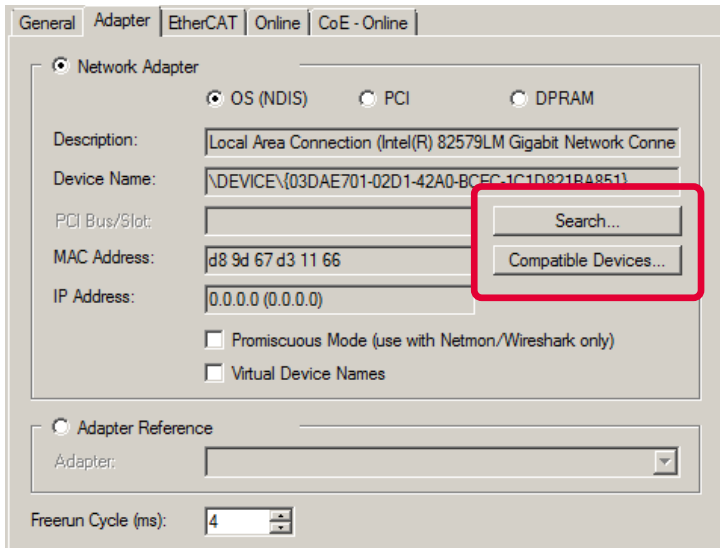
ACTIONS

After starting the TwinCAT System Manager from windows start menu:

- 1 Right Click I/O-Devices and select „Append Device...“
- 2 Create an EtherCAT master device by double click

How to test – start the TwinCAT 2 master to run (2/4)

3



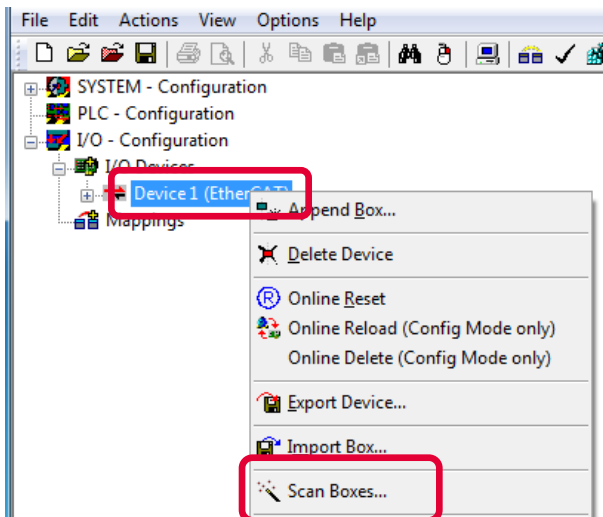
ACTIONS

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.

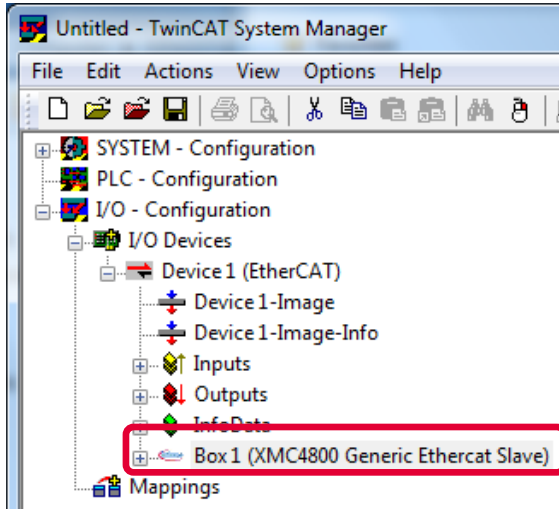
4



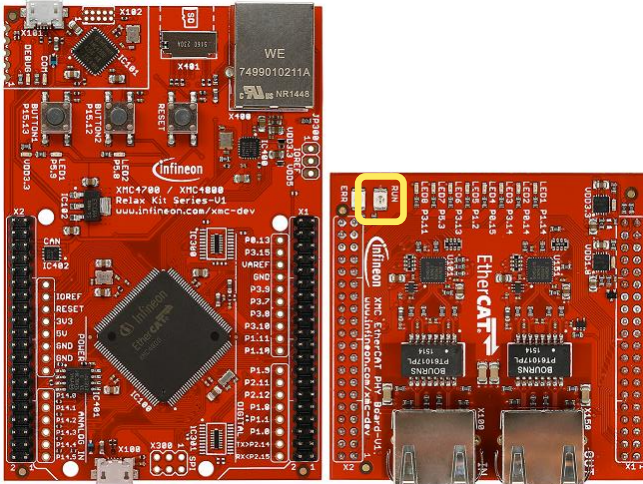
4 Right Click EtherCAT master and select „Scan Boxes...“

