MB9BFx18_AN706-00075-1v0-E

SPANSION®

FM3

32-BIT MICROCONTROLLER
FM Family Application note

EtherCAT on FM Family Microcontrollers

Revision History

Rev	Date	Remark
0.1	Aug. 22, 2013	CNo, First Edition

This document contains 26 pages.

Table of Contents

Target products

This application note is described about below products;

(TYPE2)

Series	Product Number (not included Package suffix)
MB9BF210	MB9BF216, MB9BF217, MB9BF218
MB9BF610	MB9BF616, MB9BF617, MB9BF618
MB9BFD10	MB9BFD16, MB9BFD17, MB9BFD18

1. Preface

Spansion's microcontroller line FM Family consists of more than 600 derivatives. The high performance line is designed with industrial automation requirements in mind.
Besides controlling sensors and actuators, industrial automation systems need standard communication interfaces to ensure interoperability of equipment from different vendors. FM Family controllers support many of those interfaces, including UART, LIN, CAN, USB, and Ethernet.

Since the 2000s, Ethernet based automation protocols began to supersede the older serial or CAN based Fieldbus protocols as it provides high bandwidth, high noise immunity and long cable lengths.
This application note explains how an important Ethernet based Fieldbus protocol, called EtherCAT, can be used with the Spansion FM3 microcontroller MB9BFD18T.

The software example is based on the free-of-charge software package SSC (Slave Stack Code) by Beckhoff Automation GmbH.

The project was developed on the evaluation board SK-FM3-176PMC-FA but can easily be adapted to other hardware platforms with FM3 or FM4 microcontrollers that feature an MFS (MultiFunctional Serial interface), as an external ASIC is used that ensures the necessary hard real-time requirements.

1 Introduction to EtherCAT

THIS CHAPTER EXPLAINS WHERE TO FIND INFORMATION ON ETHERCAT AND SOME NOTES ABOUT HARDWARE DESIGN

1.1 Some Background Information on EtherCAT

EtherCAT stands for *Ethernet for Controller and Automation Technology* and was published in 2003 by Beckhoff Automation GmbH. Every user must become member of ETG, the EtherCAT Technology Group but membership is free of charge.

EtherCAT is a new Fieldbus concept, which diverges deliberately from the Ethernet standard for reasons of efficiency. The ideas are the following:

- IP, TCP, and UDP are needed for routing through large networks and to multiplex different applications. These functions bring no benefits in cyclic fieldbus communication but add unnecessary overhead. Simple Ethernet frames would suffice in a controlled local network.
- Star topologies need expensive switches and add cabling complexity compared to classic fieldbus technologies, which utilize simple structures such as line and ring.
- A frame that traverses all nodes yields a lower cycle time compared to individual polling.

Beside Ethernet, EtherCAT borrows many concepts from CANopen.

For more information see the dedicated literature such as:

- The ET1100 Hardware Data Sheet to be found online at: http://beckhoff.com/english/download/ethercat_development_products.htm
- Many resources at http://ethercat.org/

1.2 Hardware implementation on SK-FM3-176PMC-FA

EtherCAT is a hard real-time protocol which depends on a dedicated MAC layer. This function cannot be emulated by software and has to be provided by a special integrated circuit. Therefore, not one of the two integrated Ethernet controllers has been used but an external ASIC by Beckhoff Automation GmbH, called ET1100. On SK-FM3-176PMC-FA, it is connected to MFS5.

The EtherCAT ASIC ET1100 is a versatile device, which has to be configured properly to work the intended way. This is done by two means: pin-strapping and programming configuration data into the FRAM.

Pin-strapping is a common technique to configure complex integrated circuits. Different options will be active depending on the voltage level at specified pins while the device is powered on. Pull-up and pull-down resistors must be tied to these pins to define the desired configuration set. Beckhoff offers a convenient tool called “ET1100 Configuration and Pinout” for finding out the correct settings at the aforementioned website.

A special feature has to be kept in mind. The ET1100 requires an electric signal “MII_LINK” which is used to react quickly to link loss in order to set motion controllers into a safe state and to prevent cycling frames. As only few PHYs provide for such a signal, it is recommended to use a signal, which normally is used to drive an LED that is integrated into the RJ45 connector.

For the used PHY KSZ8081 by Micrel Inc., the SPEED LED is suitable for this purpose, unlike the LINK/ACT LED, which would blink.

2 The FM3 software support for EtherCAT ASIC ET1100

THIS CHAPTER EXPLAINS HOW THE SSC IS PORTED TO FM3 MICROCONTROLLER

2.1 **Beckhoff Slave Stack Code (SSC)**

As most EtherCAT related intelligence is contained in the ASIC, the actual application is independent from most communication details. However, to use this ASIC correctly, several thousand lines of code are necessary.

Beckhoff provides a software example called SSC–Slave Stack Code–which implements all needed functionality. The adaption process consists of two modifications:

- The PDI (Process Data Interface) must be matched to the actual microcontroller's hardware
- The Process data must be mapped to EtherCAT's PDOs (Process Data Objects)

The code has to be very generic as the ET1100 can be configured very flexibly and shall run on different hardware platforms. Therefore a lot of configuration parameters are deployed. To speed up development of customer applications, Beckhoff provides a graphical configuration tool, called *SSC Tool*, which checks them for plausibility and generates a suitable sample code, consisting of ANSII C .c and .h files.

The SSC Tool fulfills another important role, as it generates the ESI[1] file, which is necessary to configure crucial settings on the ET1100, such as PDI selection[2].

This section describes the steps necessary to implement the ET1100 support via SPI on FM3. The development consists of four steps:

1. Generate SSC template and ESI file
2. Write the ESI file's contents into the ESC EEPROM
3. Implement low-level PDI access functions on MCU
4. Write the application

On SK-FM3-176PMC-FA, SPI has been used as PDI.

[1] The EtherCAT Slave Configuration file's role is described in chapter 3.8.8

[2] Without ESI file, SPI cannot be used and thus the MCU cannot communicate with the ASIC!

