AX58101(2) Software API Design Guide

Revision 1.00
September 16th, 2025

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

Revision History

Revision	Date	Description
1.00	2025/09/16	Preliminary release

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

Table of Contents

1. Introduction 8

2. AX58101(2) Overview 8

2-1. Block Diagram 9
2-2. Typical Applications 10
2-3. ESC/Function Registers Memory Map 11

3. EtherCAT SSC Overview 14

3-1. Code Structure 14
3-2. Execution Structure 16
3-3. Interrupt Handling 17
3-4. Process Data Handling 18

4. SSC Hardware Access Implementation 20

4-1. Interrupt Handler Functions 23
4-1-1. ECAT_CheckTimer 23
4-1-2. PDI_Isr 23
4-1-3. Sync0_Isr 23
4-1-4. Sync1_Isr 23
4-2. Interface Functions/Marcos 24
4-2-1. HW_Init 24
4-2-2. HW_Release 24
4-2-3. HW_GetALEventRegister 24
4-2-4. HW_GetALEventRegister_Isr 24
4-2-5. HW_ResetALEventMask 24
4-2-6. HW_SetALEventMask 25
4-2-7. HW_SetLed 25
4-2-8. HW_RestartTarget 25
4-2-9. HW_DisableSyncManChannel 25
4-2-10. HW_EnableSyncManChannel 25
4-2-11. HW_GetSyncMan 26
4-2-12. HW_GetTimer 26
4-2-13. HW_ClearTimer 26
4-2-14. HW_EepromReload 26
4-3. Read Access Functions 27
4-3-1. HW_EscRead 27
4-3-2. HW_EscReadIsr 27
4-3-3. HW_EscReadDWord 27
4-3-4. HW_EscReadDWordIsr 28

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

4-3-5. HW_EscReadWord 28
4-3-6. HW_EscReadWordIsr 28
4-3-7. HW_EscReadByte 28
4-3-8. HW_EscReadByteIsr 29
4-3-9. HW_EscReadMbxMem 29
4-4. Write Access Functions 30
4-4-1. HW_EscWrite 30
4-4-2. HW_EscWriteIsr 30
4-4-3. HW_EscWriteDWord 30
4-4-4. HW_EscWriteDWordIsr 31
4-4-5. HW_EscWriteWord 31
4-4-6. HW_EscWriteWordIsr 31
4-4-7. HW_EscWriteByte 31
4-4-8. HW_EscWriteByteIsr 32
4-4-9. HW_EscWriteMbxMem 32
5. AX58101(2) Extended Function Driver API 33
5-1. MISC Module 34
5-2-1. MISC Module Definition/Macro Description 35
5-2-1-1. MISC_GET_HOST_INTF_STATUS 35
5-2-1-2. MISC_GET_INTR_CONTROL 35
5-2-1-3. MISC_GET_INTR_STATUS 35
5-2-1-4. MISC_CLR_INTR_STATUS 35
5-2-2. MISC Module Function Description 35
5-2-2-1. MISC_InterruptsEnable 35
5-2-2-2. MISC_InterruptsDisable 35
5-2-2-3. MISC_GetIRQ 36
5-2-2-4. MISC_EfuseRead 36
5-2-2-5. MISC_MotorControlOverride 36
5-2-2-6. MISC_SpiMasterOverride 36
5-2-2-7. MISC_ForceFunctionActive 36
5-2. IOWD Module 37
5-3-1. IOWD Module Definition/Macro Description 38
5-3-1-1. IOWDEM_OBJECT 38
5-3-2. IOWD Module Function Description 39
5-3-2-1. IOWDEM_Init 39
5-3-2-2. IOWDEM_DeInit 39
5-3-2-3. IOWD_Start 39
5-3-2-4. IOWD_Stop 39
5-3-2-5. IOWD_SetTimeout 39
5-3-2-6. IOWD_GetTimePeakValue 39
5-3-2-7. IOWD_ClearTimerPeakValue 40
5-3-2-8. IOWD_UnlockIO 40
5-3-2-9. EM_Start 40
5-3-2-10. EM_Stop 40

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-3-2-11. EM_UnlockIO *40*
5-3. SPIM Module 41
5-4-1. SPIM Module Definition/Macro Description *42*
5-4-1-1. SPIM_STATE *42*
5-4-1-2. SPIM_STATUS *42*
5-4-1-3. SPIM_CLK_MODE *43*
5-4-1-4. SPIM_POLARITY *43*
5-4-1-5. SPIM_SLAVE_ENABLE *43*
5-4-1-6. SPIM_SLAVE_ID *44*
5-4-1-7. SPIM_CHANNELS *44*
5-4-1-8. SPIM_CFG_OBJECT *45*
5-4-1-9. SPIM_SLAVE_INFO *46*
5-4-1-10. SPIM_OBJECT *46*
5-4-2. SPIM Module Function Description *47*
5-4-2-1. SPIM_Init *47*
5-4-2-2. SPIM_DeInit *47*
5-4-2-3. SPIM_Config *47*
5-4-2-4. SPIM_SetRxStartByte *47*
5-4-2-5. SPIM_ClearInterrupt *47*
5-4-2-6. SPIM_OneShotStart *48*
5-4-2-7. SPIM_ContinueStart *48*
5-4-2-8. SPIM_ContinueStop *48*
5-4-2-9. SPIM_ReadyLatchPulseCtrl *48*
5-4-2-10. SPIM_ClearSlaveAllocation *48*
5-4-2-11. SPIM_SwitchSlave *49*
5-4-2-12. SPIM_WriteTxBuf *49*
5-4-2-13. SPIM_Start *49*
5-4-2-14. SPIM_CheckComplete *49*
5-4-2-15. SPIM_ReadRxBuf *50*
5-4-2-16. SPIM_TransmitReceive *50*
5-4. MC Module 51
5-5-1. MC Module Definition/Macro Description *53*
5-5-2. PWM Module Definition/Macro Description *53*
5-5-2-1. PWM_Polarity *53*
5-5-2-2. PWM_AlignMode *53*
5-5-2-3. PWM_Channels *53*
5-5-2-4. PWM_OBJECT *53*
5-5-3. STEP Module Definition/Macro Description *54*
5-5-3-1. STEP_OBJECT *54*
5-5-4. PWM Mode Function Description *54*
5-5-4-1. PWM_Init *54*
5-5-4-2. PWM_DeInit *55*
5-5-4-3. PWM_SetHighPulseWidth *55*
5-5-4-4. PWM_SetAPulsePosition *55*
5-5-4-5. PWM_Driving *55*
5-5-4-6. PWM_H_Output *55*
5-5-4-7. PWM_L_Output *56*

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-5-4-8. PWM_A_Output 56
5-5-4-9. PWM_B_Output 56
5-5-4-10. PWM_C_Output 56
5-5-4-11. PWM_Z_Output 56
5-5-4-12. PWM_AB_Output 56
5-5-4-13. PWM_FuncEnable 57
5-5-4-14. PWM_FuncDisable 57
5-5-4-15. PWM_StopAllOutput 57
5-5-4-16. PWM_H_Polarity 57
5-5-4-17. PWM_L_Polarity 57
5-5-5. STEP Mode Function Description 58
5-5-5-1. STEP_Init 58
5-5-5-2. STEP_DeInit 58
5-5-5-3. STEP_Start 58
5-5-5-4. STEP_CheckCompleted 58
5-5-5-5. STEP_ClearStepCounter 58
5-5-5-6. STEP_GetStepCounter 59
5-5-5-7. STEP_StartFreeRun 59
5-5-5-8. STEP_StopFreeRun 59
5-5. ENC Module 60
5-6-1. ENC Module Definition/Macro Description 61
5-6-2. ABZ Module Definition/Macro Description 61
5-6-2-1. ENC_ABZ_RATIO 61
5-6-2-2. ENC_CLR_EVENT_AT_IZ_EDGE 61
5-6-2-3. ENC_ABZ_OBJECT 61
5-6-3. CW/CCW Module Definition/Macro Description 62
5-6-3-1. ENC_CWCCW_OBJECT 62
5-6-4. CLK/DIR Module Definition/Macro Description 62
5-6-4-1. ENC_CLKDIR_OBJECT 62
5-6-5. ABZ Mode Function Description 63
5-6-5-1. ENC_ABZ_Init 63
5-6-5-2. ENC_ABZ_DeInit 63
5-6-6. CW/CCW Mode Function Description 63
5-6-6-1. ENC_CWCCW_Init 63
5-6-6-2. ENC_CWCCW_DeInit 63
5-6-7. CLKDIR Mode Function Description 64
5-6-7-1. ENC_CLKDIR_Init 64
5-6-7-2. ENC_CLKDIR_DeInit 64
5-6-8. HALL Mode Function Description 64
5-6-8-1. ENC_HALL_Init 64
5-6-8-2. ENC_HALL_DeInit 64
5-6-8-3. ENC_HALL_GetStatus 64
5-6-8-4. ENC_HALL_GetPhaseDuration 65
5-6-8-5. ENC_HALL_ClearOverrun 65
5-6-9. ENC General Function Description 65
5-6-9-1. ENC_GetLatchedValue 65
5-6-9-2. ENC_GetCounter 65

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-6-9-3. ENC_ClearCounter 65
Appendix A. PHY Enhancement Solution 66

List of Figures

Figure 2-1. AX58101(2) Features 8
Figure 2-2. AX58101(2) Block Diagram 9
Figure 2-3. AX58101(2) Typical Applications 10
Figure 2-4. Function Registers Mirror Mapping Table 13
Figure 3-1. EtherCAT SSC Code Structure 14
Figure 3-2. EtherCAT SSC Files Association 15
Figure 3-3. Basic SSC Execution Structure 16
Figure 3-4. SSC MainLoop Execution 16
Figure 3-5. ESC Interrupts Basics 17
Figure 3-6. PDO Output Mapping Example 18
Figure 3-7. PDO Input Mapping Example 19
Figure 4-1. SSC Hardware Related Defines 20
Figure 4-2. SSC uC Hardware Configuration Examples 21
Figure 4-3. SSC Hardware Functions Schema 22
Figure 5-1. AX58101(2) Extended Function Driver APIs 33
Figure 5-2. MISC Module Block Diagram 34
Figure 5-3. IOWD Module Block Diagram 37
Figure 5-4. SPIM Module Block Diagram 41
Figure 5-5. MC Module Block Diagram 52
Figure 5-6. ENC Module Block Diagram 60

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

1. Introduction

This document describes the EtherCAT SubDevice Stack Code (SSC) software API to assist developers to porting their EtherCAT SubDevice stack applications with AX58101(2) EtherCAT SubDevice Controller (ESC).

2. AX58101(2) Overview

The AX58101(2) integrates two embedded Fast Ethernet PHYs which can support both copper and fiber industrial Ethernet applications and supports some additional interfaces such as Pulse Width Modulation (PWM), Incremental (ABZ)/Hall Encoder, SPI master, 32 Digital I/O, Emergency Stop Input, etc. for designers to easily implement AX58101(2) on different EtherCAT industrial fieldbus applications without extra microcontroller.

The AX58101(2) provides SPI slave and Local bus Process Data Interfaces (PDI) to provide an easy way for system designers to implement the standard EtherCAT communication functionalities on those traditional non-EtherCAT MCU and DSP industrial platforms.

Part No.	Fast Ethernet	FMMU	Sync Managers	RAM (Kbytes)	Distributed Clock	Digital I/O	SPI Slave
AX58101(2)	2 x Internal PHY 2 x MII	8	8	9	64-bits	32(64)	Yes
	Local Bus	**SPI Master**	**PWM**	**ABZ/Hall Encoder**	**Emergency Stop Input**	**Package**	**Temperature Range (℃)**
	8/16bit Aync.	Yes	3 Channel	Yes	Yes	LQFP-80(128)	-40 ~+105