How to test – start the TwinCAT 2 master to run (3/4)

1



2

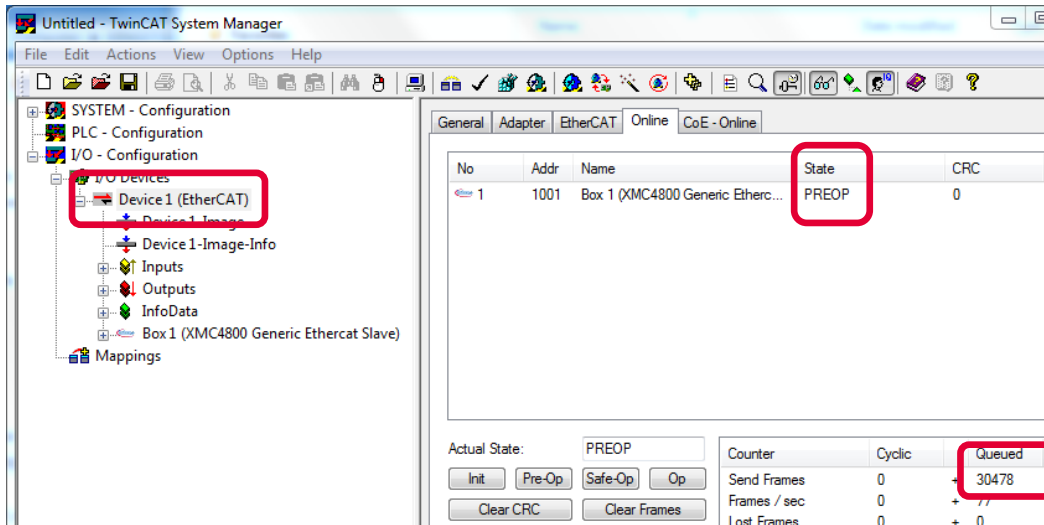


OBSERVATIONS

- 1 The slave appears as a node on the EtherCAT master bus.
- 2 The RUN-LED is flashing indicating PREOP-state

How to test – start the TwinCAT 2 master to run (4/4)

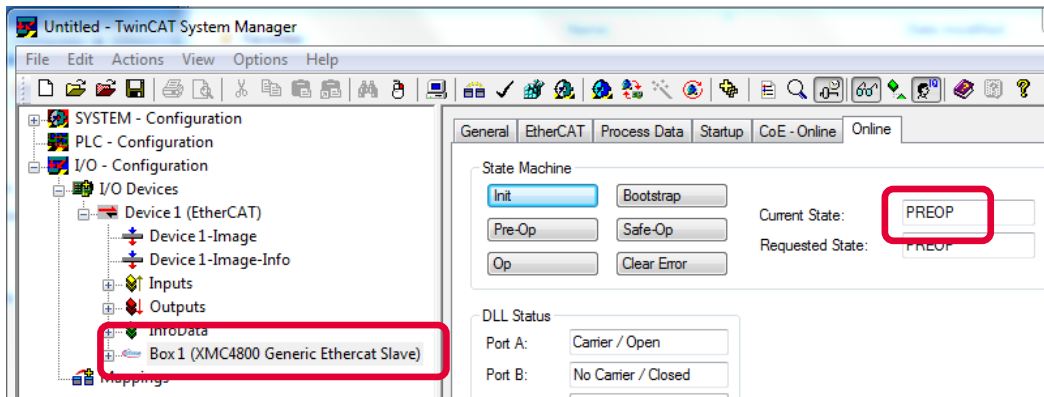
3



OBSERVATIONS

3 EtherCAT master view: Inside the EtherCAT master online state you see the queued frames counting up, the connected slave and its PREOP state.