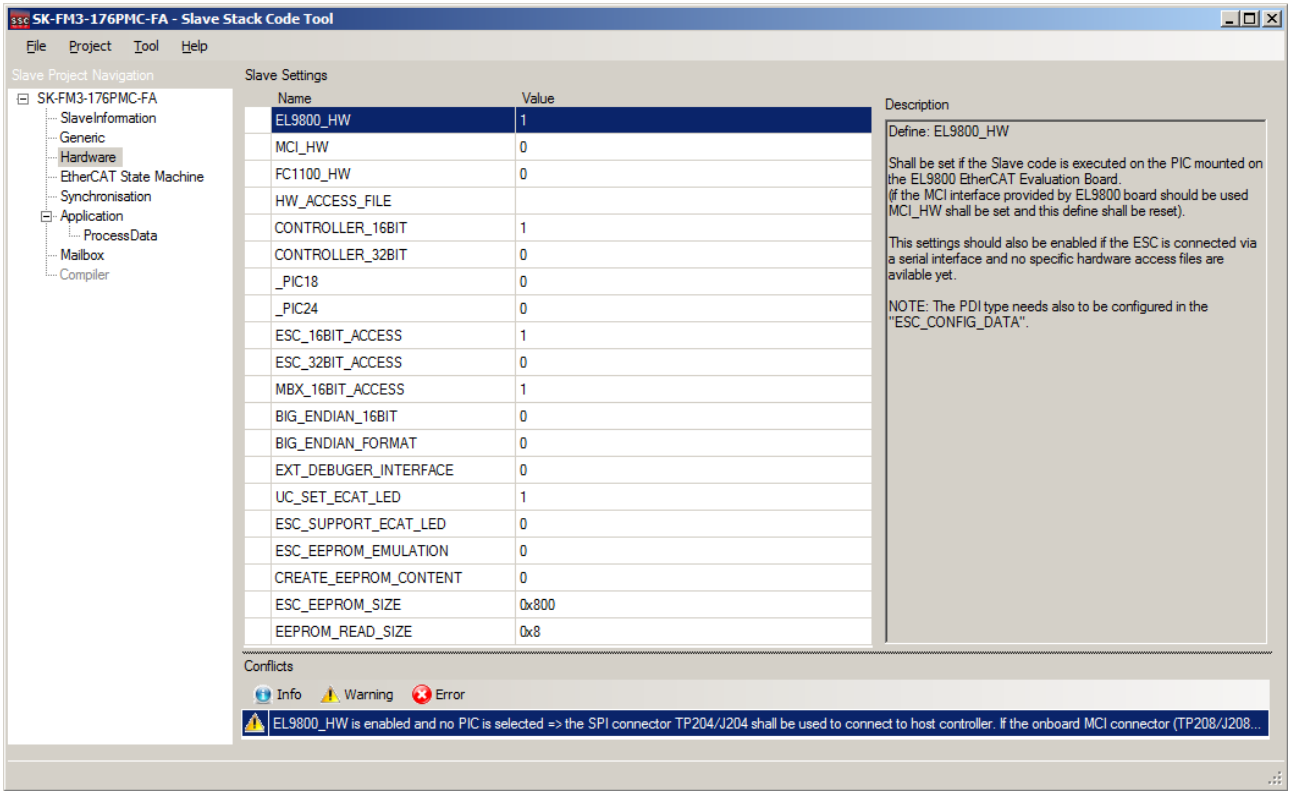


Figure 1: SSC Tool configuration for SK-FM3-176PMC firmware

To generate sample code with SPI interface, the option *EL9800_HW* must be selected in the SSC tool, which is the name of Beckhoff's evaluation board. As the FM3 microcontroller shall be used, the PIC options should be deactivated.

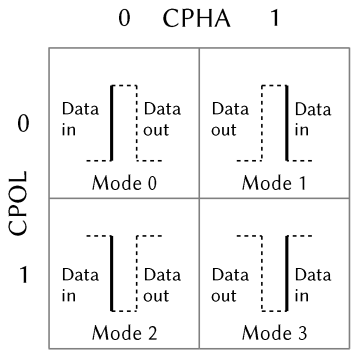


Figure 2 Four SPI modes

SPI is a synchronous serial Master-Slave bus that allows high clock rates (several MHz) and full duplex operation. It is a hardware specification, which is popular because it is free from royalties and patents.

A benefit is the low count of signal lines as it is typical for serial buses[3]. However Motorola did not specify a data transfer protocol with the result that there are everytime three wrong and one right choice when connecting an SPI slave to a master. The differences are simply the time instances when data is produced and sampled.

This is usually called the clock signal's phase (CPHA) and polarity (CPOL), which translate to mode numbers as shown in Figure 2.

3 Four lines: Chip Select, Clock, Master Out Slave In (MOSI) and Master In Slave Out) (MISO)

For successful communication, both parties have to use the identical mode.
Beckhoff offers a comprehensive Excel sheet that calculates configuration settings for the ET1100. The SPI mode can be set as shown in Figure 3.

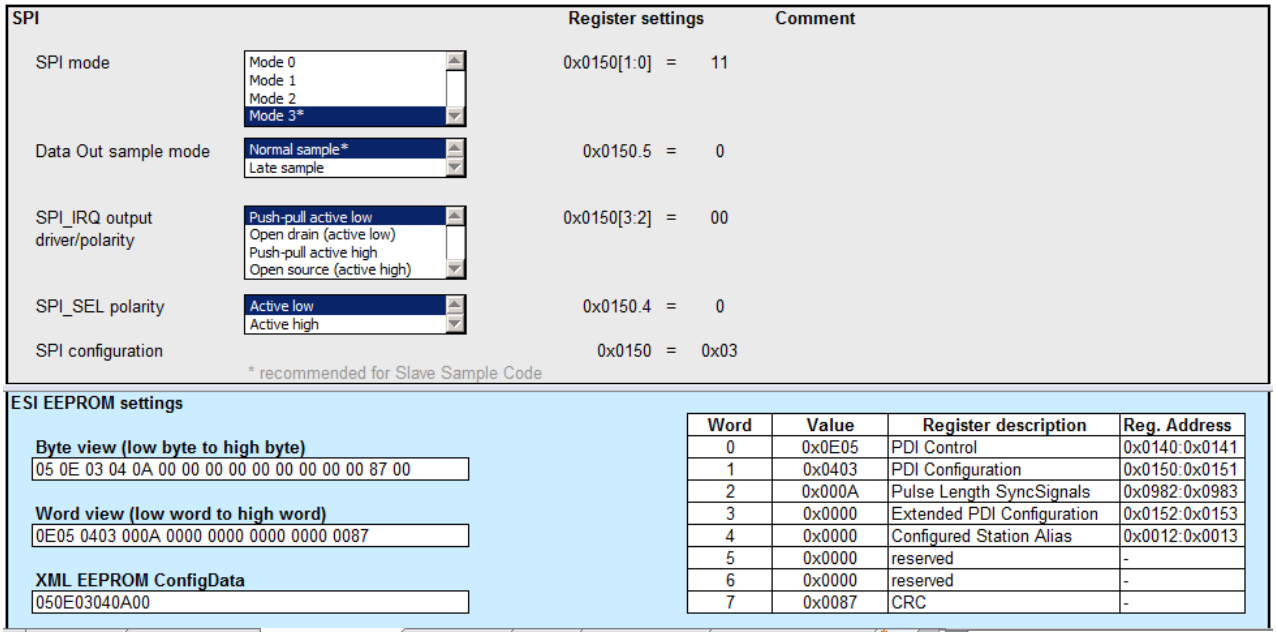


Figure 3: Beckhoff's ET1100 configuration calculator tool: SPI settings

The resulting hexadecimal number (bottom-left part of the screenshot) is reflected in the ESI file's ConfigData container:

```
<Eeprom>
  <ByteSize>32768</ByteSize>
  <ConfigData>050E03040A0000000000</ConfigData>
</Eeprom>
```

Here, ET1100 default mode 3 is kept and the FM3 configuration adapted:

- CPHA = 1: The data changes on the falling edge of the clock and is read/written on the rising edge of the clock.
- CPOL = 1: When the bus is idle, the clock signal is high.