Figure 2-1. AX58101(2) Features

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

2-1. Block Diagram

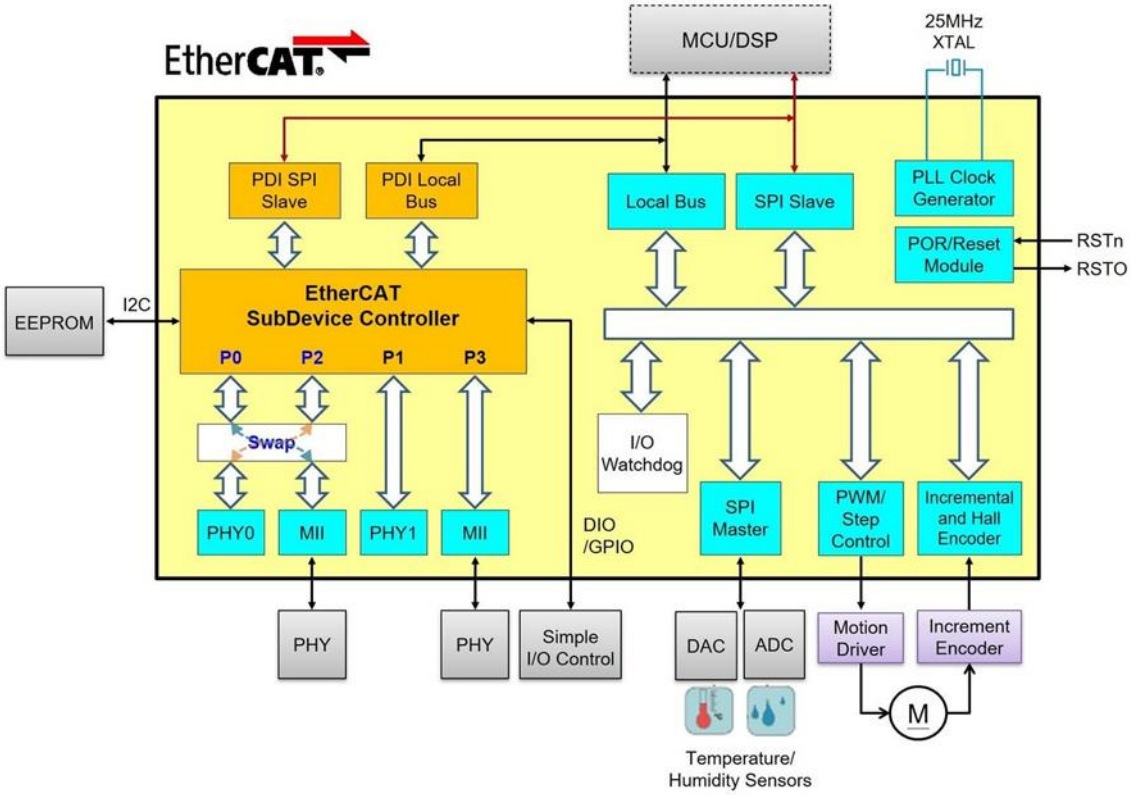

Figure 2-2. AX58101(2) Block Diagram

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

2-2. Typical Applications

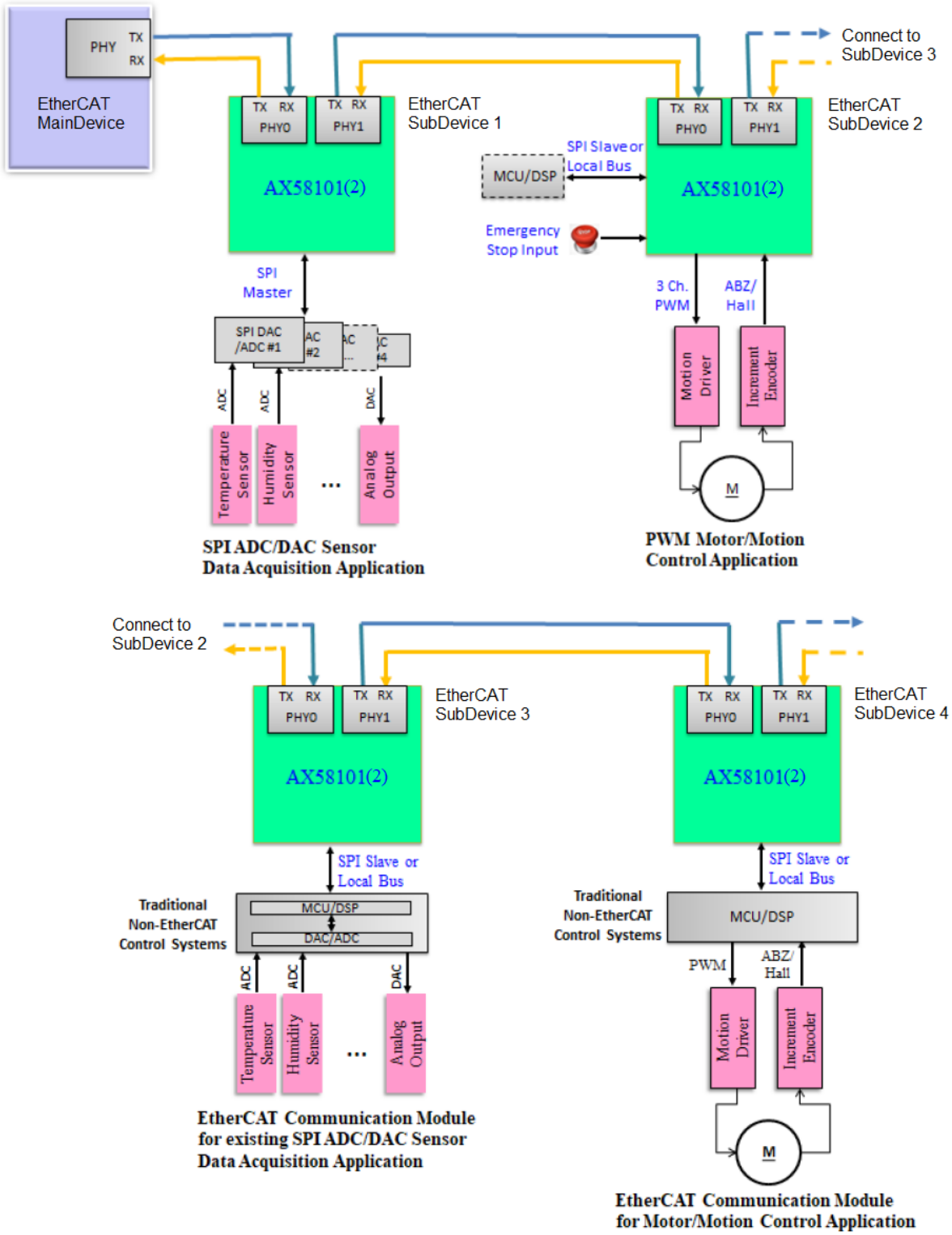

Figure 2-3. AX58101(2) Typical Applications

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

2-3. ESC/Function Registers Memory Map

Please refer to AX58101(2) datasheet for details.

Function Address	Name	Description
0x000	MCTLR	Motor Control Register
0x002	PXCFGR	PWM Pulse X Configure Register
0x004	PTAPPR	PWM Trigger A Pulse Position Register
0x006	PTBPPR	PWM Trigger B Pulse Position Register
0x008	PPCR	PWM Period Cycle Register
0x00A	PBBMR	PWM Pulse Break Before Make Register
0x00C	P1CTRLR	PWM1Control Register
0x00E	P1SHR	PWM1 Counter Shift Register
0x010	P1HPWR	PWM1 High Pulse Width Register
0x012	P2CTRLR	PWM2 Control Register
0x014	P2SHR	PWM2 Shift Register
0x016	P2HPWR	PWM2 High Pulse Width Register
0x018	P3CTRLR	PWM3 Control Register
0x01A	P3SHR	PWM3 Counter Shift Register
0x01C	P3HPWR	PWM3 High Pulse Width Register
0x020	SGTLR	Step Gap Time Low Register
0x022	SGTHR	Step Gap Time High Register
0x024	SHPWR	Step High Pulse Width Register
0x026	TDLYR	Transform Delay step Register
0x028	STNLR	Step Target Number Low Word Register
0x02A	STNHR	Step Target Number High Word Register
0x02C	SCFGR	Step Configure Register
0x02E	SCTRLR	Step Control Register
0x030	SCNTLR	Step Counter Content Low Register
0x032	SCNTHR	Step Counter Content High Register
0x034	STPSR	Step Status Register
0x038	SDGTR	Step Delta Gap Time Register
0x03A	SRMR	Step Roll Motion Register
0x040	ECNTVLR	Encoder Counter Value Low Register
0x042	ECNTVHR	Encoder Counter Value High Register
0x044	ECNSTLR	Encoder Constant Low Register
0x046	ECNSTHR	Encoder Constant High Register
0x048	ELATLR	Encoder Latched Low Register
0x04A	ELATHR	Encoder Latched High Register
0x04C	EMODR	Encoder Mode Configuration Register
0x04E	ECLRR	Encoder Clear Register
0x050	HALSTR	Hall Sensing State Register
0x52	EFLR	Encoder Filter Length Register
0x54	RTVLR	Reset Threshold Value Low Register
0x56	RTVHR	Reset Threshold Value High Register

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

0x58	LPCLR	Last Phase Counter Latch Low Register
0x5A	LPCHR	Last Phase Counter Latch High Register
0x5C	LZCLLR	Last Z to Z(phase_change) Counter Latch Low Register
0x5E	LZCLHR	Last Z to Z(phase_change) Counter Latch High Register
0x060	WTLR	Watchdog Timer Low Register
0x062	WTHR	Watchdog Timer High Register
0x064	WCFGR	Watchdog Configure Register
0x066	WTPVCR	Watchdog Timer Peak Value Clear Register
0x068	WMP0R	Watchdog Monitored Polarity 0 Register
0x06A	WMP1R	Watchdog Monitored Polarity 1 Register
0x06C	WMK0R	Watchdog Monitored Mask 0 Register
0x06E	WMK1R	Watchdog Monitored Mask 1 Register
0x070	WOM0R	Watchdog Monitored Output Mask 0 Register
0x072	WOM1R	Watchdog Monitored Output Mask 1 Register
0x074	WOE0R	Watchdog Monitored Output Enable 0 Register
0x076	WOE1R	Watchdog Monitored Output Enable 1 Register
0x078	WOP0R	Watchdog Monitored Output Polarity 0 Register
0x07A	WOP1R	Watchdog Monitored Output Polarity 1 Register
0x07C	WTPVLR	Watchdog Timer Peak Value Low Register
0x07E	WTPVHR	Watchdog Timer Peak Value High Register
0x140	WMP2R	Watchdog Monitored Polarity 2 Register
0x142	WMP3R	Watchdog Monitored Polarity 3 Register
0x144	WMK2R	Watchdog Monitored Mask 2 Register
0x146	WMK3R	Watchdog Monitored Mask 3 Register
0x148	WOM2R	Watchdog Output Mask 2 Register
0x14A	WOM3R	Watchdog Output Mask 3 Register
0x14C	WOE2R	Watchdog Output Enable 2 Register
0x14E	WOE3R	Watchdog Output Enable 3 Register
0x150	WOP2R	Watchdog Output Polarity 2 Register
0x152	WOP3R	Watchdog Output Polarity 3 Register
0x080	SPICFGR	SPI Configure Register
0x082	SPIBRR	SPI Baud Rate Register
0x084	SPIDBSR	SPI Delay Byte and SS Register
0x086	SPIDTR	SPI Delay Transfer Register
0x088	SPIRPTR	SPI RDY / Pulse Time Register
0x08A	SPILTR	SPI LDAC Time Register
0x08C	SPIPRLR	SPI Pulse/ RDY/ LDAC Register
0x090	SPI01BCR	SPI 0/1 Byte Count Register
0x092	SPI23BCR	SPI 2/3 Byte Count Register
0x094	SPI45BCR	SPI 4/5 Byte Count Register
0x096	SPI67BCR	SPI 6/7 Byte Count Register
0x098	SPI03SSR	SPI 0/1/2/3 slave Select Register
0x09A	SPI47SSR	SPI 4/5/6/7 slave Select Register
0x0A8	SPINTSR	SPI Interrupt Status Register
0x0AA	SPITSR	SPI Timeout Status Register
0x0AC	SPIPOSR	SPI Pulse Overrun Status Register
0x0AE	SPIDSR	SPI Data Status Register

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

0x0B0	SPIC0DR	SPI Channel 0 Data Register
0x0B8	SPIC1DR	SPI Channel 1 Data Register
0x0C0	SPIC2DR	SPI Channel 2 Data Register
0x0C8	SPIC3DR	SPI Channel 3 Data Register
0x0D0	SPIC4DR	SPI Channel 4 Data Register
0x0D8	SPIC5DR	SPI Channel 5 Data Register
0x0E0	SPIC6DR	SPI Channel 6 Data Register
0x0E8	SPIC7DR	SPI Channel 7 Data Register
0x0F0	SPIDSMR	SPI Data Status Mirror Register
0x0F2	SPIMCR	SPI Master Control Register
0x100	INTCR	Interrupt Configure Register
0x102	INTSR	Interrupt Status Register

Figure 2-4. Function Registers Mirror Mapping Table

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

3. EtherCAT SSC Overview

Please refer to "**Application Note ET9300 EtherCAT SubDevice Stack Code (SSC)**" from Beckhoff Automation GmbH for details.

3-1. Code Structure

The EtherCAT SubDevice Stack Code (SSC) as seen in below figure consists of 3 parts:

- **PDI / Hardware Abstraction**
  Hardware specific, need to be implemented according the platform PDI.
  Ready to used samples/implementations are available for certain platforms.
  The interface to the generic stack.

- **Generic EtherCAT Stack**
  Implements the full EtherCAT state machine, mailbox communication and generic process data exchange. No further implementation needs (only configured via the SSC Tool).

- **User Application**
  Need to be implemented (a table-based code generation for the application is available).
  Ready to used samples are available.
  The interface to the generic stack.

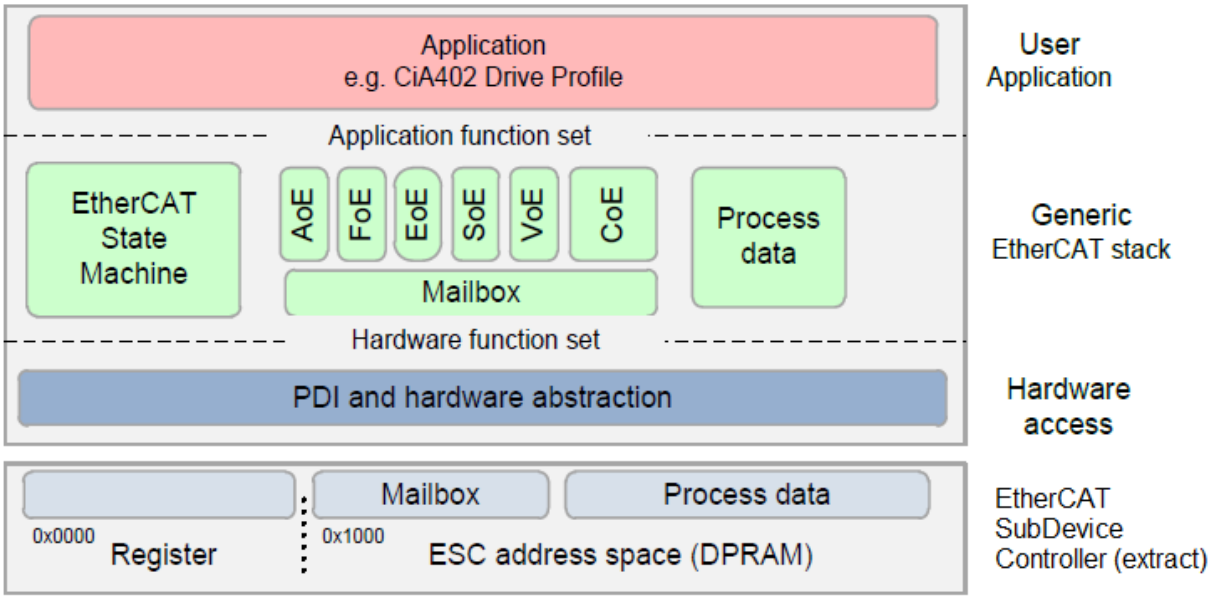

Figure 3-1. EtherCAT SSC Code Structure

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

The following figure shows the association between the SubDevice Stack Code layers and the source files.

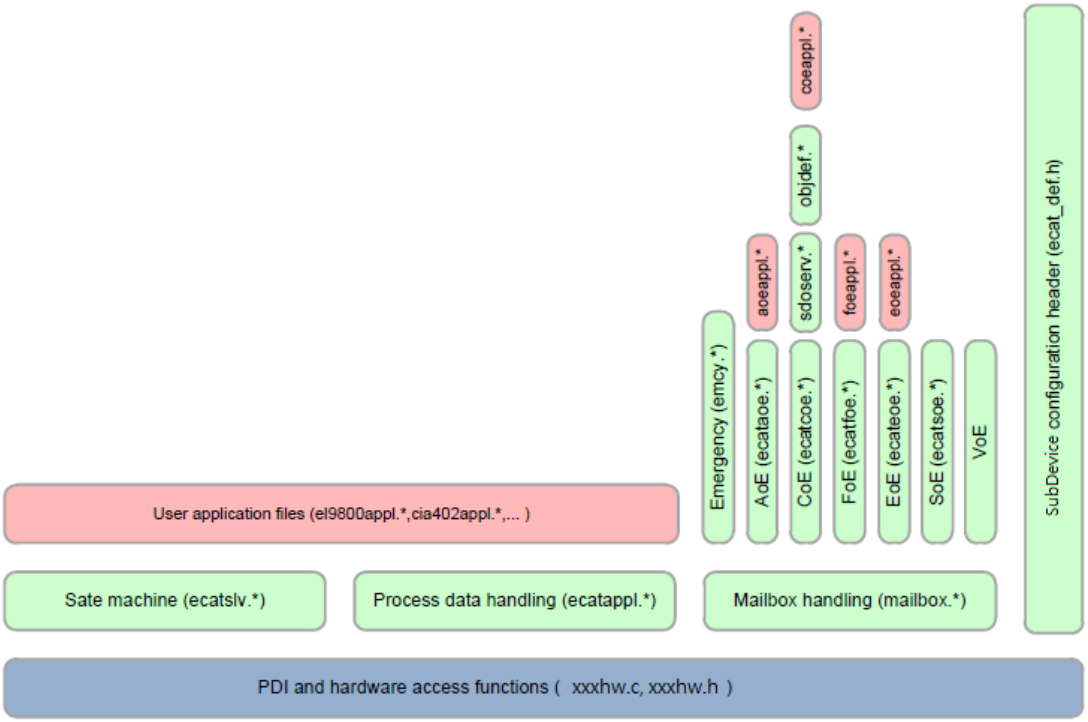

Figure 3-2. EtherCAT SSC Files Association

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

3-2. Execution Structure

The SSC execution consists of an initialization phase (executed only once) and a cyclic phase (executed continuously without interruptions).