4

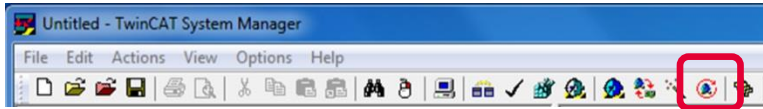


4 EtherCAT slave view: The PREOP-state of the slave is indicated within the TwinCAT system manager .

How to test – Setting slave to operational mode

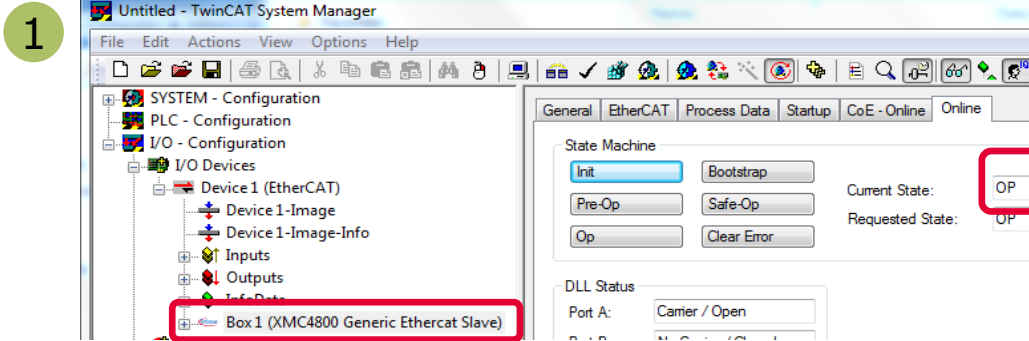


ACTION

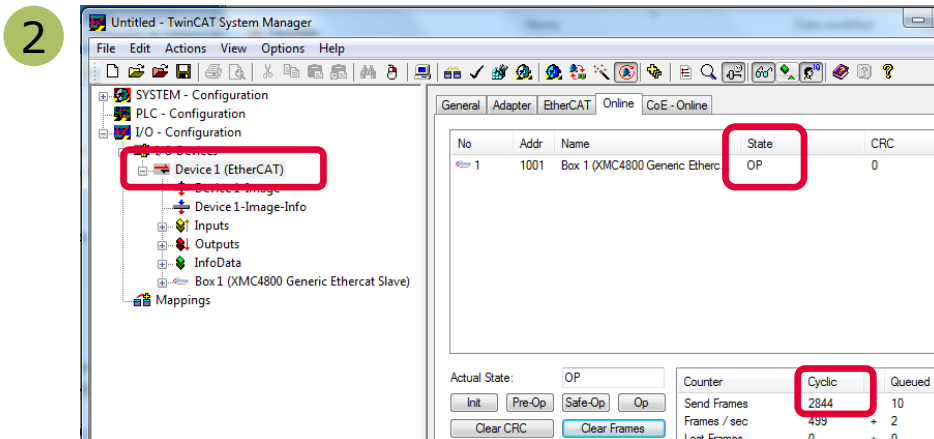


Set master device to free run mode

OBSERVATIONS



1 EtherCAT slave view:
Online status of slave shows the slave in OP state



2 EtherCAT master view:
Online status of master shows the slave in OP state. Frames are no more queued. Cyclic counter is incrementing.

3 „XMC EtherCAT PHY Board“:
RUN-LED is static turned on indicating OP-state.

How to test – Monitoring slave inputs on master

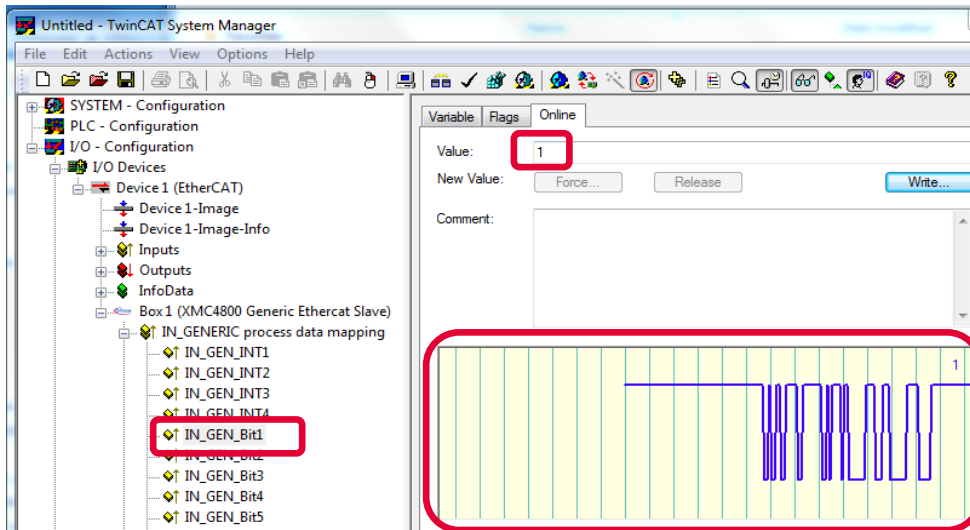


ACTIONS

While pushing Button1 on „XMC4800 Relax EtherCAT Kit“ the button state is updated on the host.