Another important setting, which must be configured in the SSC Tool, is OBJ_DWORD_ALIGN instead of the standard OBJ_WORD_ALIGN as otherwise C structs containing mailbox data and process data objects will be aligned 16-bit-wise, which would be wrong for the FM3, as it is a 32-bit architecture. As a result, the PDO structures and object dictionaries could not be read and consequently the communication fail with error message "Invalid SM IN cfg":

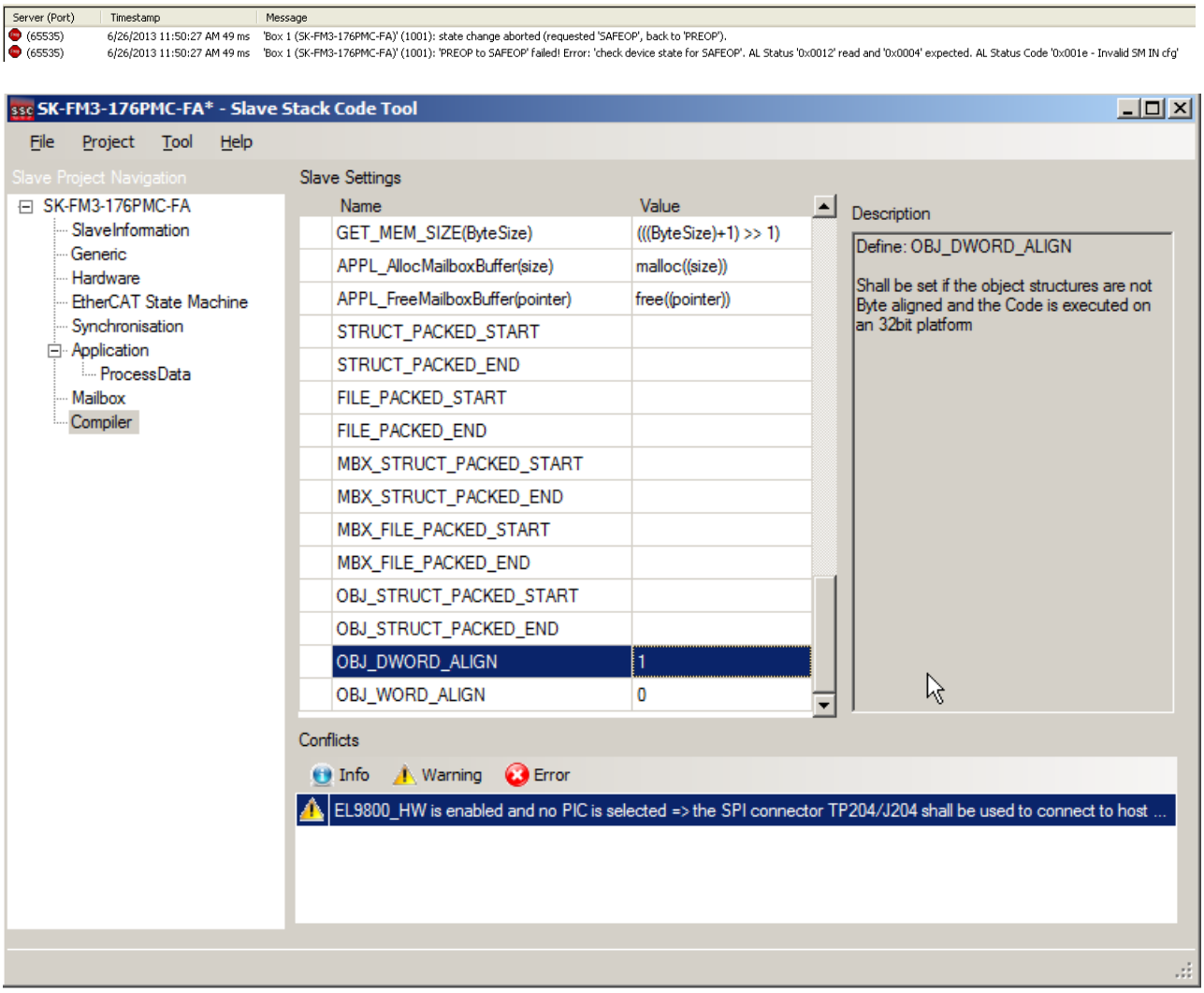


Figure 4: SSC Tool option OBJ_DWORD_ALIGN for 16-bit alignment

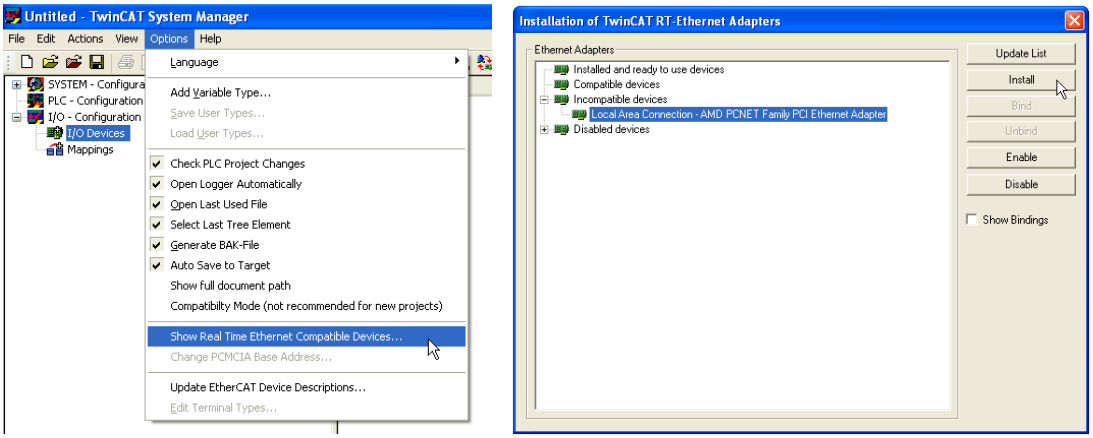
To reach this option, "Show advanced settings" must be activated in Tool -> Options -> Editor.

2.2 Write the ESI file's contents into the ESC EEPROM

As the ASIC is a fully self-contained EtherCAT entity, it is possible to connect the SK-FM3-176PMC-FA to an EtherCAT network without the need of a special firmware on the MCU[4].

To communicate to the slave device, first of all, the PC's Ethernet drivers have to be modified to ensure low-latency operation.

In TwinCAT, please select Options-> "Show Real Time Ethernet Compatible Devices…", then select your Ethernet interface and click "Install":



Now click with your right mouse button on "I/O – Configuration" -> "I/O Devices" and select "Scan Devices":

4 With the exception that ET1100's RESET signal must be deactivated by driving FM3's Pin48/P42 low.

The EtherCAT master software can then detect every connected slave with a function called *Scan for boxes* as indicated in Figure 5. If the EtherCAT configuration EEPROM does not contain any data, it will be indicated with the message box in Figure 6.