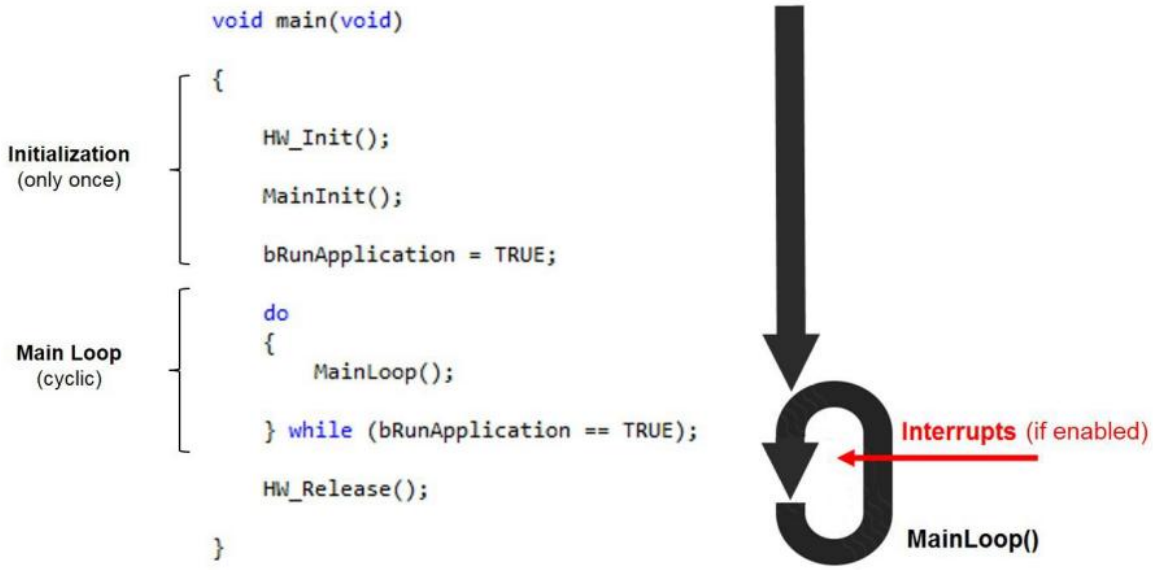

Figure 3-3. Basic SSC Execution Structure

The function MainLoop() contains the main cycle of SubDevice's firmware, which always runs in free-run (unsynchronized endless loop):

MainLoop() (file: ecatappl.c)

- ECAT_Main() (file: ecatslv.c)
- CoE_Main() (file: coeappl.c)
- [PDO_OutputMapping() (file: ecatappl.c) only in Free-Run mode]
- [ECAT_Application() (file: ecatappl.c) only in Free-Run mode]
- [PDO_InputMapping() (file: ecatappl.c) only in Free-Run mode]

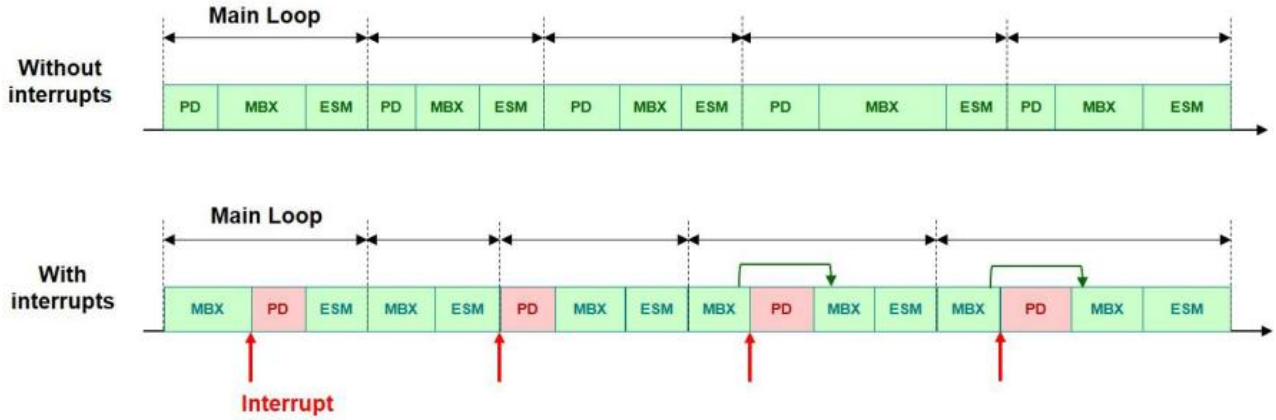

Figure 3-4. SSC MainLoop Execution

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

3-3. Interrupt Handling

The SSC makes use of up to 4 interrupts:

1. **Timer Interrupts:** Platform internal 1ms timers to set the EtherCAT LEDs and watchdogs. If no timer interrupt is configured (ECAT_TIMER_INT = 0) the required 1ms cycle is based on the Mainloop a platform internal counter.

2. **Sync0:** Process data handling and application synchronization with Distributed Clocks (DC) sync0 signal.

3. **Sync1:** Process data handling and application synchronization with Distributed Clocks (DC) sync1 signal.

4. **PDI Interrupt:** Process data handling and application synchronization with AL events. The events which trigger the PDI interrupt can be configured by the SubDevice application.

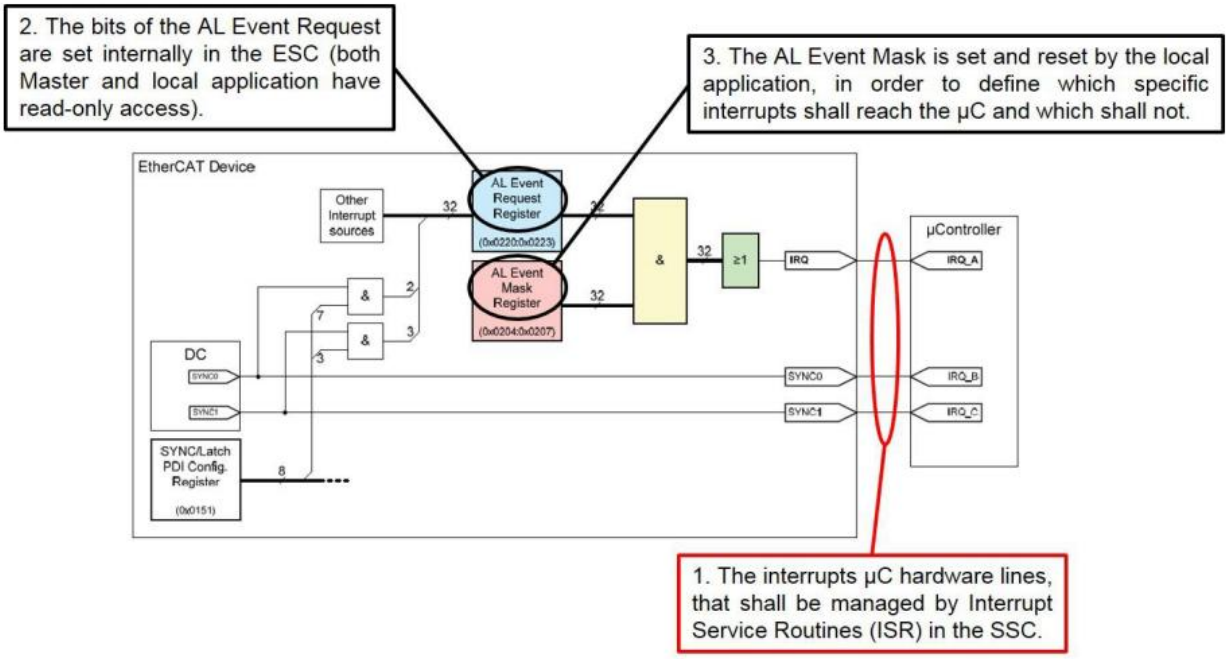

Figure 3-5. ESC Interrupts Basics

The only interrupt sources which are actually handled by PDI ISR in the SSC, and therefore not filtered by the AL Event Mask, are the Process Data SyncManager interrupts, and specifically: SM2 if Process Data Outputs are configured for the SubDevice, otherwise SM3. All the other possible sources (include all other SyncManagers) can be directly polled in ESC Register 0x0220.

The Interrupt mask is configured by the functions “SetALEventMask()” and “ResetALEventMask()” (file: ecatslv.c).

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

3-4. Process Data Handling

The EtherCAT SubDevice process data communication can be separated in 2 steps. The first one is the low level on-the-fly data exchange. The SSC reads/writes data from/to the EtherCAT frame and stores/reads the data to the internal DPRAM. In the second step the SubDevice application will do further data processing/calculation.

The process data handling in the SSC is managed in 3 functions of the generic stack. Each of these functions triggers the corresponding application specific functions.

1. **PDO_OutputMapping():** Handles the data from the MainDevice to the SubDevice.
2. **ECAT_Application():** Contains the SubDevice application, set general purpose outputs and read general purpose inputs.
3. **PDO_InputMapping():** Handles the data from the SubDevice to the MainDevice.

The calling sequence of the 3 listed functions is always the same (the one listed above). The functions trigger depends on the configured synchronization mode.

Below figure illustrates an example application mapping with PDO_OutputMapping for mapping PDO from MainDevice to local LED of SubDevice.

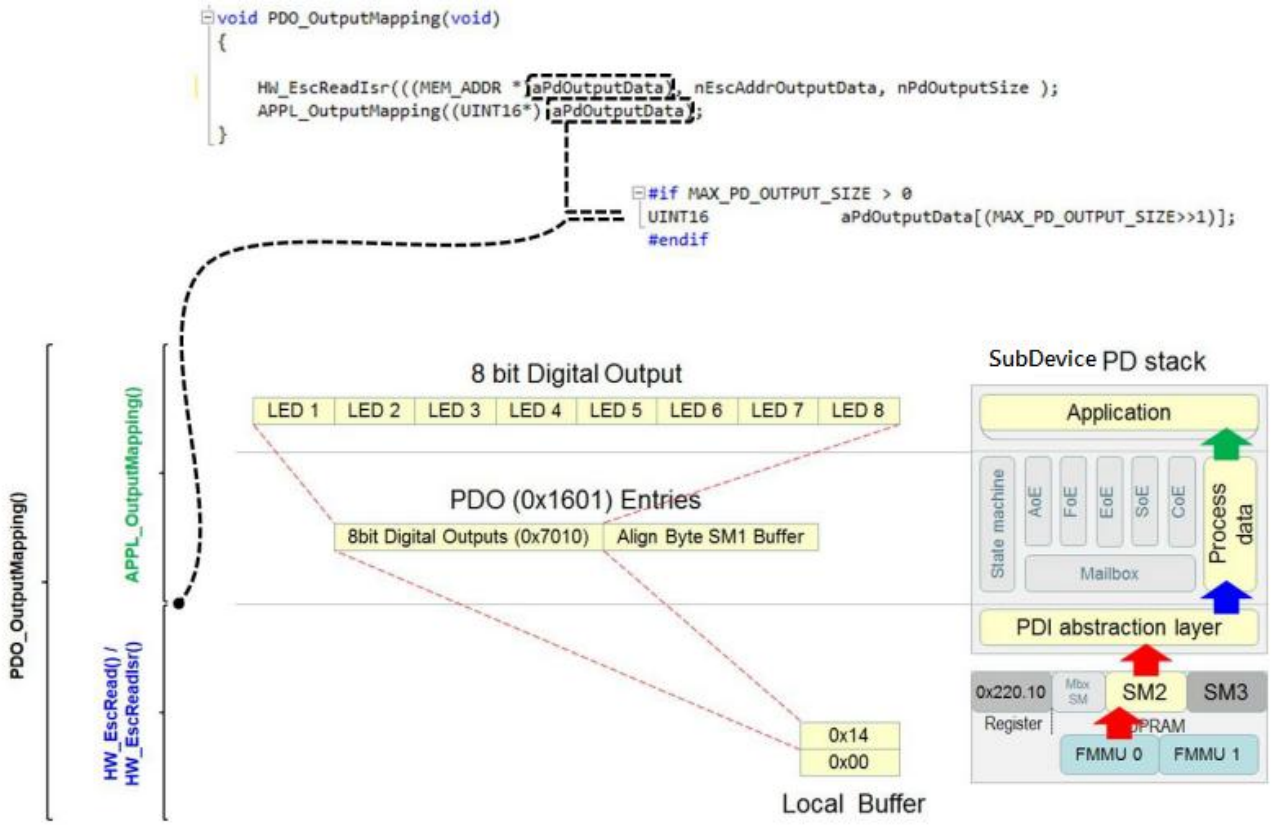

Figure 3-6. PDO Output Mapping Example

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

Below figure illustrates an example application mapping with PDO_InputMapping for mapping analog data from SubDevice to MainDevice.