OBSERVATIONS



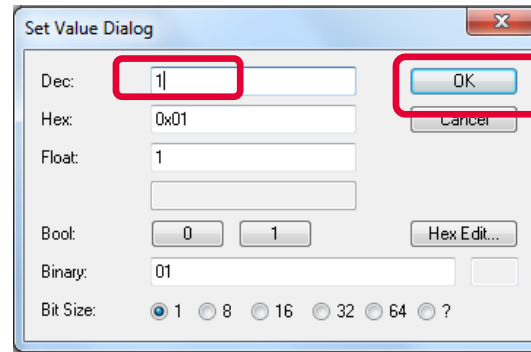
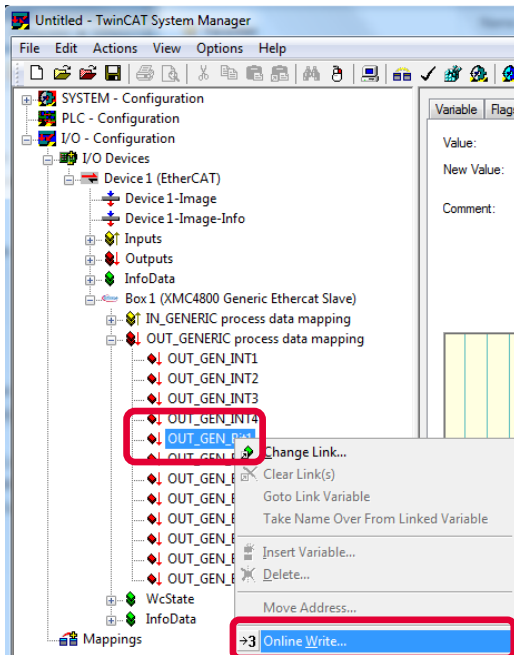
State of IN_GEN_Bit1 changes according to the state of BUTTON1. Same is true for IN_GEN_Bit2 and BUTTON2.

How to test – Setting slave outputs on master (1/2)



ACTIONS

Right click on OUT_GEN_Bit1 of the slave node and select „Online Write...” inside the context menu. Change the value from 0 to 1 to switch on LED1/ from 1 to 0 to switch off LED1.



OBSERVATION

LED1 „XMC EtherCAT PHY Board“ is turned on/off according to OUT_GEN_Bit1 setting.

How to test – Setting slave outputs on master (2/2)



ACTION

1. Right click on OUT_GEN_INT1 of the slave node and select „Online Write...” inside the context menu. Change the value from 0 to 50000.



OBSERVATION

1. Brightness of LED2 on „XMC4800 Relax EtherCAT Kit” is dimmed. The OUT_GEN_INT1 value sets the brightness of LED 2. The lower the value the brighter the LED2. To turn off LED2 just set value to max (65535).



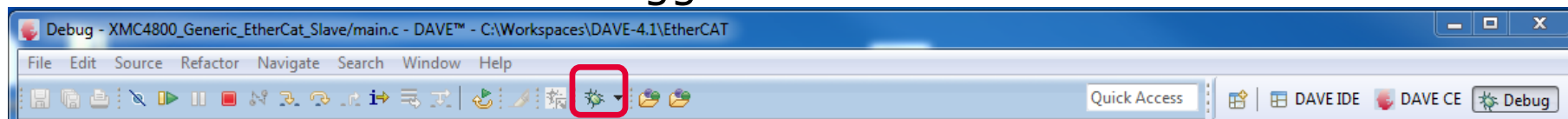
- 1 Overview and Requirements
- 2 Setup
- 3 Defining the interface of EtherCAT slave node
- 4 Generating Slave Stack Code and ESI file
- 5 Implementation of the application
- 6 How to test – using TwinCAT2 as host
- 7 How to test – using TwinCAT3 as host