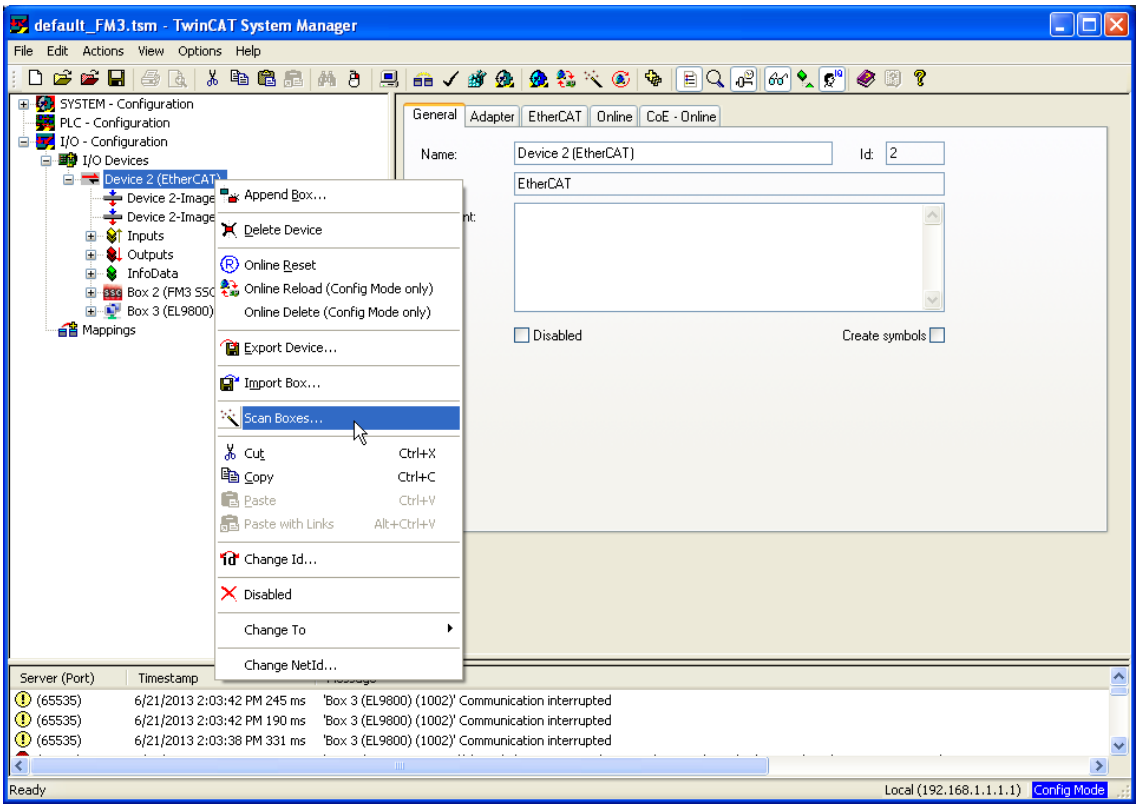


Figure 5: Automatic EtherCAT slave detection

Figure 6: Dialog window indicating empty configuration EEPROM

All this vendor and application specific information is contained in the ESI file, which was created in step 1 by the SSC tool[5]. It can be written into the EEPROM with the PC application TwinCAT by clicking through the GUI elements as shown in Figure 76.

It must be copied into the directory `C:\TwinCAT\Io\EtherCAT` before, in order to be found. TwinCAT should be restarted to rebuild its cache after the copy operation.

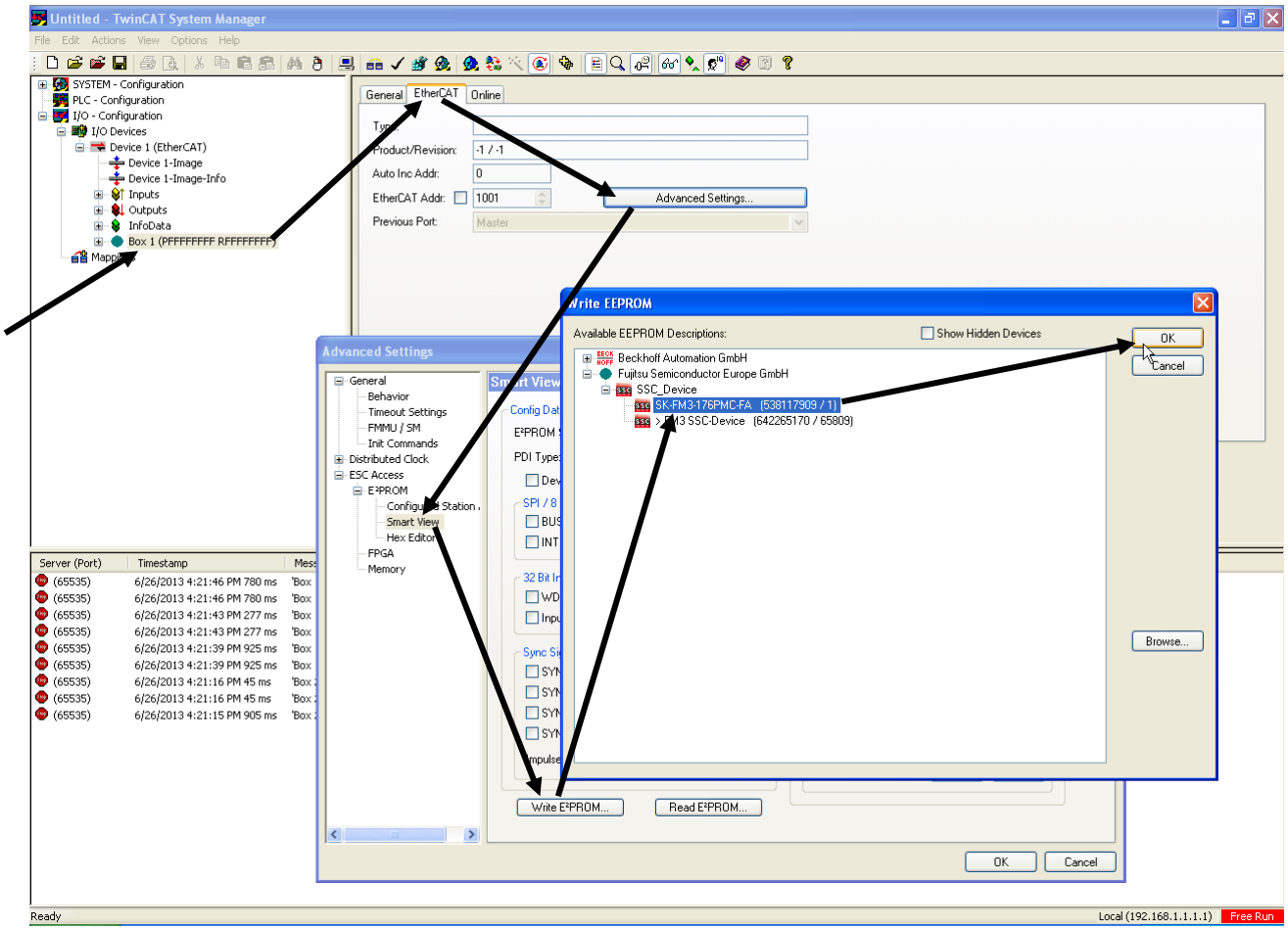


Figure 7: Writing an ESI file into ESC configuration memory with TwinCAT System Manager

After the SK-FM3-176PMC-FA is reset, the ET1100 is configured to use SPI and will be identified as "SK-FM3-176PMC-FA" when the EtherCAT master software performs another "Scan Boxes" run.

[5] Such as `/example/source/EtherCAT/SK-FM3-176PMC-FA.xml` in the prepared example

2.3 Implement Low-Level PDI Access Functions on MCU

Last chapter showed how to configure the ESC to use SPI as Process Data Interface. Now the FM3 part is described. SSC is written in hardware independent ANSI C code and accesses the PDI through functions such as HW_EscWrite() and HW_EscRead(), which have to implement the hardware access.

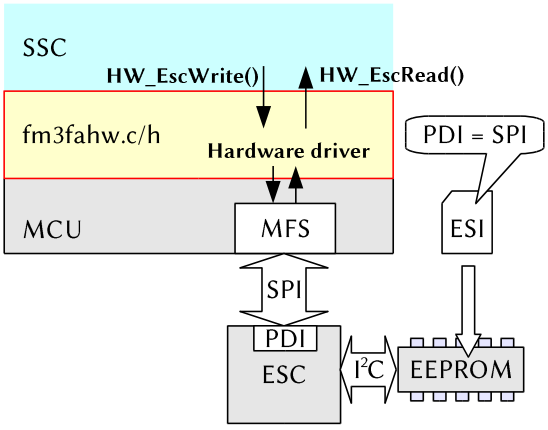


Figure 8 PDI implementation on MCU

There is an example hardware driver included in the Slave Stack Code, however it is not a generic template but a concrete implementation written for the PIC microcontroller from Microchip Inc., so PIC's SPI hardware had to be studied first before the FM3 implementation could be started.