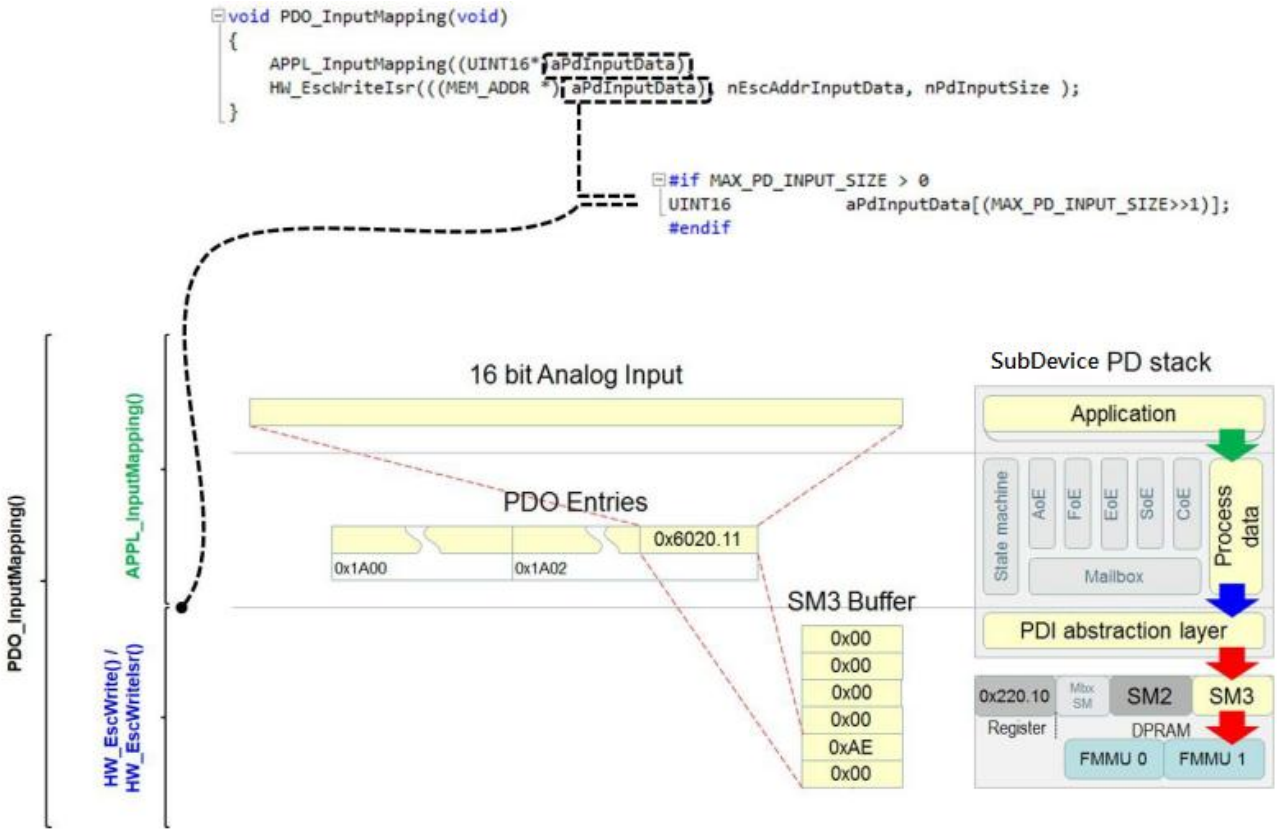

Figure 3-7. PDO Input Mapping Example

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

4. SSC Hardware Access Implementation

The SubDevice Stack Code is executable on multiple platforms and controller architectures. This chapter describes the available hardware implementations/defines and how to implement a new specific hardware access.

To support multiple hardware architectures the SSC includes multiple defines to fulfill the specific hardware requirements. Below table illustrates the SSC Hardware Related Defines includes a list of the defined hardware defines (located in ecat_def.h or in the EtherCAT SSC Tool from Beckhoff).

Define	Description
EL9800_HW	Hardware access if the SubDevice code is executed on the PIC mounted on EL9800 EtherCAT Evaluation Kit from Beckhoff Automation GmbH. It includes PIC initialization and ESC access via SPI. This configuration could also be used if the SSC needs to be adapted to any other 8 or 16Bit µC which accesses the ESC via SPI.
MCI_HW	Generic MCI implementation. Can be used if any kind of memory interface face is used to access the ESC.
FC1100_HW	Specific hardware implementation for the FC1100 PCI EtherCAT SubDevice card from Beckhoff. Used on Win32 operating system.
CONTROLLER_16BIT	This define shall be used if the SubDevice code is built for a 16Bit µC.
CONTROLLER_32BIT	This define shall be used if the SubDevice code is built for a 32Bit µC
ESC_16BIT_ACCESS	If this define is set, then only 16Bit aligned accesses will be performed on the ESC.
ESC_32BIT_ACCESS	If this define is set, then only 32Bit aligned accesses will be performed on the ESC.
MBX_16BIT_ACCESS	If this define is set, then the SubDevice code will only access mailbox data 16Bit aligned. If the mailbox data is copied to the local µC memory and the define “CONTROLLER_16BIT” is set, then this define should also be set.
BIG_ENDIAN_16BIT	These define needs to be set if the µC always accesses external memory 16Bit wise. It works in big endian format and the switching of Low Byte and High Byte is done in hardware.
BIG_ENDIAN_FORMAT	This define shall be set if the µC works in big endian format.

Figure 4-1. SSC Hardware Related Defines

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

The defines “EL_9800_HW”, “MCI_HW”, “FC1100_HW” are used to activate a predefined hardware access implementation. An extract platforms/uC is listed in below figure. Recommended Hardware Configurations including the recommended defines. Some of the configurations can also be selected if a new project is created with the SSC Tool. If none of these defines are used, then user specific hardware access files need to be added to the SubDevice project.

In general the hardware access implementation needs to support the following features:
- ESC read/write access
- Timer supply (at least 1ms base tick)
- Calling of timer handler every 1 ms (only required if timer interrupt handling is supported. “ECAT_TIMER_INT” set to 1)
- Calling the interrupt specific functions (only required if synchronization is supported)
  - PDI ISR (required if “AL_EVENT_SUPPORTED” set to 1)
  - SYNC0 ISR (required if DC_SUPPORTED set to 1)

Platform	EL9800_HW	PIC24	PIC18	MCI_HW	FC1100_HW	CONTROLLER_16BIT	CONTROLLER_32BIT	ESC_16BIT_ACCESS	ESC_32BIT_ACCESS	MBX_16BIT_ACCESS	BIG_ENDIAN_16BIT	BIG_ENDIAN_FORMAT	Comment
Microchip PIC18F452 Generic : 8Bit µC ; SPI ESC access	1	0	1	0	0	0	0	0	0	0	0	0	The stack is ready to use if the PIC 18 on the EL9800 EtherCAT Evaluation board is used. Otherwise there might be requirements to adapt the hardware access
Microchip PIC24HJ128GP306 Generic: 16Bit µC; SPI ESC access	1	1	0	0	0	1	0	1	0	1	0	0	The stack is ready to use if the PIC 24 on the EL9800 EtherCAT Evaluation board is used. Otherwise there might be requirements to adapt the hardware access.
x86 (OS Windows)	0	0	0	0	1	0	1	0	1	0	0	0	The stack is ready to use if the stack shall run on a Win32 OS in user mode. Otherwise changes in hardware access might be required. The define “FC1100_HW” is a adapted implementation based on “MCI_HW
Texas Instruments Sitara AM335x	0	0	0	0	0	0	0	0	0	0	0	0	To use the SSC on TI AM335x chips the hardware access files from the TI SDK need to be added to the project. The files can be added to the slave project via the patch file (delivered with the SDK), by selecting the TI configuration in the SSC Tool or by adding the files manually.
Altera® NIOS®II (ESC connected via Avalon bus)	0	0	0	1	0	x	x	x	x	x	0	0	x: depends on the NIOS® configuration in the SOPC builder. In general the following points need to be adapted: - define “MAKE_PTR_TO_ESC” - ISRs for Timer/PDI interrupt and Sync0 (depends on the supported features) - Implement timer access functions and macros Depending on the platform configuration further changes may be required.
Xilinx Microblaze™ (ESC connected via PLB)	0	0	0	1	0	x	x	x	x	x	x	0	x: depends on the Microblaze™ configuration. In general the following points need to be adapted: - define “MAKE_PTR_TO_ESC” - ISRs for Timer/PDI interrupt and Sync0 (depends on the supported features) - Implement timer access functions and macros Depending on the platform configuration further changes may be required.
Renesas - RIN32M3	0	0	0	0	0	0	0	0	0	0	0	0	To use the SSC on Renesas RIN32M3 chip the chip specific hardware access files need to be added to the project. The files are added automatically if the Renesas PIN32M3 configuration is selected in the SSC Tool.
Xilinx ZYNQ™ (ESC connected via the on-chip bus)	0	0	0	1	0	x	x	x	x	x	0	0	x: depends on the ZYNQ™ configuration. In general the following points need to be adapted: - define “MAKE_PTR_TO_ESC” - ISRs for Timer/PDI interrupt and Sync0 (depends on the supported features) - Implement timer access functions and macros Depending on the platform configuration further changes may be required.

Figure 4-2. SSC uC Hardware Configuration Examples

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

The following functions should be called and provided by the hardware access layer as below figure.

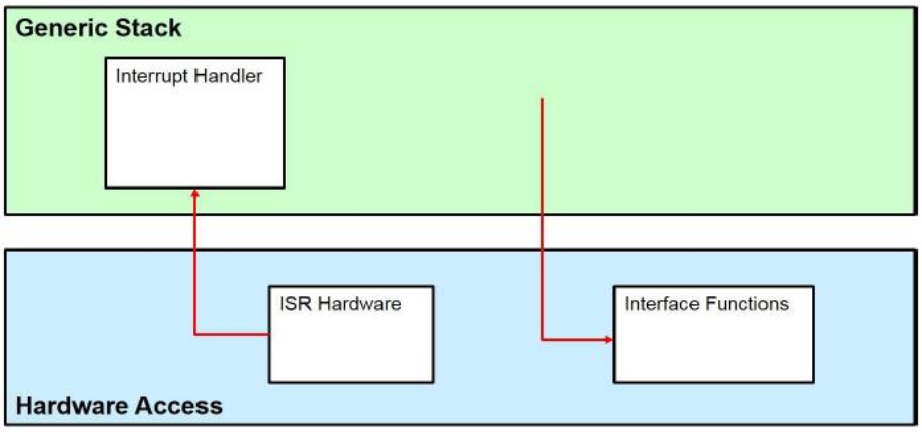

Figure 4-3. SSC Hardware Functions Schema

- **Interrupt Handler:** Functions completely defined and implemented in the generic EtherCAT stack, shall be called by the hardware interrupt routines of the specific uC. If interrupts are used also 2 macros shall be defined “ENABLE_ESC_INT” and “DISABLE_ESC_INT”. These shall enable/disable all 4 interrupt sources.

- **Interface Functions/Macros, Read Access, Write Access:**
  Functions called by the generic EtherCAT stack, shall be implemented in the hardware access code.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

4-1. Interrupt Handler Functions

The following functions are provided by the generic SubDevice Stack Code (defined in ecatappl.h) and need to be called from the hardware access layer.

4-1-1. ECAT_CheckTimer

Prototype	**void ECAT_CheckTimer (void)**
Parameter	void
Return	void
Description	This function needs to be called every 1ms from a timer ISR (ECAT_TIMER_INT = 1). If no timer interrupt is supported this function is called automatically when 1ms is elapsed (based on the provided timer).

4-1-2. PDI_Isr

Prototype	**void PDI_Isr (void)**
Parameter	void
Return	void
Description	This function need to be called from the PDI ISR. For the PDI specific pin naming and the interrupt generation logic please refer to [6]. To support PDI interrupt handling it is also required to set “AL_EVENT_ENABLED” to 1.

4-1-3. Sync0_Isr

Prototype	**void Sync0_Isr (void)**
Parameter	void
Return	void
Description	This function needs to be called from the Sync0 ISR. The Sync0 interrupt is generated by the DC Unit of the ESC. It is currently not supported by default to map the Sync0 signal to the PDI interrupt. To support Dc synchronization “DC_SUPPORTED” need to be set.

4-1-4. Sync1_Isr

Prototype	**void Sync1_Isr (void)**
Parameter	void
Return	void
Description	This function needs to be called from the Sync1 ISR. The Sync1 interrupt is generated by the DC Unit of the ESC. It is currently not supported by default to map the Sync1 signal to the PDI interrupt. To support Dc synchronization “DC_SUPPORTED” need to be set.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

4-2. Interface Functions/Marcos

The functions and marcos listed in this chapter need to be implemented by the hardware access layer for different microcontroller.

4-2-1. HW_Init

Prototype	**UINT16 HW_Init (void)**
Parameter	void
Return	0 if initialization was successful > 0 if error has occurred while initialization
Description	Initializes the host controller, process data interface (PDI) and allocates resources which are required for hardware access.

4-2-2. HW_Release

Prototype	**void HW_Release (void)**
Parameter	void
Return	void
Description	Initializes the host controller, process data interface (PDI) and allocates resources which are required for hardware access.

4-2-3. HW_GetALEventRegister

Prototype	**UINT16 HW_GetALEventRegister (void)**
Parameter	void
Return	Content of register 0x220-0x221
Description	Get the first two bytes of the AL Event register (0x220-0x221).

4-2-4. HW_GetALEventRegister_Isr

Prototype	**UINT16 HW_GetALEventRegister_Isr (void)**
Parameter	void
Return	Content of register 0x220-0x221
Description	This function should be implemented if a special function for ESC access from interrupt service routines is required; otherwise this function is defined as HW_GetALEventRegister. Get the first two bytes of the AL Event register (0x220-0x221).

4-2-5. HW_ResetALEventMask

Prototype	**void HW_ResetALEventMask (UINT16 intMask)**
Parameter	“intMask” Interrupt mask (disabled interrupt shall be zero)
Return	void
Description	Performs a logical AND with the AL Event Mask register (0x0204 : 0x0205). This function is only required if “AL_EVENT_ENABLED” is set. **NOTE**: This function is only required for SSC 5.10 or older.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

4-2-6. HW_SetALEventMask

Prototype	**void HW_SetALEventMask (UINT16 intMask)**
Parameter	"intMask" Interrupt mask (enabled interrupt shall be one)
Return	void
Description	Performs a logical OR with the AL Event Mask register (0x0204 : 0x0205). This function is only required if "AL_EVENT_ENABLED" is set. **NOTE**: This function is only required for SSC 5.10 or older.

4-2-7. HW_SetLed

Prototype	**void HW_SetLed (UINT8 RunLed, UINT8 ErrLed)**
Parameter	"RunLed" EtherCAT Run LED state "ErrLed" EtherCAT Error LED state
Return	void
Description	Updates the EtherCAT Run and Error LEDs (or EtherCAT Status LED).

4-2-8. HW_RestartTarget

Prototype	**void HW_RestartTarget (void)**
Parameter	void
Return	void
Description	Resets the hardware. This function is only required if "BOOTSTRAPMODE_SUPPORTED" is set.

4-2-9. HW_DisableSyncManChannel

Prototype	**void HW_DisableSyncManChannel (UINT8 channel)**
Parameter	"channel" SyncManager channel
Return	void
Description	Disables selected SyncManager channel. Sets bit 0 of the corresponding 0x807 register. **NOTE**: This function is only required for SSC 5.10 or older.

4-2-10. HW_EnableSyncManChannel

Prototype	**void HW_EnableSyncManChannel (UINT8 channel)**
Parameter	"channel" SyncManager channel
Return	void
Description	Enables selected SyncManager channel. Resets bit 0 of the corresponding 0x807 register. **NOTE**: This function is only required for SSC 5.10 or older.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

4-2-11. HW_GetSyncMan

Prototype	**TSYNCMAN * HW_GetSyncMan (UINT8 channel)**
Parameter	"channel" SyncManager channel
Return	Pointer to the SyncManager channel description. The SyncManager description structure size is always 8 Byte, the content of "TSYNCMAN" differs depending on the supported ESC access.
Description	Gets the content of the SyncManager register from the stated channel. Reads 8 Bytes starting at 0x800 + 8*channel. **NOTE**: This function is only required for SSC 5.10 or older.