How to test – start the slave to run

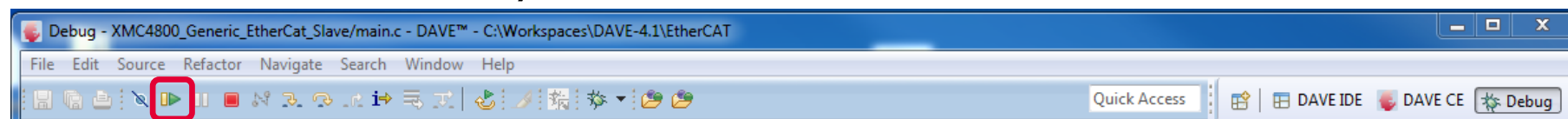


ACTIONS

1. Build and download the example application software to the XMC4800 and start the debugger



2. Start the software by the run button

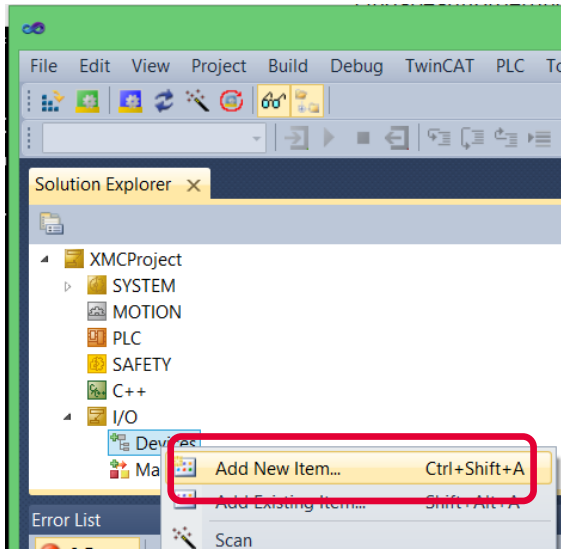


OBSERVATIONS

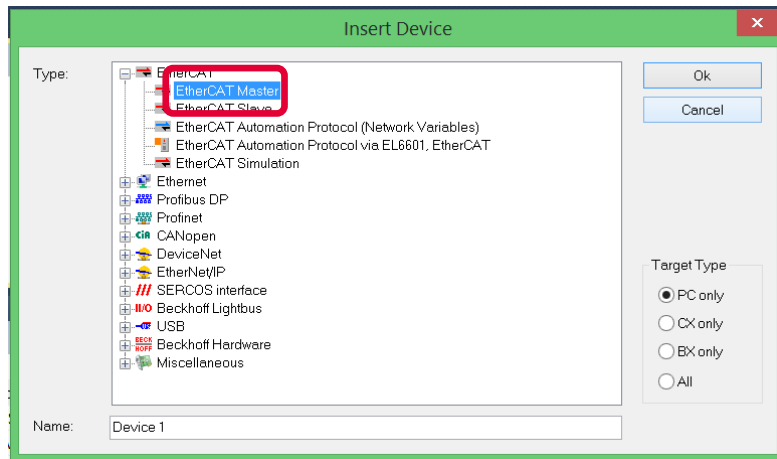
1. The ERR-LED on the "XMC EtherCAT PHY Board" will turn on and immediately turn off again.
2. The LED2 on the "XMC4800 Relax EtherCAT Kit" will remain turned on.

How to test – start the TwinCAT 3 master to run (1/4)

1



2



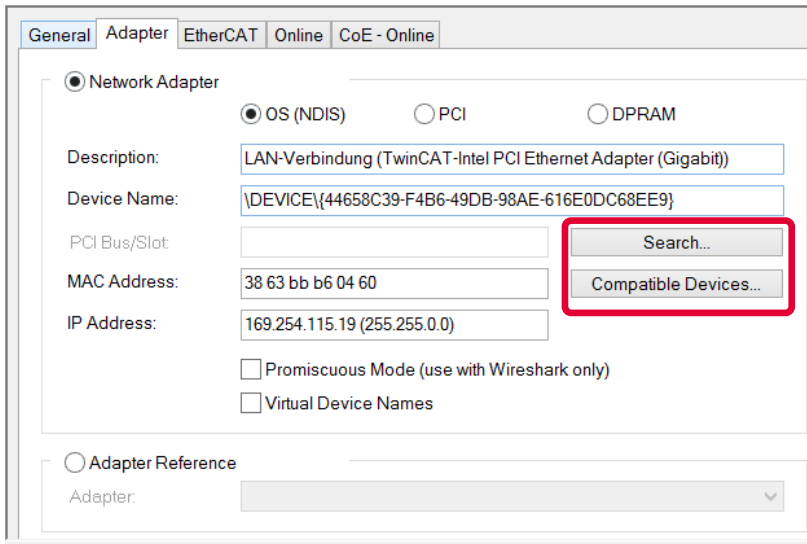
ACTIONS

After starting the TwinCAT System Manager from windows start menu:

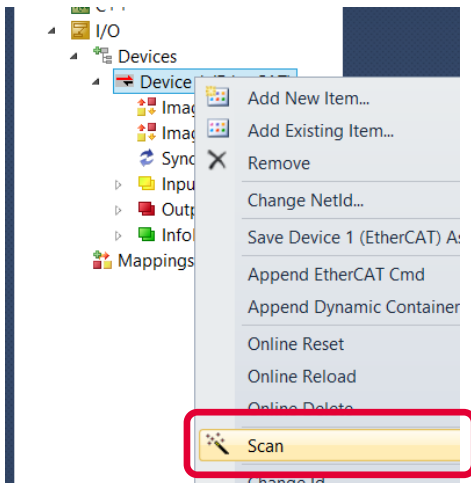
- 1 Right Click I/O-Devices and select „Add New Item...“
- 2 Create an EtherCAT master device by double click

How to test – start the TwinCAT 3 master to run (2/4)

3



4



ACTIONS

3 Select the network adapter you want to use (search and select).

Application hint:

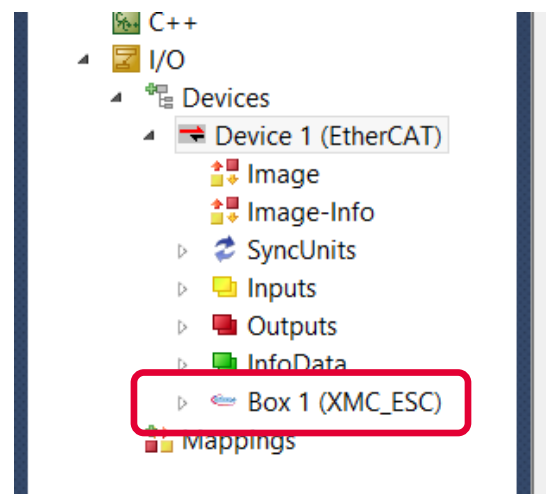
In case the device is not found please install the respective device driver by following the instructions given by TwinCAT through the „Compatible Devices...“ button.

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

How to test – start the TwinCAT 3 master to run (3/4)



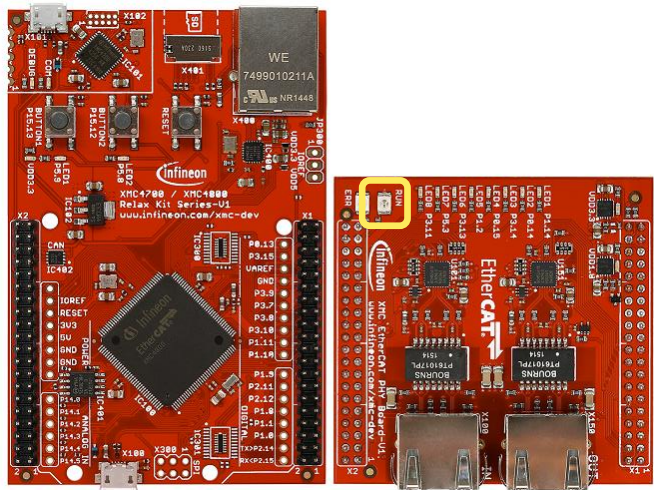
1



OBSERVATIONS

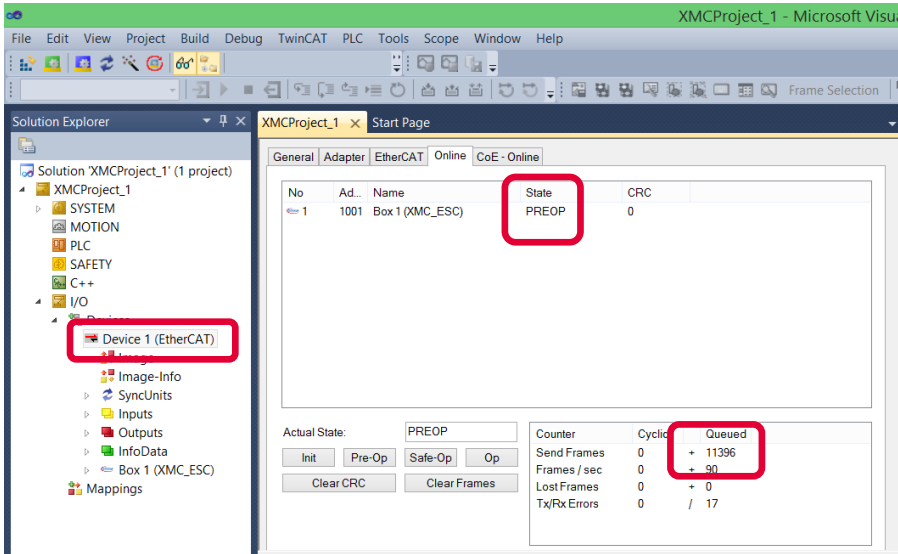
- 1 The slave appears as a node on the EtherCAT master bus.
- 2 The RUN-LED is flashing indicating PREOP-state