The driver is based on the sample code, just with FM3 specific SPI access routines. This shall be exemplified using the writing function:

```
/**
 \param pData       Pointer to a byte array which holds data to write or saves write data.
 \param Address     EtherCAT ASIC address ( upper limit is 0x1FFF )    for access.
 \param Len           Access size in Bytes.

  \brief  This function operates the SPI write access to the EtherCAT ASIC.
*/
void HW EscWrite( MEM ADDR *pData, UINT16 Address, UINT16 Len )
{
  UINT16 i = Len;
  UINT8 *pTmpData = (UINT8 *)pData;

  if(Len == 0)
  {
    return;
  }

  AddressingEsc( Address, ESC WR );

  /* loop for all bytes to be read */
  while ( i-- > 0 )
  {

    /* get data byte */
    Ecat RWChar(*pTmpData++);

    /* next address */
    Address++;

  }
  DESELECT SPI;
}
```

Function AddressingESC() enables the SPI Chip Select signal, ensures the correct byte order, and sends the two address bytes. During the address phase the ET1100 transmits the AL Event register, which contains the interrupt cause. This information is stored in variable EscALEvent, which other functions in the Slave Stack Code evaluate.

```
/**
 \param Address    EtherCAT ASIC address ( upper limit is 0x1FFF )    for access.
 \param Command    ESC WR performs a write access; ESC RD performs a read access.

 \brief The function addresses the EtherCAT ASIC via SPI for a following SPI access.
*/
static void AddressingEsc( UINT16 Address, UINT8 Command )
{
  UBYTETOWORD tmp;
  uint8 t au8RxBuffer[2];
  uint8 t au8TxBuffer[2];

  tmp.Word = ( Address << 3 ) | Command;

  au8TxBuffer[0] = tmp.Byte[1];
  au8TxBuffer[1] = tmp.Byte[0];

    /* select the SPI (chip select for SPI slave) */
    SELECT SPI;


    /* send two address/command bytes to the ESC */
    Mfs SynchronousTransfer((stc mfsn t*)&ECATMFS,
                         (uint8 t*) &au8TxBuffer,
                         (uint8 t*) &au8RxBuffer,
                         2);

    EscALEvent.Byte[0] = au8RxBuffer[0];
    EscALEvent.Byte[1] = au8RxBuffer[1];

}
```

2.4 Write the application

The SSC is implemented as a module to be used with the standard software framework from chapter **Error! Reference source not found.**. It consists of two tasks, one with high-priority, that is executed every millisecond and the other with low-priority, called every 10 milliseconds. These tasks call a function that maps the local process variables from chapter **Error! Reference source not found.** to the EtherCAT PDOs. This chapter explains how that is done.

For every variable a PDO (Process Data Object) must be created. This must be handcrafted and is explained step-by-step in Application Note ET9300 by Beckhoff.

This section exemplifies this process by mapping global variable u16ADCData to a PDO with address 0x6010. The software maps all variables to EtherCAT PDOs, but are omitted from this description as the steps are identical:

Input variables

- 0x6000: uint32_t u32Heartbeat
- 0x6010: uint16_t u16ADCData
- 0x6020: uint16_t u16DataIn

Output variables

- 0x7010: uint16_t u16DataOut

- 0x7020: uint8_t u8LedRed
- 0x7030: uint8_t u8LedGreen
- 0x7040: uint8_t u8LedBlue

Three files must be adapted: sampleappl.h, sampleappl.c and SK-FM3-176PMC-FA, the ESI file.

The first six changes take place in sampleappl.h.

1: There are two so-called PDO mapping objects, input objects are listed at address 0x1A00, output objects at 0x1601. When we with the ADC value an input PDO is added to the existing one, the size has to be increased to 2:

```
/** \brief 0x1A00 (TxPDO) data structure*/
typedef struct OBJ STRUCT PACKED START {
  UINT16   u16SubIndex0; /**< \brief SubIndex 0*/
  UINT32   aEntries[2]; /**< \brief Entry buffer*/
} OBJ STRUCT PACKED END
TOBJ1A00;
```

2: Next space for the PDO description is reserved:

```
#ifdef  OBJD
/**
 * \brief Object 0x1A00 (TxPDO) entry descriptions
 *
 * SubIndex 0<br>
 * SubIndex 1
*/
OBJCONST TSDOINFOENTRYDESC    OBJMEM asEntryDesc0x1A00[] = {
  {DEFTYPE UNSIGNED8, 0x8, ACCESS READ },
  {DEFTYPE UNSIGNED32, 0x20, ACCESS READ},
  {DEFTYPE UNSIGNED32, 0x20, ACCESS READ} // u16ADCData
};
```

3: The mapping object is extended to accommodate the new PDO:

```
/**
 * \brief Object 0x1A00 (TxPDO) variable to handle object data
 *
 * SubIndex 0 : 1<br>
 * SubIndex 1 : 0x6000.0 32bit
 * SubIndex 2 : 0x6010.0 16bit u16ADCData
 */
PROTO TOBJ1A00 TxPDOMap
#ifdef  SAMPLE APPLICATION
 = {2, {0x60000020, 0x60100020}}
#endif
;
```

4: The PDO name is stored in the Object Dictionary, so the master can read it out later:

```
OBJCONST TSDOINFOENTRYDESC    OBJMEM EntryDesc0x6010 = {DEFTYPE UNSIGNED32, 0x20,
                                                        ACCESS READ | OBJACCESS TXPDOMAPPING};

/** \brief Object 0x6010 (u16ADCData) name*/
OBJCONST UCHAR OBJMEM aName0x6010[] = "Variable Resistor RP1";
#endif //#ifdef  OBJD

/// \brief Object 0x6010 (u16ADCData) variable*/
/// imported from tasks.h
;
```

5: Now a new entry in the Object Dictionary, known from CANopen in chapter **Error!**