4-2-12. HW_GetTimer

Prototype	**UINT32 HW_GetTimer (void)**
Parameter	void
Return	Current timer value
Description	Reads the current register value of the hardware timer. If no hardware timer is available the function shall return the counter value of a multimedia timer. The timer ticks value (increments / ms) is defined in "ECAT_TIMER_INC_P_MS". This function is required if no timer interrupt is supported ("ECAT_TIMER_INT" = 0) and to calculate the bus cycle time.

4-2-13. HW_ClearTimer

Prototype	**void HW_ClearTimer (void)**
Parameter	void
Return	void
Description	Clears the hardware timer value.

4-2-14. HW_EepromReload

Prototype	**UINT16 HW_EepromReload (void)**
Parameter	void
Return	0 <> Error during EEPORM reload 0 = EEPROM load correct
Description	This function is called if an EEPROM reload request is triggered by the MainDevice. Only required if EEPROM Emulation is supported and the function pointer "pAPPL_EEPROM_Reload" is not set. In case that the full eeprom emulation is configured (register 0x502, bit6 is 1) the reload function is not called and does not to be implemented.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

4-3. Read Access Functions

4-3-1. HW_EscRead

Prototype	**void HW_EscRead (MEM_ADDR *pData, UINT16 Address, UINT16 Len)**
Parameter	"pData" Pointer to local destination buffer. Type of the pointer depends on the host controller architecture (specified in ecat_def.h or the SSC Tool). "Address" EtherCAT SubDevice Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid addresses are used depending on 8Bit/16Bit or 32 Bit ESC access (specified in ecat_def.h or the SSC Tool). "Len" Access size in Bytes
Return	void
Description	Reads from the EtherCAT SubDevice Controller. This function is used to access ESC registers and the DPRAM area.

4-3-2. HW_EscReadIsr

Prototype	**void HW_EscReadIsr (MEM_ADDR *pData, UINT16 Address, UINT16 Len)**
Parameter	"pData" Pointer to local destination buffer. Type of the pointer depends on the host controller architecture (specified in ecat_def.h or the SSC Tool). "Address" EtherCAT SubDevice Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid addresses are used depending on 8Bit/16Bit or 32 Bit ESC access (specified in ecat_def.h or the SSC Tool). "Len" Access size in Bytes
Return	void
Description	This function should be implemented if a special function for ESC access from interrupt service routines is required; otherwise this function is defined as "HW_EscRead". Reads from the EtherCAT SubDevice Controller. This function is used to access ESC registers and the DPRAM area.

4-3-3. HW_EscReadDWord

Prototype	**void HW_EscReadDWord (UINT32 DWordValue, UINT16 Address)**
Parameter	"DWordValue" Local 32Bit variable where the register value shall be stored. "Address" EtherCAT SubDevice Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid 32Bit addresses are used.
Return	void
Description	Reads two words from the specified address of the EtherCAT SubDevice Controller. In case that no specific read DWORD marco is used the default EscRead function may be used: "HW_EscRead(((MEM_ADDR *)&(DWordValue)),((UINT16)(Address)),4)"

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

4-3-4. HW_EscReadDWordIsr

Prototype	**void HW_EscReadDWordIsr (UINT32 DWordValue, UINT16 Address)**
Parameter	"DWordValue" Local 32Bit variable where the register value shall be stored. "Address" EtherCAT SubDevice Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid 32Bit addresses are used.
Return	void
Description	This function should be implemented if a special function for ESC access from interrupt service routines is required; otherwise this function is defined as "HW_EscReadWord". Reads two words from the specified address of the EtherCAT SubDevice Controller.

4-3-5. HW_EscReadWord

Prototype	**void HW_EscReadWord (UINT16 WordValue, UINT16 Address)**
Parameter	"WordValue" Local 16Bit variable where the register value shall be stored. "Address" EtherCAT SubDevice Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid 16Bit addresses are used.
Return	void
Description	Reads one word from the specified address of the EtherCAT SubDevice Controller. Only required if "ESC_32BIT_ACCESS" is not set. In case that no specific read WORD marco is used the default EscRead function may be used: "HW_EscRead(((MEM_ADDR *)&(WordValue)),((UINT16)(Address)),2)"

4-3-6. HW_EscReadWordIsr

Prototype	**void HW_EscReadWordIsr (UINT16 WordValue, UINT16 Address)**
Parameter	"WordValue" Local 16Bit variable where the register value shall be stored. "Address" EtherCAT SubDevice Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid 16Bit addresses are used.
Return	void
Description	This function should be implemented if a special function for ESC access from interrupt service routines is required; otherwise this function is defined as "HW_EscReadWord". Reads one word from the specified address of the EtherCAT SubDevice Controller. Only required if "ESC_32_BIT_ACCESS" is not set.

4-3-7. HW_EscReadByte

Prototype	**void HW_EscReadByte (UINT8 ByteValue, UINT16 Address)**
Parameter	"ByteValue" Local 8Bit variable where the register value shall be stored. "Address" EtherCAT SubDevice Controller address. Specifies the offset within the ESC memory area in Bytes.
Return	void
Description	Reads one byte from the EtherCAT SubDevice Controller. Only required if "ESC_16BIT_ACCESS" and "ESC_32BIT_ACCESS" are not set.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

4-3-8. HW_EscReadByteIsr

Prototype	**void HW_EscReadByteIsr (UINT8 ByteValue, UINT16 Address)**
Parameter	"ByteValue" Local 8Bit variable where the register value shall be stored. "Address" EtherCAT SubDevice Controller address. Specifies the offset within the ESC memory area in Bytes.
Return	void
Description	This function should be implemented if a special function for ESC access from interrupt service routines is required; otherwise this function is defined as "HW_EscReadByte". Reads one byte from the EtherCAT SubDevice Controller. Only required if "ESC_16BIT_ACCESS" and "ESC_32BIT_ACCESS" are not set.

4-3-9. HW_EscReadMbxMem

Prototype	**void HW_EscReadMbxMem (MEM_ADDR *pData, UINT16 Address, UINT16 Len)**
Parameter	"pData" Pointer to local destination mailbox buffer. Type of the pointer depends on the host controller architecture (specified in ecat_def.h or the SSC Tool). "Address" EtherCAT SubDevice Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid addresses are used depending on 8Bit/16Bit or 32 Bit ESC access (specified in ecat_def.h or the SSC Tool). "Len" Access size in Bytes
Return	void
Description	Reads data from the ESC and copies to SubDevice mailbox memory. If the local mailbox memory is also located in the application memory this function is equal to "HW_EscRead".

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

4-4. Write Access Functions

4-4-1. HW_EscWrite

Prototype	**void HW_EscWrite (MEM_ADDR *pData, UINT16 Address, UINT16 Len)**
Parameter	"pData" Pointer to local source buffer. Type of the pointer depends on the host controller architecture (specified in ecat_def.h or the SSC Tool). "Address" EtherCAT SubDevice Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid addresses are used depending on 8Bit/16Bit or 32 Bit ESC access (specified in ecat_def.h or the SSC Tool). "Len" Access size in Bytes
Return	void
Description	Writes from the EtherCAT SubDevice Controller. This function is used to access ESC registers and the DPRAM area.

4-4-2. HW_EscWriteIsr

Prototype	**void HW_EscWriteIsr (MEM_ADDR *pData, UINT16 Address, UINT16 Len)**
Parameter	"pData" Pointer to local source buffer. Type of the pointer depends on the host controller architecture (specified in ecat_def.h or the SSC Tool). "Address" EtherCAT SubDevice Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid addresses are used depending on 8Bit/16Bit or 32 Bit ESC access (specified in ecat_def.h or the SSC Tool). "Len" Access size in Bytes
Return	void
Description	This function should be implemented if a special function for ESC access from interrupt service routines is required; otherwise this function is defined as "HW_EscWrite". Writes from the EtherCAT SubDevice Controller. This function is used to access ESC registers and the DPRAM area.

4-4-3. HW_EscWriteDWord

Prototype	**void HW_EscWriteDWord (UINT32 DWordValue, UINT16 Address)**
Parameter	"DWordValue" Local 32Bit variable which contains the data to be written to the ESC memory area. "Address" EtherCAT SubDevice Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid 32Bit addresses are used.
Return	void
Description	This function should be implemented if a special function for ESC access from interrupt service routines is required; otherwise this function is defined as "HW_EscWrite". Writes from the EtherCAT SubDevice Controller. This function is used to access ESC registers and the DPRAM area.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

4-4-4. HW_EscWriteDWordIsr

Prototype	**void HW_EscWriteDWordIsr (UINT32 DWordValue, UINT16 Address)**
Parameter	"DWordValue" Local 32Bit variable which contains the data to be written to the ESC memory area. "Address" EtherCAT SubDevice Controller address . Specifies the offset within the ESC memory area in Bytes. Only valid 32Bit addresses are used.
Return	void
Description	This function should be implemented if a special function for ESC access from interrupt service routines is required; otherwise this function is defined as "HW_EscWriteWord". Writes two words to the EtherCAT SubDevice Controller.

4-4-5. HW_EscWriteWord

Prototype	**void HW_EscWriteWord (UINT16 WordValue, UINT16 Address)**
Parameter	"WordValue" Local 16Bit variable which contains the data to be written to the ESC memory area. "Address" EtherCAT SubDevice Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid 16Bit addresses are used.
Return	void
Description	Writes one word to the EtherCAT SubDevice Controller. Only required if "ESC_32BIT_ACCESS" is not set.

4-4-6. HW_EscWriteWordIsr

Prototype	**void HW_EscWriteWordIsr (UINT16 WordValue, UINT16 Address)**
Parameter	"WordValue" Local 16Bit variable which contains the data to be written to the ESC memory area. "Address" EtherCAT SubDevice Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid 16Bit addresses are used.
Return	void
Description	This function should be implemented if a special function for ESC access from interrupt service routines is required; otherwise this function is defined as "HW_EscWriteWord". Writes one word to the EtherCAT SubDevice Controller. Only required if "ESC_32BIT_ACCESS" is not set.

4-4-7. HW_EscWriteByte

Prototype	**void HW_EscWriteByte (UINT8 ByteValue, UINT16 Address)**
Parameter	"ByteValue" Local 8Bit variable which contains the data to be written to the ESC memory area. "Address" EtherCAT SubDevice Controller address.Specifies the offset within the ESC memory area in Bytes.
Return	void
Description	Writes one byte to the EtherCAT SubDevice Controller. Only defined if "ESC_16BIT_ACCESS" and "ESC_32BIT_ACCESS" are disabled.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

4-4-8. HW_EscWriteByteIsr

Prototype	**void HW_EscWriteByteIsr (UINT8 ByteValue, UINT16 Address)**
Parameter	"ByteValue" Local 8Bit variable which contains the data to be written to the ESC memory area "Address" EtherCAT SubDevice Controller address. Specifies the offset within the ESC memory area in Bytes.
Return	void
Description	This function should be implemented if a special function for ESC access from interrupt service routines is required; otherwise this function is defined as "HW_EscWriteByte". Writes one byte to the EtherCAT SubDevice Controller. Only defined if "ESC_16BIT_ACCESS" and "ESC_32BIT_ACCESS" are disabled.

4-4-9. HW_EscWriteMbxMem

Prototype	**void HW_EscWriteMbxMem (MEM_ADDR *pData, UINT16 Address, UINT16 Len)**
Parameter	"pData" Pointer to local source mailbox buffer. Type of the pointer depends on the host controller architecture (specified in ecat_def.h or the SSC Tool). "Address" EtherCAT SubDevice Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid addresses are used depending on 8Bit/16Bit or 32 Bit ESC access (specified in ecat_def.h or the SSC Tool). "Len" Access size in Bytes.
Return	void
Description	Writes data from the SubDevice mailbox memory to ESC memory. If the local mailbox memory is also located in the application memory this function is equal to "HW_EscWrite".

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5. AX58101(2) Extended Function Driver API

The AX58101(2) EtherCAT SubDevice controller (ESC) is an application–specific integrated circuit (ASIC) and designed to target on EtherCAT based industrial applications, it consists of radical ESC functions and AX58101(2) extended functions, this section is mainly to introduce the software drivers of AX58101(2) extended functions.
The AX58101(2) Extended Function Driver can run on various Micro-Controllers and used to control and monitoring extended function hardware via PDI interface, user can develop his own EtherCAT applications base on it.
The typical architecture of ESC software is shown as below, it contains driver, middleware and application layers. The PDI Master Driver can be Local Bus or Serial Peripheral Interface (SPI) and depended on operated platform (PC or MCU) used, the SubDevice Stack Code (SSC) is generated by SSC tool, and the EtherCAT Application is user defined.

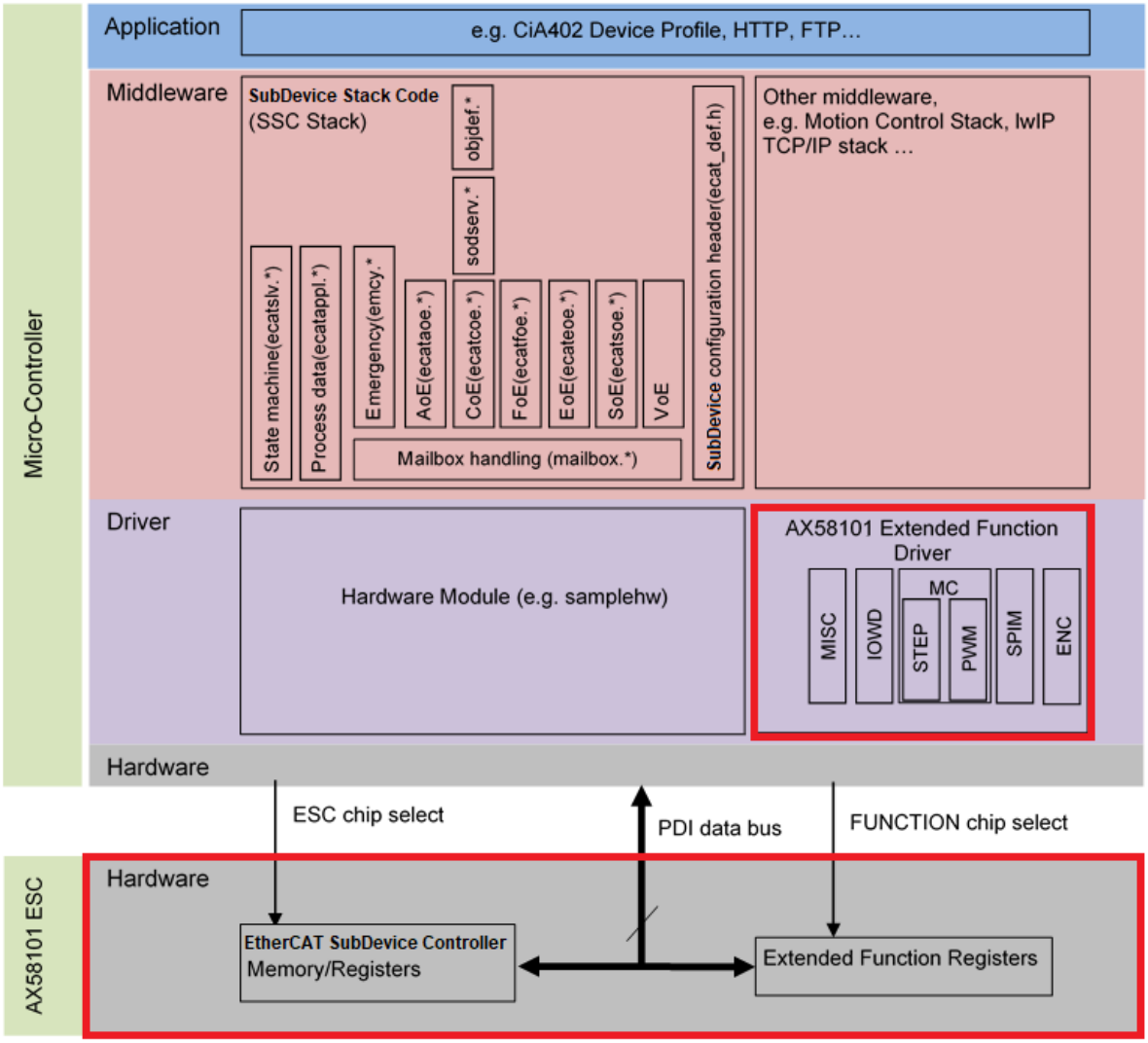

Figure 5-1. AX58101(2) Extended Function Driver APIs

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-1. MISC Module

MISC module is used to handle interrupts mask/un-mask, clear pended interrupt flags and enable/disable debug mode by setting ESTOR register.