2



How to test – start the TwinCAT 3 master to run (4/4)

3

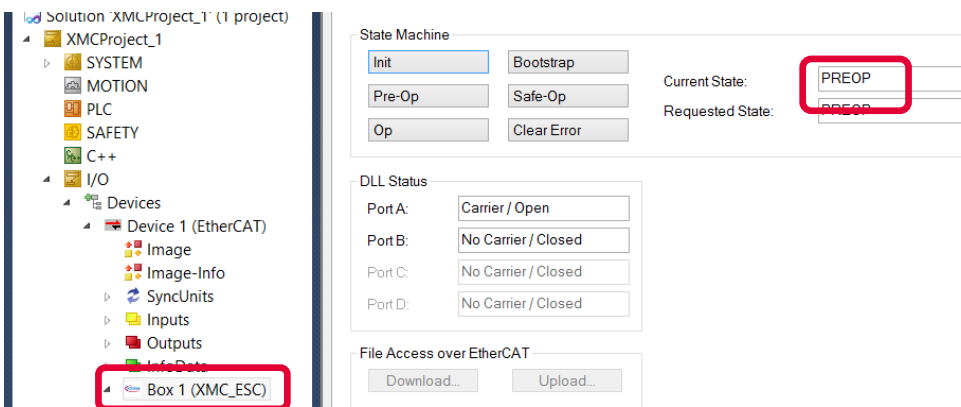


OBSERVATIONS

3 EtherCAT master view: Inside the EtherCAT master online state you see the queued frames counting up, the connected slave and its PREOP state.

4 EtherCAT slave view: The PREOP-state of the slave is indicated within the TwinCAT system manager .

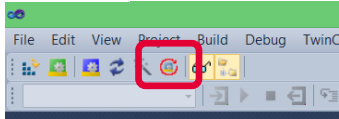
4



How to test – Setting slave to operational mode

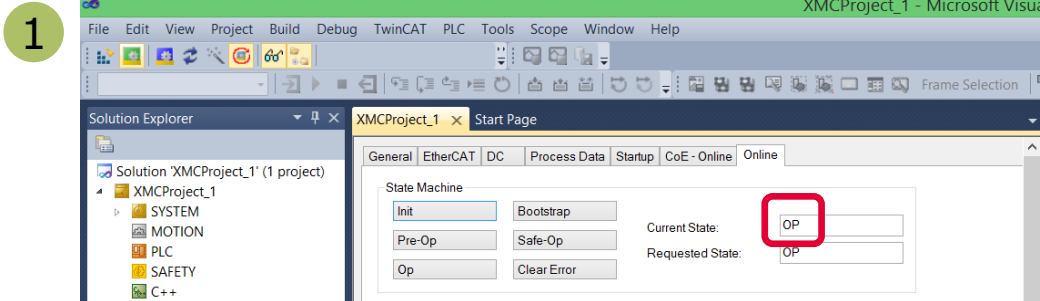


ACTION

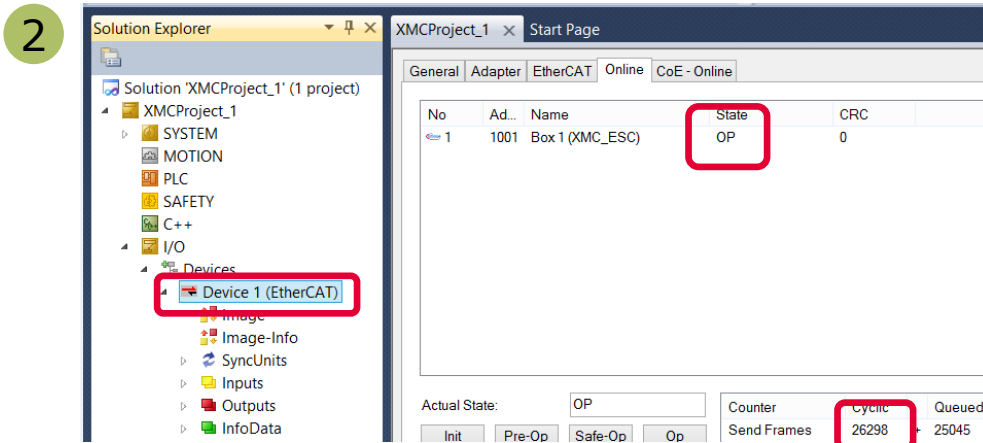


Set master device to free run mode

OBSERVATIONS



1 EtherCAT slave view:
Online status of slave shows the slave in OP state



2 EtherCAT master view:
Online status of master shows the slave in OP state. Frames are no more queued. Cyclic counter is incrementing.