Reference source not found., must be written:

```
/**
 * \brief Object dictionary related to the Sample Application
 *
 * For every application related object an object description is added to the dictionary
 */
/*if  PIC18 is enabled the object dictionary is fixed defined in coeappl.c*/
TOBJECT    OBJMEM ApplicationObjDic[] = {
   /* Object 0x1601 */
   {NULL,NULL,  0x1601, {DEFTYPE PDOMAPPING, 1 | (OBJCODE REC << 8)}, asEntryDesc0x1601,
aName0x1601, &RxPDOMap, NULL, NULL, 0x0000 },
   /* Object 0x1A00 */
   {NULL,NULL,   0x1A00, {DEFTYPE PDOMAPPING, 2 | (OBJCODE REC << 8)}, asEntryDesc0x1A00,
aName0x1A00, &TxPDOMap, NULL, NULL, 0x0000 },
    /* Object 0x1C12 */
   {NULL,NULL,   0x1C12, {DEFTYPE UNSIGNED16, 1 | (OBJCODE ARR << 8)}, asPDOAssignEntryDesc,
aName0x1C12, &sRxPDOassign, NULL, NULL, 0x0000 },
     /* Object 0x1C13 */
   {NULL,NULL,   0x1C13, {DEFTYPE UNSIGNED16, 1 | (OBJCODE ARR << 8)}, asPDOAssignEntryDesc,
aName0x1C13, &sTxPDOassign, NULL, NULL, 0x0000 },
   /* Object 0x6000 */
   {NULL,NULL,   0x6000, {DEFTYPE UNSIGNED32, 0 | (OBJCODE VAR << 8)}, &EntryDesc0x6000,
aName0x6000, &InputCounter, NULL, NULL, 0x0000 },

   /* Object 0x6010 */
   {NULL,NULL,   0x6010, {DEFTYPE UNSIGNED16, 0 | (OBJCODE VAR << 8)}, &EntryDesc0x6010,
aName0x6010, &u16ADCData, NULL, NULL, 0x0000 },


   /* Object 0x7010 */
   {NULL,NULL,   0x7010, {DEFTYPE_UNSIGNED32, 0 | (OBJCODE_VAR << 8)}, &EntryDesc0x7010,
aName0x7010, &OutputCounter, NULL, NULL, 0x0000 },
   {NULL,NULL, 0xFFFF, {0, 0}, NULL, NULL, NULL, NULL}};
#endif
```

6: In sampleappl.c there are two process data mapping functions that actually copy process data between MCU internal variables and EtherCAT-PDOs. As an example this is how the Input process data mapping function must look like to include the new values as well:

```
/**
\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,&InputCounter,SIZEOF(InputCounter));

    pData += 2;
    MEMCPY(pData,&u16ADCData,SIZEOF(u16ADCData));
}
```

7: Next the ESI file must be adjusted to match the altered PDO structure in the XML description:

```
<TxPdo Mandatory="true" Fixed="true" Sm="3">
     <Index>#x1a00</Index>
     <Name>TXPDO</Name>
     <Entry>
          <Index>#x6000</Index>
          <SubIndex>0</SubIndex>
          <BitLen>32</BitLen>
          <Name>32Bit Input</Name>
          <DataType>UDINT</DataType>
     </Entry>
     <Entry>
          <Index>#x6010</Index>
          <SubIndex>0</SubIndex>
          <BitLen>32</BitLen>
          <Name>Variable Resistor RP1</Name>
          <DataType>UDINT</DataType>
     </Entry>
</TxPdo>
```

8: Lastly the SyncManager data count size has to match the amount of input data:

```
<Sm DefaultSize="8" StartAddress="#x1C00" ControlByte="#x20" Enable="1">Inputs</Sm>
```

For a thorough description of the SSC code and more helpful advice, refer to the dedicated documentation by Beckhoff, or ETG respectively. Especially:

- ETG2200: Slave Implementation Guide
- AN_ET9300: EtherCAT Slave Stack Code

3 How to Use the Demonstration

THIS CHAPTER EXPLAINS HOW THE ETHERCAT EXAMPLE CODE CAN BE RUN ON SK-FM3-176PMC-FA WITH BECKHOFF TWINCAT SOFTWARE

The product finder on ETG website lists 82 EtherCAT master options, with 22 being PC software available for several operating systems and by different vendors.
Among them are two free open-source projects:

- SOEM (Simple Open EtherCAT master): http://soem.berlios.de/
- IgH EtherCAT-Master for Linux: http://www.etherlab.org/en/ethercat/index.php

In case of Microsoft Windows special real-time capable drivers must be installed on the master computer before it can be used to control an EtherCAT network. The Linux based systems are either implement the master itself as a kernel module or require a Linux with real-time extension.
However TwinCAT from Beckhoff has been selected for this chapter, because it is convenient to use and is necessary for programming the configuration anyway.
Firstly the EtherCAT slaves must detected by the Master and programmed with the appropriate ESI as explained in section 2.2.

As shown in

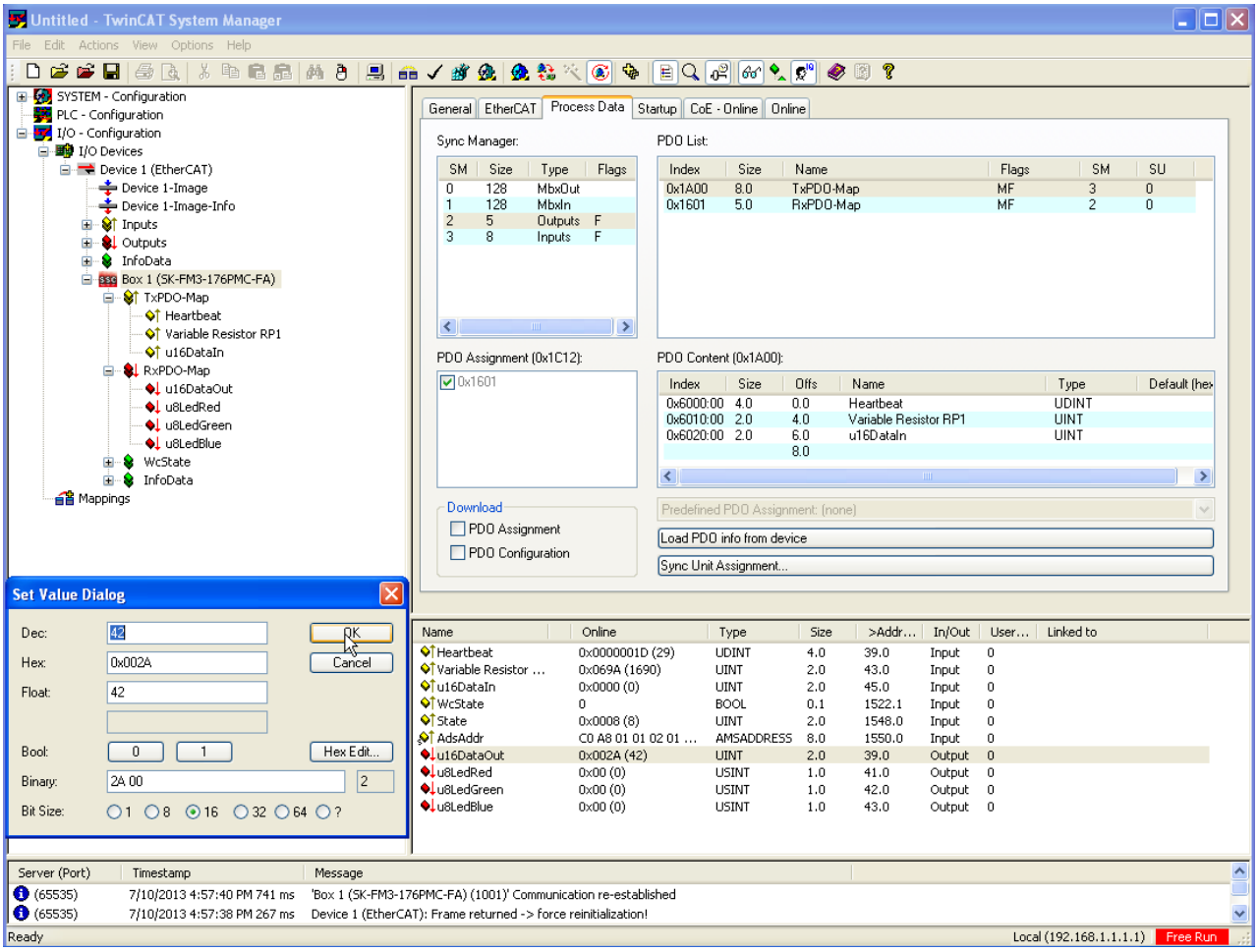


Figure 9,

1. The SK-FM3-176PMC-FA is detected
2. The PDOs are loaded from ESI file
3. The EtherCAT slave is set into RUN mode
4. Input PDOs can be watched life
5. Output PDOs can be written by right clicking them, selecting “Online Write” and using the “Set Value Dialog”