Related files of MISC module are explained as below.

ax58101_misc.c –
The major file of MISC sub-module, it is responsible of interrupt handling and get Efuse content.

ax58101_misc.h –
The header file of MISC module.

The MISC Module Macros and Functions are illustrated as below figure.

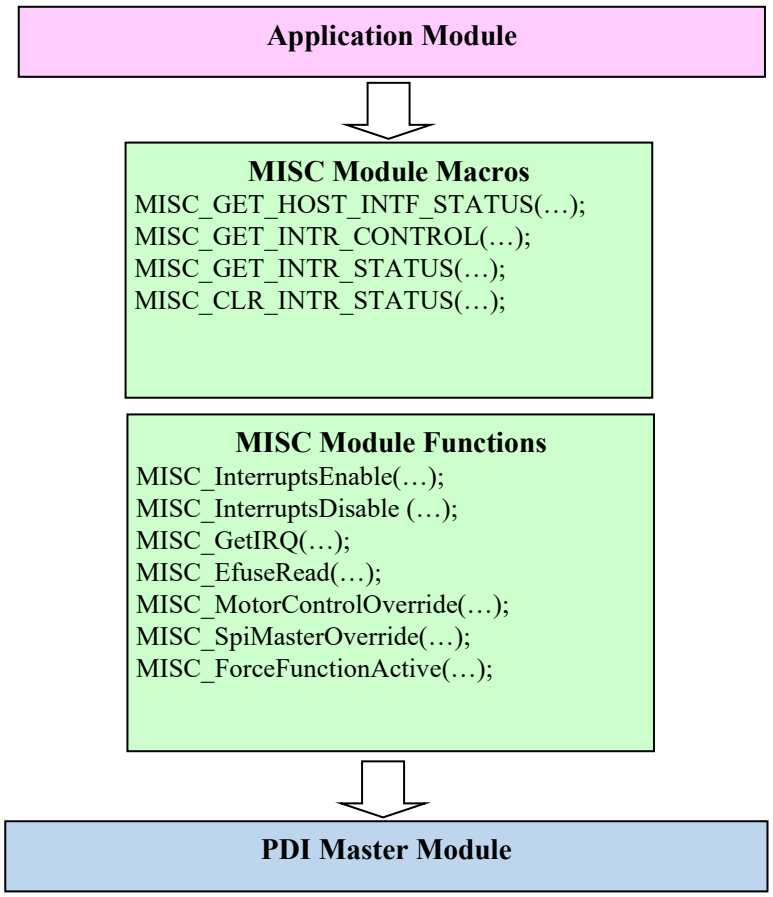

Figure 5-2. MISC Module Block Diagram

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-2-1. MISC Module Definition/Macro Description

5-2-1-1. MISC_GET_HOST_INTF_STATUS

Prototype	**MISC_GET_HOST_INTF_STATUS(pcStu)**
Parameter	"pcStu" A byte pointer to store HSTSR value.
Return	Return the value of HSTSR register.
Description	This macro is used get host interface status register (HSTSR) value.

5-2-1-2. MISC_GET_INTR_CONTROL

Prototype	**MISC_GET_INTR_CONTROL(pwCtrl)**
Parameter	"pwCtrl" A word pointer to store INTCR value.
Return	Return the value of INTCR register.
Description	This macro is used get interrupt control register (INTCR) value.

5-2-1-3. MISC_GET_INTR_STATUS

Prototype	**MISC_GET_INTR_STATUS(pwFlags)**
Parameter	"pwFlags" A word pointer to store INTSR value.
Return	Return the value of INTSR register.
Description	This macro is used get interrupt status register (INTSR) value.

5-2-1-4. MISC_CLR_INTR_STATUS

Prototype	**MISC_CLR_INTR_STATUS(pwFlags)**
Parameter	"pwFlags" A word pointer to a bit-Mask value which used to clear INTSR.
Return	None
Description	This macro is used clear interrupt status register (INTSR) value.

5-2-2. MISC Module Function Description

The MISC module provides following API functions.

5-2-2-1. MISC_InterruptsEnable

Prototype	**void MISC_InterruptsEnable(u16 EnableMask)**
Parameter	"EnableMask" b0: Don't care. b1: Select the interrupts to be enabled.
Return	void
Description	This function is called to enable interrupts, it's a bit-wise operation.

5-2-2-2. MISC_InterruptsDisable

Prototype	**void MISC_InterruptsDisable(u16 DisableMask)**
Parameter	"DisableMask" b0: Don't care. b1: Select the interrupts to be disabled.
Return	void

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

Description	This function is called to disable interrupts, it's a bit-wise operation.

5-2-2-3. MISC_GetIRQ

Prototype	**u16 MISC_GetIRQ(void)**
Parameter	void
Return	IRQ status.
Description	This function will return the value of (INTCR & INTSR) as the IRQ status.

5-2-2-4. MISC_EfuseRead

Prototype	**AX_STATUS MISC_EfuseRead(u8 Addr, u8 *pBuf, u8 Size)**
Parameter	"Addr" The start address of Efuse that you want to read. *pBuf The empty buffer used to store Efuse data. Size Byte size that you want to read.
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is used to read Efuse content.

5-2-2-5. MISC_MotorControlOverride

Prototype	**void MISC_MotorControlOverride(u8 Enable)**
Parameter	"Enable" 0: Disable MC override, MC hardware only operates at EtherCAT OP mode. 1: Enable MC override, MC hardware operates no matter what EtherCAT state is.
Return	void
Description	This function is used to Enable/Disable MC hardware for debugging.

5-2-2-6. MISC_SpiMasterOverride

Prototype	**void MISC_SpiMasterOverride (u8 Enable)**
Parameter	"Enable" 0: Disable SPIM override, SPIM hardware only operate at EtherCAT OP mode. 1: Enable SPIM override, SPIM hardware operate no matter what EtherCAT state is.
Return	void
Description	This function is used to Enable/Disable SPIM hardware for debugging.

5-2-2-7. MISC_ForceFunctionActive

Prototype	**void MISC_ForceFunctionActive (u8 Enable)**
Parameter	"Enable" 0: Function side output active when ESC at OP state. 1: Force function side output active.
Return	void
Description	This function is used to Enable/Disable function side hardware for debugging.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-2. IOWD Module

IOWD module is responsible to control I/O watch-dog or Emergency functions.

Related files of IOWD module are explained as below.

ax58101_iowd.c –
The major file of IOWD sub-module, it is used to handle I/O Watchdog and Emergency function.

ax58101_iowd.h –
The header file of IOWD sub-module.

The IOWD module structures and functions are illustrated as below figure.

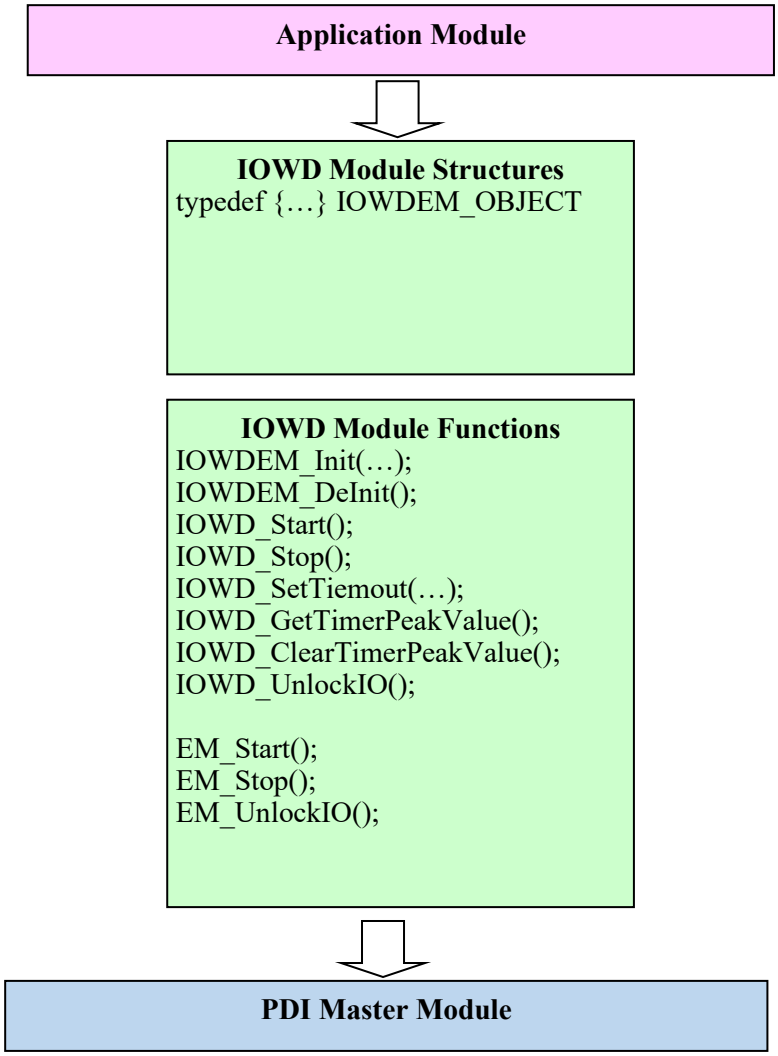

Figure 5-3. IOWD Module Block Diagram

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-3-1. IOWD Module Definition/Macro Description

5-3-1-1. IOWDEM_OBJECT

Prototype	**typedef struct {…} IOWDEM_OBJECT**
Member	u32 InputMonitorMask – b0: Disable monitoring on the input. b1: Enable monitoring on the input. This property is bit-wise operation. u32 InputMonitorPolarity – b0: Set monitored polarity to be low active on the input. b1: Set monitored polarity to be high active on the input. This property is bit-wise operation. u32 OutputActionMask – b0: Disable the action on the output after event triggered. b1: Enable the action on the output after event triggered. This property is bit-wise operation. u32 OutputActionEnable – b0: Select action to be high impedance on the output after event triggered. b1: Select action to be driving out on the output after event triggered. This property is bit-wise operation. u32 OutputActionPolarity – b0: Set polarity to be low active while driving out. b1: Set polarity to be high active while driving out. This property only valid while the accorded OutputActionEnable is b1. u8 EnableSofInputMonitor – 0: Disable monitoring on SOF event. 1: Enable monitoring on SOF event. u8 EnableEscCsInputMonitor – 0: Disable monitoring on ESC chip select event. 1: Enable monitoring on ESC chip select event. u8 EnableFunCsInputMonitor – 0: Disable monitoring of Function chip select event. 1: Enable monitoring of Function chip select event. u8 DetectInputRisingEdge – 0: Detect falling edge on monitored inputs. 1: Detect rising edge on monitored inputs.
Description	This structure is used setup IOWD properties.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-3-2. IOWD Module Function Description

The IOWD module provides following API functions.

5-3-2-1. IOWDEM_Init

Prototype	**AX_STATUS IOWDEM_Init(IOWDEM_OBJECT *pIowdEmObj)**
Parameter	“*pIowdEmObj” Point to an IOWDEM_OBJECT instance object.
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is called to initiate IOWD/EM peripheral hardware module.

5-3-2-2. IOWDEM_DeInit

Prototype	**void IOWDEM_DeInit(void)**
Parameter	void
Return	void
Description	This function is called to de-initiate IOWD/EM peripheral hardware module.

5-3-2-3. IOWD_Start

Prototype	**void IOWD_Start(void)**
Parameter	void
Return	void
Description	This function is used to start IOWD function.

5-3-2-4. IOWD_Stop

Prototype	**void IOWD_Stop(void)**
Parameter	void
Return	void
Description	This function is called to stop IOWD function.

5-3-2-5. IOWD_SetTimeout

Prototype	**void IOWD_SetTimeout(u32 Timeout)**
Parameter	Timeout Specify watchdog timeout value in 10nsec unit.
Return	void
Description	This function is used set watchdog timeout value.

5-3-2-6. IOWD_GetTimePeakValue

Prototype	**u32 IOWD_GetTimePeakValue(void)**
Parameter	void
Return	Time peak value in 10nsec unit.
Description	This function is called to get time peak value.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-3-2-7. IOWD_ClearTimerPeakValue

Prototype	**void IOWD_ClearTimerPeakValue(void)**
Parameter	void
Return	void
Description	This function is called to clear up time peak value.

5-3-2-8. IOWD_UnlockIO

Prototype	**void IOWD_UnlockIO(void)**
Parameter	void
Return	void
Description	This function is called to unlock IOs by IOWD function.

5-3-2-9. EM_Start

Prototype	**void EM_Start(void)**
Parameter	void
Return	void
Description	This function is used to enable Emergency function.

5-3-2-10. EM_Stop

Prototype	**void EM_Stop(void)**
Parameter	void
Return	void
Description	This function is called to stop Emergency function.

5-3-2-11. EM_UnlockIO

Prototype	**void EM_UnlockIO(void)**
Parameter	void
Return	void
Description	This function is called to unlock IOs by Emergency function.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-3. SPIM Module

The SPI master (SPIM) driver module provides an interface and used by upper layer applications to control SPI master hardware by access registers.