3 „XMC EtherCAT PHY Board“:
RUN-LED is static turned on indicating OP-state.

How to test – Monitoring slave inputs on master

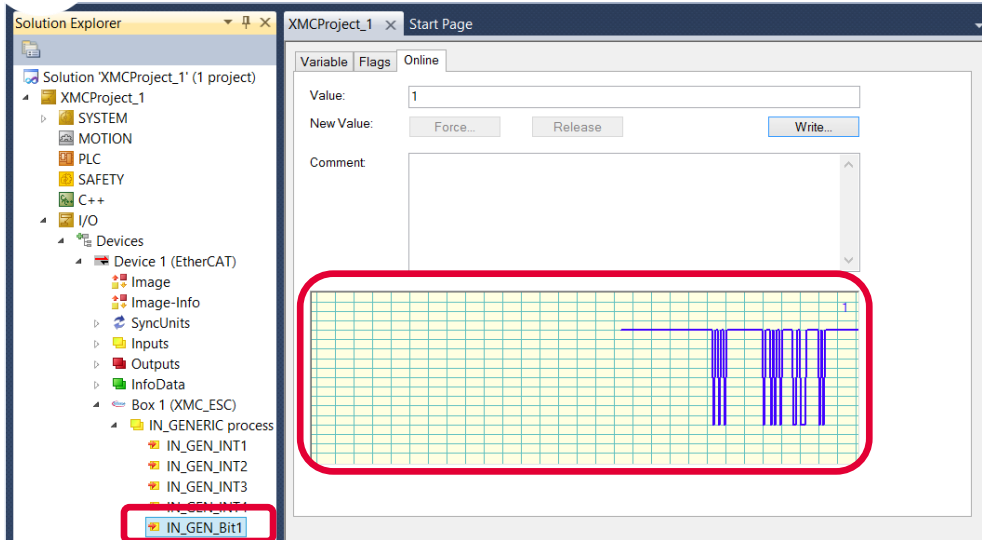


ACTIONS

While pushing Button1 on „XMC4800 Relax EtherCAT Kit“ the button state is updated on the host.



OBSERVATIONS



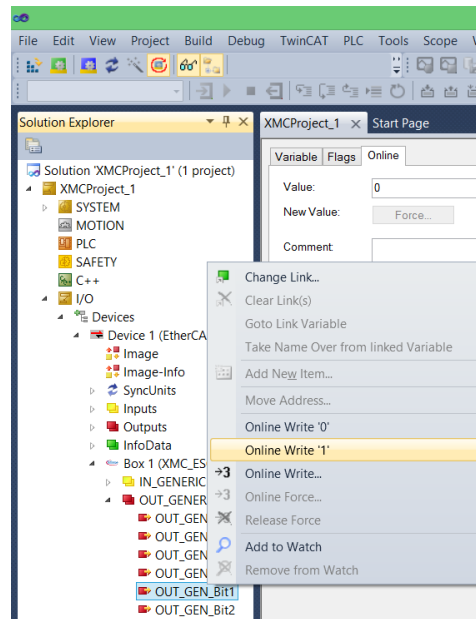
State of IN_GEN_Bit1 changes according to the state of BUTTON1. Same is true for IN_GEN_Bit2 and BUTTON2.

How to test – Setting slave outputs on master (1/2)



ACTIONS

Right click on OUT_GEN_Bit1 of the slave node and select „Online Write...” inside the context menu. Change the value from 0 to 1 to switch on LED1/ from 1 to 0 to switch off LED1.



OBSERVATION

LED1 „XMC EtherCAT PHY Board“ is turned on/off according to OUT_GEN_Bit1 setting.

How to test – Setting slave outputs on master (2/2)



ACTION

1. Right click on OUT_GEN_INT1 of the slave node and select „Online Write...” inside the context menu. Change the value from 0 to 50000.



OBSERVATION

1. Brightness of LED2 on „XMC4800 Relax EtherCAT Kit” is dimmed. The OUT_GEN_INT1 value sets the brightness of LED 2. The lower the value the brighter the LED2. To turn off LED2 just set value to max (65535).





Part of your life. Part of tomorrow.