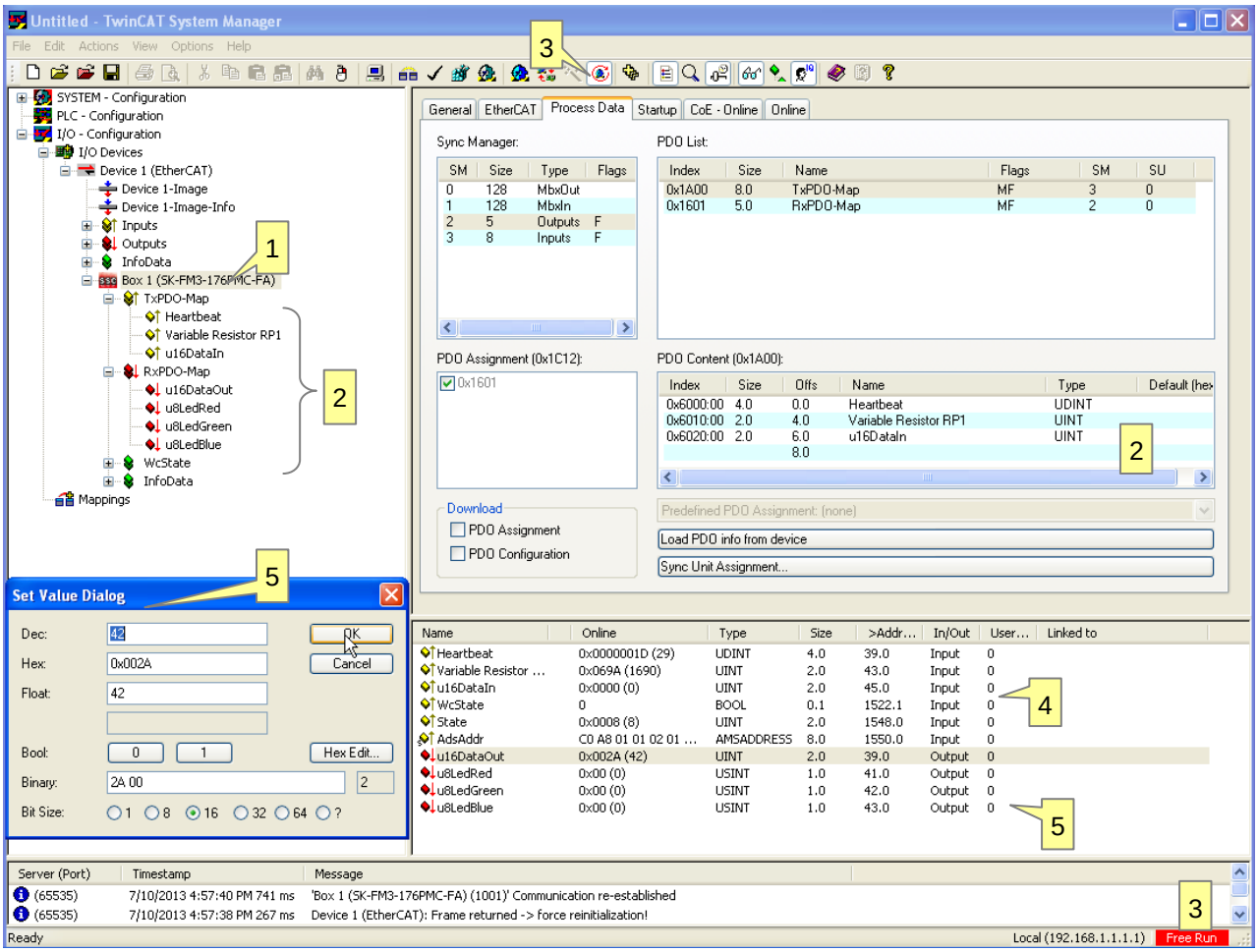


Figure 9: Process Data Objects in TwinCAT

EtherCAT's kinship with CANopen can be seen in the “CoE - Online” tab as shown in