As the driver block diagram below, the top module can be temperature/humidity meter or data recorder application and use SPIM driver to trigger SPI master hardware to get the required data from external SPI slaves. The SPIM driver definitions, objects and functions will be described at followed sections. The lowest PDI Master Module is provided by third party and interfaces with MCU via SPI or local bus interface.

ax58101_spim.c –
The major file of SPIM module.

ax58101_spim.h –
The header file of SPIM module.

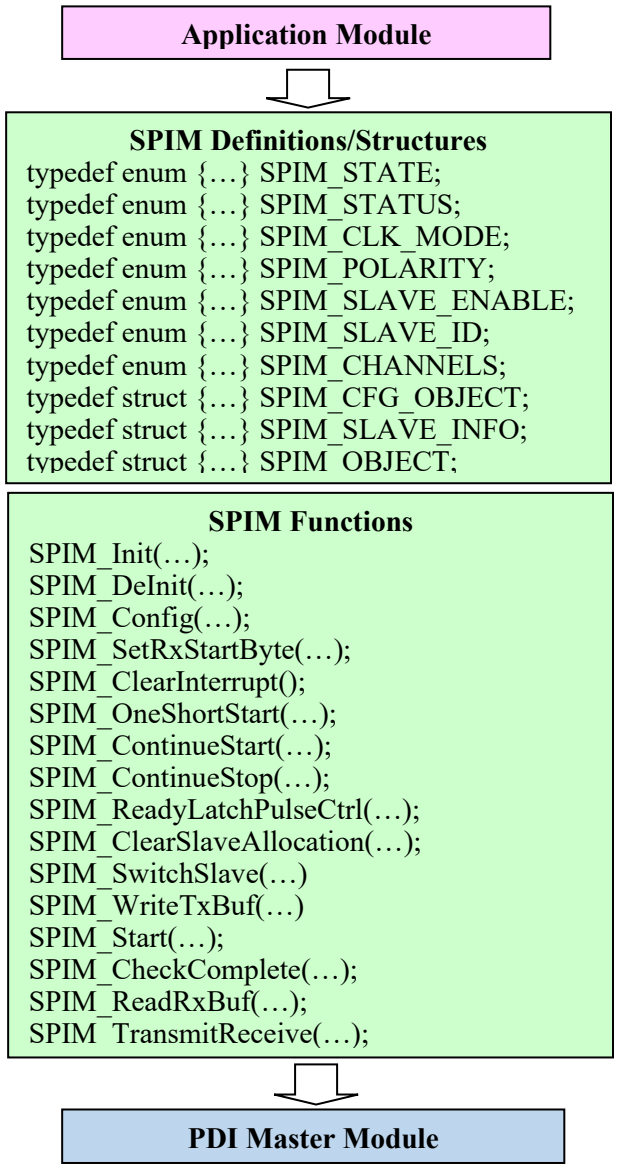

Figure 5-4. SPIM Module Block Diagram

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-4-1. SPIM Module Definition/Macro Description

AX58101(2) SPIM Module supports either One Shot or Continue access mode. Some definitions/structures should be defined/declared. These definitions/structures are declared in the ax58101_spim.h file.
Following subsections describe the enumerations, structures and functions for SPI master use.

5-4-1-1. SPIM_STATE

Prototype	**typedef enum {…} SPIM_STATE**
Member	SPIM_STATE_DE_INITED – Default state and wait to be initiated. SPIM_STATE_INITED – Driver has been initiated. SPIM_STATE_BUSY – Driver in busy. SPIM_STATE_READY – Driver is ready to go. SPIM_STATE_ERR – Some error occurred and shall re-initiate driver.
Description	This defines the state of SPIM driver.

5-4-1-2. SPIM_STATUS

Prototype	**typedef enum {…} SPIM_STATUS**
Member	SPIM_STATUS_OK – No error. SPIM_STATUS_ERR – Un-defined error occurred. SPIM_STATUS_TRG_PULSE_OVERRUN – SPIM trigger pulse overrun. SPIM_STATUS_RX_BUF_OVERRUN – SPIM RX buffer overrun. SPIM_STATUS_TRG_PULSE_TIMEOUT – SPIM trigger pulse timeout. SPIM_STATUS_READY_PULSE_TIMEOUT – SPIM ready pulse timeout. SPIM_STATUS_COMPLETE_TIMEOUT – SPIM completed timeout. SPIM_STATUS_SLAVE_SELECT_ERR – SPIM slave configuration error.
Description	This defines the result of SPIM driver function process.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-4-1-3. SPIM_CLK_MODE

Prototype	**typedef enum {…} SPIM_CLK_MODE**
Member	SCLK_MODE_0 – Set clock mode 0. SCLK_MODE_1 – Set clock mode 1. SCLK_MODE_2 – Set clock mode 2. SCLK_MODE_3 – Set clock mode 3.
Description	This defines the SPIM clock mode that can be used.

5-4-1-4. SPIM_POLARITY

Prototype	**typedef enum {…} SPIM_POLARITY**
Member	SPIM_PolarityActiveLow – Set polarity to active low. SPIM_PolarityActiveHigh – Set polarity to active high.
Description	This defines the polarity of SPIM output signals.

5-4-1-5. SPIM_SLAVE_ENABLE

Prototype	**typedef enum {…} SPIM_SLAVE_ENABLE**
Member	SPIM_EnableSS_0 – Enable slave select 0. SPIM_EnableSS_1 – Enable slave select 1. SPIM_EnableSS_2 – Enable slave select 2. SPIM_EnableSS_3 – Enable slave select 3. SPIM_EnableSS_4 – Enable slave select 4. SPIM_EnableSS_5 – Enable slave select 5. SPIM_EnableSS_6 – Enable slave select 6. SPIM_EnableSS_7 – Enable slave select 7.
Description	This defines the identification of slave select will be enabled.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-4-1-6. SPIM_SLAVE_ID

Prototype	**typedef enum {…} SPIM_SLAVE_ID**
Member	SPIM_SlaveId_0 – Set slave select 0. SPIM_SlaveId_1 – Set slave select 1. SPIM_SlaveId_2 – Set slave select 2. SPIM_SlaveId_3 – Set slave select 3. SPIM_SlaveId_4 – Set slave select 4. SPIM_SlaveId_5 – Set slave select 5. SPIM_SlaveId_6 – Set slave select 6. SPIM_SlaveId_7 – Set slave select 7.
Description	This defines the identification of slave select.

5-4-1-7. SPIM_CHANNELS

Prototype	**typedef enum {…} SPIM_CHANNELS**
Member	SPIM_Channel_0 – Set channel 0. SPIM_Channel_1 – Set channel 1. SPIM_Channel_2 – Set channel 2. SPIM_Channel_3 – Set channel 3. SPIM_Channel_4 – Set channel 4. SPIM_Channel_5 – Set channel 5. SPIM_Channel_6 – Set channel 6. SPIM_Channel_7 – Set channel 7.
Description	This defines the identification of channel.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-4-1-8. SPIM_CFG_OBJECT

Prototype	**typedef struct {…} SPIM_CFG_OBJECT**
Member	SPIM_CLK_MODE ClkMode – Specify which clock mode will be used. u8 LsbXferFirst – 0: LSB transfer first. 1: MSB transfer first u8 LateSampleEnable – 0: Disable late sample. 1: Enable late sample u32 Baudrate – Specify the clock rate, it shall be 100M/N, N=2, 3, …255. u32 FirstClkDelay – Specify the delay time of first valid clock, this value will be set to SPIDBR reg. directly. u32 InterSlaveSelectDelay – Specify the delay time between two slave selects active. u32 ReadyTrgPulseTimeout – Specify the timeout value that wait for trigger pulse. u32 LatchPulseGap – Specify the delay time of Slave select to DAC latch. u32 LatchPulseWidth – Specify the latch pulse width. SPIM_POLARITY TrgPulsePolarity – Specify the polarity of trigger pulse input. SPIM_POLARITY ReadyLatchPulsePolarity – Specify the polarity of latch pulse output. SPIM_POLARITY InterruptPolarity – Specify the polarity of SPI interrupt. u8 InterruptModeEnable – 0: Disable interrupt mode. 1: Enable interrupt mode. u8 RxBufHandShakeEnable – 0: Disable RX buffer handshake mode. 1: Enable RX buffer handshake mode. u8 TrgPulseEnable – 0: Disable the waiting for external trigger pulse. 1: Enable the waiting for external trigger pulse. u8 ReadyOrLatchPulseEnable – ADC Mode Enabled: 0: Disable waiting for external ready pulse. 1: Enable waiting for external ready pulse. DAC Mode Enabled: 0: Disable generation of latch pulse. 1: Enable generation of latch pulse. u8 DacModeEnable – 0: Enable ADC mode. 1: Enable DAC mode. u8 AutoKeepSlaveSelectEnable – 0: Disable auto keep slave select function. 1: Enable auto keep slave select function.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

	u8 ContinueXferEnable – 0: Disable continue transfer mode. 1: Enable continue transfer mode. u8 ReadyCombinedWithMiso – 0: Don't accept ready pulse combined with MISO line. 1: Accept ready pulse combined with MISO line. u8 ExtDecodeEnable – 0: Disable encoder mode, the number of slave selects can up to 4. 1: Enable encoder mode, shall additional decoder and extend slave selects up to 8.
Description	This object is used to configure SPIM properties.

5-4-1-9. SPIM_SLAVE_INFO

Prototype	**typedef struct {…} SPIM_SLAVE_INFO**
Member	u8 InUse – 0: This transfer is idle. 1: This transfer is in used. u16 TxBufAddr – Specify the start address of TX buffer for the transfer. u16 RxBufAddr – Specify the start address of RX buffer for the transfer. u8 TxLen – Specify the TX data length for the transfer. u8 RxLen – Specify the RX data length for the transfer. u8 RxStartByte – Specify the start byte in RX buffer for the transfer. u8 StartChannel – Specify the start channel of the transfer. u8 UsedChannels – Specify how channels in used for the transfer.
Description	This object is used to allocate multiple slave transfers.

5-4-1-10. SPIM_OBJECT

Prototype	**typedef struct {…} SPIM_OBJECT**
Member	SPIM_STATE State – SPIM state. u8 SlaveNum – Specify the number of slaves will be transfer. SPIM_SLAVE_INFO SlaveInfo[] – Used to allocate multiple slave transfers. u8 CurrSlaveId – Record which slave id in process. u8 SlaveIdOfChannel[] – Record the mapping of slave id and channel. u8 ChannelNumInUse – Record how many channels will be used for transfer. SPIM_CFG_OBJECT Cfg – Used to configure the SPIM properties.
Description	This object is used as SPIM instance object and responsible of configuration and controlling.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-4-2. SPIM Module Function Description

The SPIM module provides the following SPI master API functions.

5-4-2-1. SPIM_Init

Prototype	**AX_STATUS SPIM_Init(SPIM_OBJECT *pSpimObj)**
Parameter	"*pSpimObj" This is a pointer to SPIM_OBJECT instance object.
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is called to initiate SPI master hardware in idle state.

5-4-2-2. SPIM_DeInit

Prototype	**void SPIM_DeInit(SPIM_OBJECT *pSpimObj)**
Parameter	"*pSpimObj" This is a pointer to SPIM_OBJECT instance object.
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is called to stop SPI master signal outputs and force SPI master output pins in high impedance state.

5-4-2-3. SPIM_Config

Prototype	**AX_STATUS SPIM_Config(SPIM_OBJECT *pSpimObj)**
Parameter	"*pSpimObj" This is a pointer to SPIM_OBJECT instance object.
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is will configure SPIM hardware based on SPIM_CFG_OBJECT Cfg member content.

5-4-2-4. SPIM_SetRxStartByte

Prototype	**void SPIM_SetRxStartByte(SPIM_OBJECT *pSpimObj, u8 StartByte)**
Parameter	"*pSpimObj" This is a pointer to SPIM_OBJECT instance object. "StartByte" Specify the start byte of RX data will be stored into buffer.
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is used to set start data byte that will be received into RX buffer.

5-4-2-5. SPIM_ClearInterrupt

Prototype	**void SPIM_ClearInterrupt(void)**
Parameter	void
Return	void
Description	This function is used to clear SPIM complete flag and interrupt flag.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-4-2-6. SPIM_OneShotStart

Prototype	**AX_STATUS SPIM_OneShotStart(SPIM_OBJECT *pSpimObj)**
Parameter	"*pSpimObj" This is a pointer to SPIM_OBJECT instance object.
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is used to start one-time single transfer or multiple transfers.

5-4-2-7. SPIM_ContinueStart

Prototype	**AX_STATUS SPIM_ContinueStart(SPIM_OBJECT *pSpimObj)**
Parameter	"*pSpimObj" This is a pointer to SPIM_OBJECT instance object.
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is used to start continue single transfer or multiple transfers.

5-4-2-8. SPIM_ContinueStop

Prototype	**AX_STATUS SPIM_ContinueStop(SPIM_OBJECT *pSpimObj)**
Parameter	"*pSpimObj" This is a pointer to SPIM_OBJECT instance object.
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is used to terminate continue transfer(s).

5-4-2-9. SPIM_ReadyLatchPulseCtrl

Prototype	**void SPIM_ReadyLatchPulseCtrl(SPIM_OBJECT *pSpimObj, u8 Enable)**
Parameter	"*pSpimObj" This is a pointer to SPIM_OBJECT instance object.
Return	"Enable" If ADC Mode is Enabled: 0: Disable ready pulse waiting. 1: Enable ready pulse waiting. If DAC Mode is Enabled: 0: Disable latch pulse generation. 1: Enable latch pulse generation.
Description	This function is used to disable/enable ready pulse waiting or latch pulse generation.

5-4-2-10. SPIM_ClearSlaveAllocation

Prototype	**AX_STATUS SPIM_ClearSlaveAllocation(SPIM_OBJECT *pSpimObj)**
Parameter	"*pSpimObj" This is a pointer to SPIM_OBJECT instance object.
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is used to reset the allocation of slave transfers.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-4-2-11. SPIM_SwitchSlave

Prototype	**AX_STATUS SPIM_SwitchSlave(SPIM_OBJECT *pSpimObj, SPIM_SLAVE_ID SlaveId)**
Parameter	"*pSpimObj" This is a pointer to SPIM_OBJECT instance object. "SPIM_SLAVE_ID SlaveId" Select the slave id which will be allocated transfer.
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is used to change slave for transfer allocation.

5-4-2-12. SPIM_WriteTxBuf

Prototype	**s32 SPIM_WriteTxBuf(SPIM_OBJECT *pSpimObj, u8 *pData, u8 Len)**
Parameter	"*pSpimObj" This is a pointer to SPIM_OBJECT instance object. "u8 *pData" A pointer to the data will be transmitted. "u8 Len" Specify the byte length of transmit data.
Return	>=0 The actual byte length of data be filled into TX buffer. <0 Fail.
Description	This function is used to fill data into TX buffer.