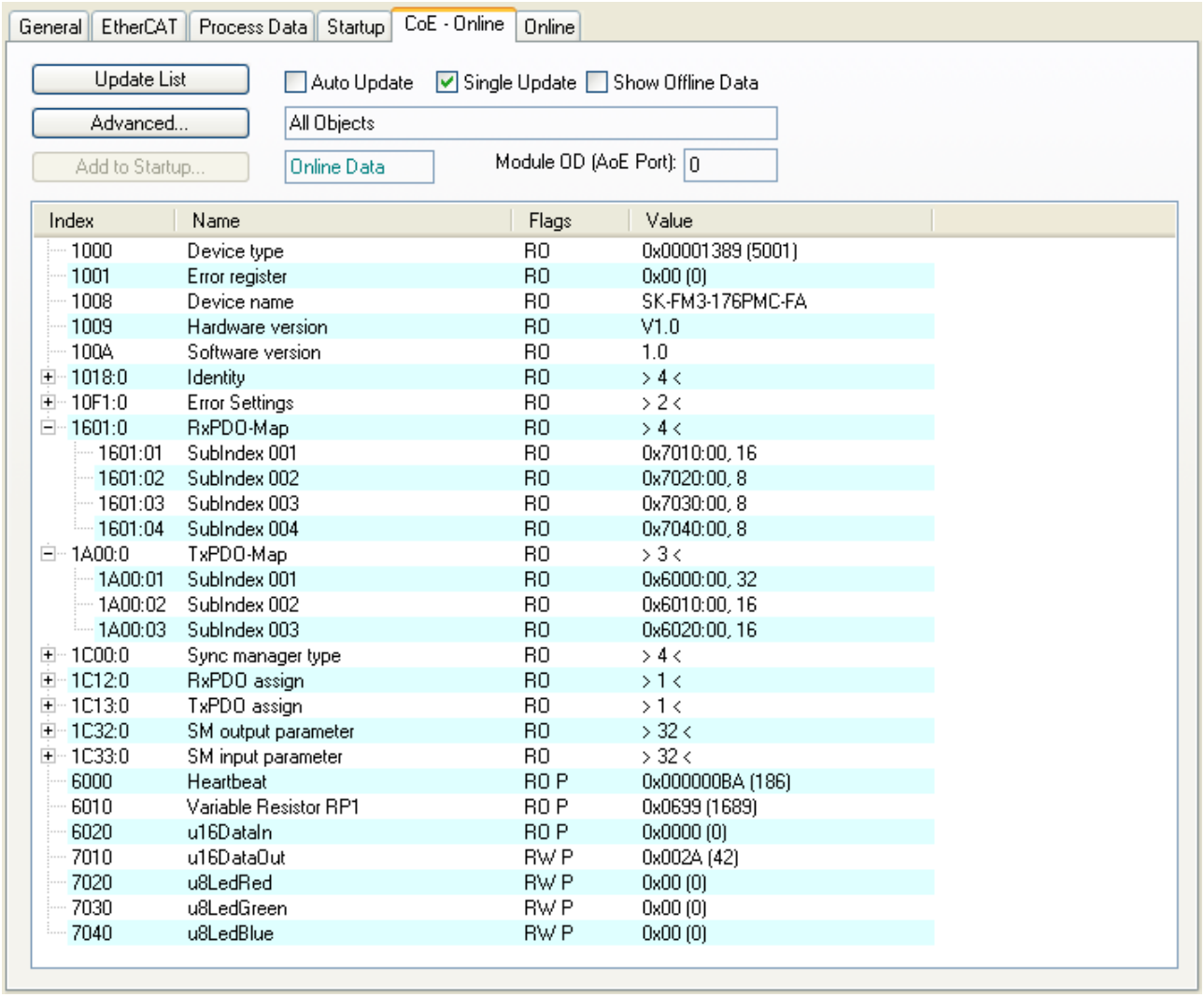
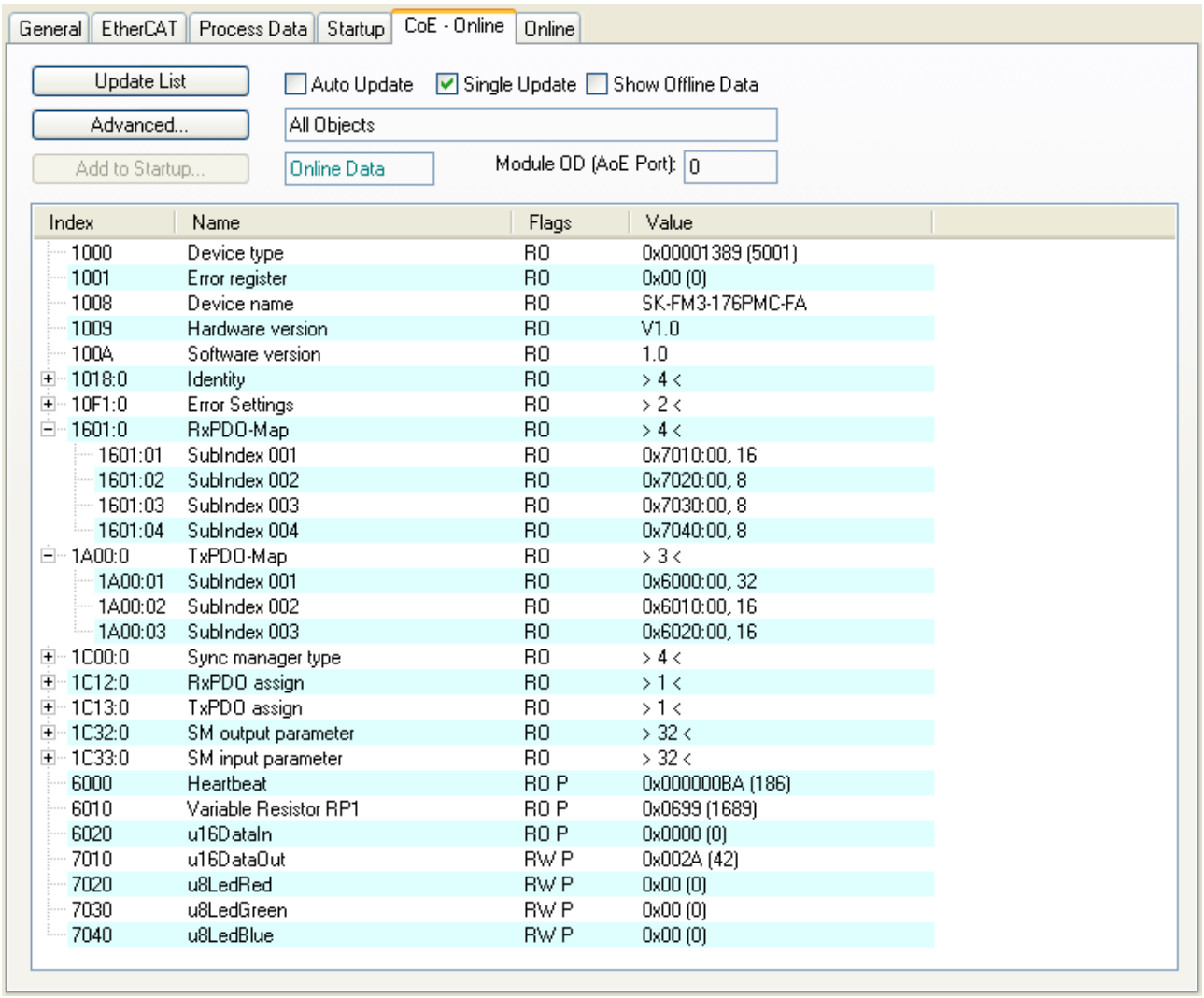


Figure 10. The Object Dictionary contains EtherCAT specific information like Device Name, SyncManager properties and so forth, which have been configured in the ESI file, but also all PDOs and mapping objects, which have been set in sampleappl.c.



Index	Name	Flags	Value
1000	Device type	RO	0x00001389 (5001)
1001	Error register	RO	0x00 (0)
1008	Device name	RO	SK-FM3-176PMC-FA
1009	Hardware version	RO	V1.0
100A	Software version	RO	1.0
1018:0	Identity	RO	> 4 <
10F1:0	Error Settings	RO	> 2 <
1601:0	RxPDO-Map	RO	> 4 <
1601:01	SubIndex 001	RO	0x7010:00, 16
1601:02	SubIndex 002	RO	0x7020:00, 8
1601:03	SubIndex 003	RO	0x7030:00, 8
1601:04	SubIndex 004	RO	0x7040:00, 8
1A00:0	TxPDO-Map	RO	> 3 <
1A00:01	SubIndex 001	RO	0x6000:00, 32
1A00:02	SubIndex 002	RO	0x6010:00, 16
1A00:03	SubIndex 003	RO	0x6020:00, 16
1C00:0	Sync manager type	RO	> 4 <
1C12:0	RxPDO assign	RO	> 1 <
1C13:0	TxPDO assign	RO	> 1 <
1C32:0	SM output parameter	RO	> 32 <
1C33:0	SM input parameter	RO	> 32 <
6000	Heartbeat	RO P	0x000000BA (186)
6010	Variable Resistor RP1	RO P	0x0699 (1689)
6020	u16DataIn	RO P	0x0000 (0)
7010	u16DataOut	RW P	0x002A (42)
7020	u8LedRed	RW P	0x00 (0)
7030	u8LedGreen	RW P	0x00 (0)
7040	u8LedBlue	RW P	0x00 (0)

Figure 10: The Object Dictionary in TwinCAT's CoE view

4 Final Thoughts on EtherCAT

- EtherCAT is an efficient real-time fieldbus that uses Ethernet cables and is controlled by a master station that uses a standard network interface card only.
- However it is not Ethernet, as the modifications are very strong, therefore a dedicated segment must be used.
- As the complete communication is encapsulated in an external chip, compatibility is inherently given, which would not be the case in software based solutions.
- Compared to MODBUS TCP, the Object Dictionary and Electronic Datasheets are very convenient and reduce configuration errors later for the user.

5 Additional Information

More information about Spansion's Microcontrollers can be found on the Internet at
http://www.spansion.com/products/microcontrollers/Pages/default.aspx

The software example related to this application note is:
mb9bfxxx_mfs_ethercat_ssc-v10.zip