5-4-2-13. SPIM_Start

Prototype	**AX_STATUS SPIM_Start(SPIM_OBJECT *pSpimObj)**
Parameter	"*pSpimObj" This is a pointer to SPIM_OBJECT instance object.
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is used to start the one time or continue transfer(s) based on configuration.

5-4-2-14. SPIM_CheckComplete

Prototype	**AX_STATUS SPIM_CheckComplete(SPIM_OBJECT *pSpimObj, u32 Timeout)**
Parameter	"*pSpimObj" This is a pointer to SPIM_OBJECT instance object. "u32 Timeout" The time duration to wait for transfer complete, unit in msec.
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is used to wait transfer complete.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-4-2-15. SPIM_ReadRxBuf

Prototype	**s32 SPIM_ReadRxBuf(SPIM_OBJECT *pSpimObj, u8 *pBuf, u8 Size)**
Parameter	"*pSpimObj" This is a pointer to SPIM_OBJECT instance object. "u8 *pBuf" Specify an empty buffer to store received data. "u8 Size" Specify how many bytes to be read from RX buffer.
Return	>=0 Actual bytes have been copied into pBuf from RX buffer. <0 Fail.
Description	This function is used to get received data from RX buffer.

5-4-2-16. SPIM_TransmitReceive

Prototype	**s32 SPIM_TransmitReceive(SPIM_OBJECT *pSpimObj, u8 *pData, u8 *pBuf, u32 Len)**
Parameter	"*pSpimObj" This is a pointer to SPIM_OBJECT instance object. "u8 *pData" The data buffer will be transmitted. "u8 *pBuf" The empty buffer will be stored received data. "u8 Len" The data length of transfer.
Return	>=0 Actual bytes have been copied into pBuf from RX buffer. <0 Fail.
Description	This function is used to start a single SPI transfer.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-4. MC Module

The AX58101(2) Motor Control (MC) driver modules are implemented to operate the MC hardware interface by controlling MC registers, The MC module is separated in two parts, PWM driver and STEP driver. The PWM driver is used to configure MC hardware in PWM mode and trigger hardware to output signals on PWM channel 1~3 high side/low side, the period, high pulse width and other features can be configured.

The STEP driver also provides several APIs to initiate and start STEP hardware, the gap time of two step pulses, high pulse width, target pulse number and direction can be configured.
Related files of MC module are explained as below.

ax58101_pwm.c –
The major file of PWM sub-module, it handles the interrupt service routine.

ax58101_pwm.h –
The header file of the PWM sub-module.

ax58101_step.c –
The major file of STEP sub-module, it handles the interrupt service routine.

ax58101_step.h –
The header file of the STEP sub-module.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

The MC Module are illustrated as below figure.

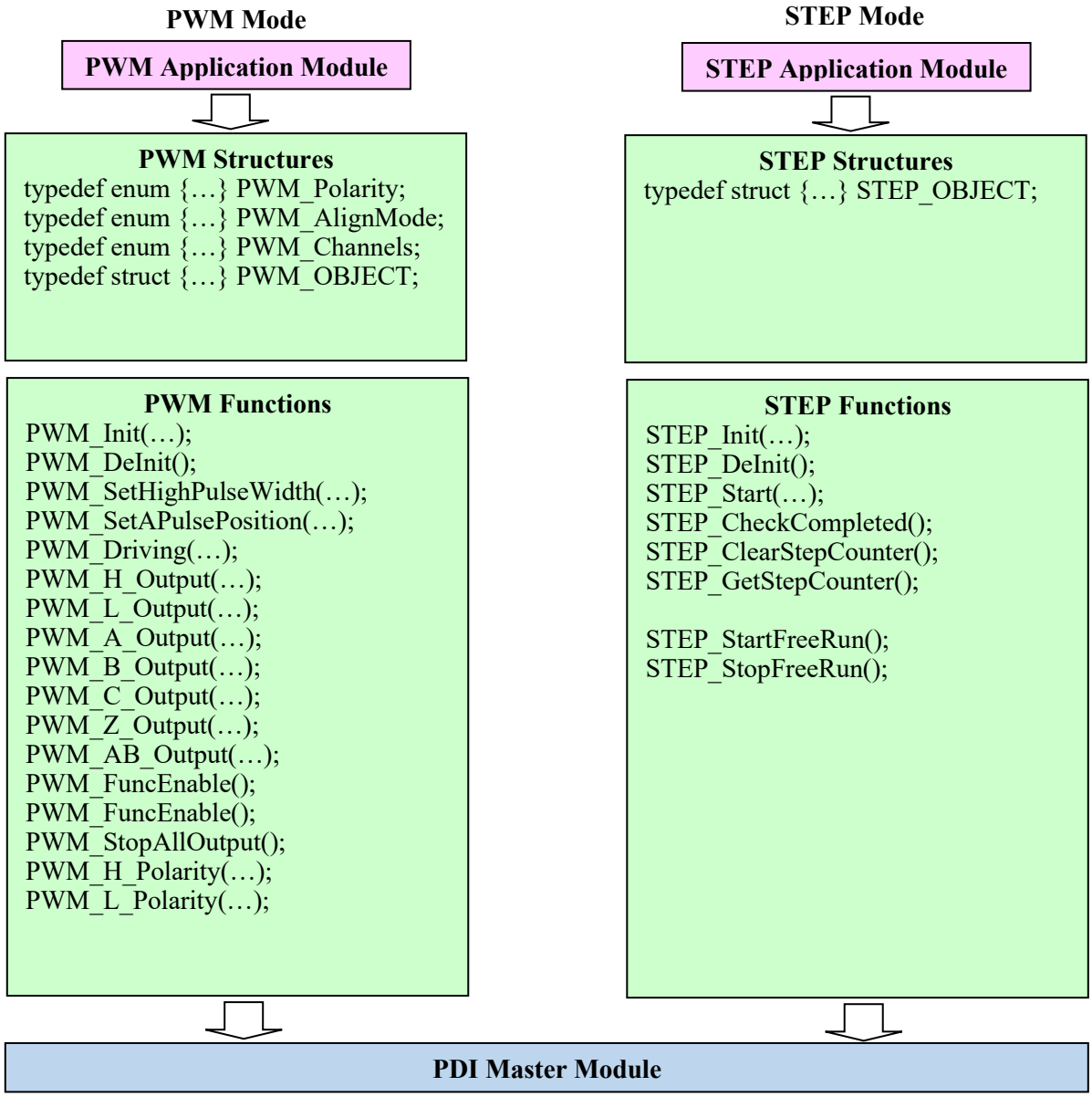

Figure 5-5. MC Module Block Diagram

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-5-1. MC Module Definition/Macro Description

AX58101(2) MC Module driver supports either PWM or STEP mode. Some definitions/structures should be defined/declared. These definitions/structures are declared in the ax58101_pwm.h and ax58101_step.h file.

5-5-2. PWM Module Definition/Macro Description

5-5-2-1. PWM_Polarity

Prototype	**typedef enum {…} PWM_Polarity**
Member	PWM_PolarityPositive – Set PWM signals to high polarity. PWM_PolarityNegative – Set PWM signals to low polarity.
Description	This definition is used to define PWM signals polarity.

5-5-2-2. PWM_AlignMode

Prototype	**typedef enum {…} PWM_AlignMode**
Member	PWM_LeftAlign – Set PWM signals to left alignment. PWM_RightAlign – Set PWM signals to right alignment. PWM_CenterAlign – Set PWM signals to center alignment.
Description	This definition is used to define PWM alignment modes.

5-5-2-3. PWM_Channels

Prototype	**typedef enum {…} PWM_Channels**
Member	PWM_Channel_1 – Specify channel 1. PWM_Channel_2 – Specify channel 2. PWM_Channel_3 – Specify channel 3.
Description	This definition is used to define PWM channels.

5-5-2-4. PWM_OBJECT

Prototype	**typedef struct {…} PWM_OBJECT**
Member	u16 PeriodCycle – Specify the period time of PWM signal will be generated, unit in 10nsec. u16 PwmhBbmTime – Specify the break before make time of PWM high side signal, unit in 10nsec. u16 PwmlBbmTime – Specify the break before make time of PWM low side signal, unit in 10nsec. PWM_Polarity PwmhPolarity[] –

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

	Specify the polarity of PWM high side signal. PWM_Polarity PwmlPolarity[] – Specify the polarity of PWM low side signal. u16 PwmShiftTime[] – Specify the shift time of PWM signal will be generated, unit in 10nsec. PWM_AlignMode AlignMode – Specify the alignment mode of PWM signals. u16 AbczHighPulseWidth – Specify the pulse width of A/B/C/Z signals. PWM_Polarity AbczPulsePolarity – Specify the polarity of A/B/C/Z signals. u16 A_PulseTrigPosition – Specify the trigger position of A pulse, unit in 10nsec. u16 B_PulseTrigPosition – Specify the trigger position of B pulse, unit in 10nsec.
Description	This object is used as a PWM module instance for further configuration or control.

5-5-3. STEP Module Definition/Macro Description

5-5-3-1. STEP_OBJECT

Prototype	**typedef enum {…} STEP_OBJECT**
Member	u32 GapTime – Specify the gap time of two STEP pulses, unit in 10nsec. u16 HighPulseWidth – Specify the high pulse width of STEP pulse will be generated, unit in 10nsec. u16 DirXferDelay – Specify the transfer delay time of direction signal, unit in 10nsec. STEP_Polarity DirectionSignalPolarity – Specify the polarity of direction signal. STEP_Polarity StepPulsePolarity – Specify the polarity of STEP pulse signal.
Description	This object is used as a STEP module instance for further configuration or control.

5-5-4. PWM Mode Function Description

5-5-4-1. PWM_Init

Prototype	**AX_STATUS PWM_Init(PWM_OBJECT *pPwmObj)**
Parameter	“*pPwmObj” This is a pointer to PWM_OBJECT instance object.
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is called to initiate PWM hardware and driving PWM output pins in idle state.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-5-4-2. PWM_DeInit

Prototype	**void PWM_DeInit(void)**
Parameter	void
Return	void
Description	This function is called to stop PWM signal outputs and force PWM output pins in high impedance state.

5-5-4-3. PWM_SetHighPulseWidth

Prototype	**void PWM_SetHighPulseWidth(PWM_OBJECT *pPwmObj, PWM_Channels Channel, u16 HighPulseWidth)**
Parameter	"*pPwmObj" This is a pointer to PWM_OBJECT instance object. "Channel" Specify a channel number that has defined in **PWM_Channels** enumeration. "HighPulseWidth" Specify the high pulse width.
Return	None
Description	This function is used to set high pulse width.

5-5-4-4. PWM_SetAPulsePosition

Prototype	**void PWM_SetAPulsePosition(PWM_OBJECT *pPwmObj, u16 Position)**
Parameter	"*pPwmObj" This is a pointer to PWM_OBJECT instance object. "Position" Specify the position of A pulse in 10ns unit.
Return	void
Description	This function is used to specify the position where the A pulse will be generated.

5-5-4-5. PWM_Driving

Prototype	**void PWM_Driving(PWM_Channels Channel, u8 EnableDriving)**
Parameter	"Channel" Specify a channel number that has defined in PWM_Channels enumeration. "EnableDriving" 0: Stop PWM driving. 1: Start PWM driving.
Return	None
Description	This function is used to Start/Stop PWM driving.

5-5-4-6. PWM_H_Output

Prototype	**void PWM_H_Output(PWM_Channels Channel, u8 EnableOutput)**
Parameter	Channel Specify a channel number that has defined in PWM_Channels enumeration. EnableOutput 0: Disable PWM high side pulse output. 1: Enable PWM high side pulse output.
Return	void
Description	This function is called to disable/enable PWM pulse output on high side.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-5-4-7. PWM_L_Output

Prototype	**void PWM_L_Output(PWM_Channels Channel, u8 EnableOutput)**
Parameter	"Channel" Specify a channel number that has defined in **PWM_Channels** enumeration. "EnableOutput" 0: Disable PWM low side pulse output. 1: Enable PWM low side pulse output.
Return	void
Description	This function is called to disable/enable PWM pulse output on low side.

5-5-4-8. PWM_A_Output

Prototype	**void PWM_A_Output(u8 EnableOutput)**
Parameter	"EnableOutput" 0: Disable A pulse output. 1: Enable A pulse output.
Return	void
Description	This function is used to disable/enable A pulse output.

5-5-4-9. PWM_B_Output

Prototype	**void PWM_B_Output(u8 EnableOutput)**
Parameter	"EnableOutput" 0: Disable B pulse output. 1: Enable B pulse output.
Return	void
Description	This function is used to disable/enable B pulse output.

5-5-4-10. PWM_C_Output

Prototype	**void PWM_C_Output(u8 EnableOutput)**
Parameter	"EnableOutput" 0: Disable C pulse output. 1: Enable C pulse output.
Return	void
Description	This function is used to disable/enable C pulse output.

5-5-4-11. PWM_Z_Output

Prototype	**void PWM_Z_Output(EnableOutput)**
Parameter	"EnableOutput" 0: Disable Z pulse output. 1: Enable Z pulse output.
Return	void
Description	This function is used to disable/enable Z pulse output.

5-5-4-12. PWM_AB_Output

Prototype	**void PWM_AB_Output(EnableOutput)**
Parameter	"EnableOutput" 0: Disable AB pulse output. 1: Enable AB pulse output.
Return	void
Description	This function is used to disable/enable AB pulse output.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-5-4-13. PWM_FuncEnable

Prototype	**void PWM_FuncEnable(void)**
Parameter	void
Return	void
Description	This function is used to enable PWM function.

5-5-4-14. PWM_FuncDisable

Prototype	**void PWM_FuncDisable(void)**
Parameter	void
Return	void
Description	This function is used to disable PWM function.

5-5-4-15. PWM_StopAllOutput

Prototype	**void PWM_StopAllOutput(void)**
Parameter	void
Return	void
Description	This function is called to stop all output signals, which includes PWM_H/L 1~3, A/B/C/Z outputs and return to idle state.

5-5-4-16. PWM_H_Polarity

Prototype	**void PWM_H_Polarity(PWM_Channels Channel, PWM_Polarity HighPolarity)**
Parameter	"Channel" Specify a channel number that has defined in **PWM_Channels** enumeration. "HighPolarity" Specify high side polarity that has defined in **PWM_Polarity** enumeration.
Return	void
Description	This function is used to change high side polarity.

5-5-4-17. PWM_L_Polarity

Prototype	**void PWM_L_Polarity(PWM_Channels Channel, PWM_Polarity LowPolarity)**
Parameter	"Channel" Specify a channel number that has defined in **PWM_Channels** enumeration. "HighPolarity" Specify high side polarity that has defined in **PWM_Polarity** enumeration.
Return	void
Description	This function is used to change low side polarity.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-5-5. STEP Mode Function Description

5-5-5-1. STEP_Init

Prototype	**AX_STATUS STEP_Init(STEP_OBJECT *pStepObj)**
Parameter	"*pStepObj" This is a pointer to STEP_OBJECT instance object.
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is called to initiate STEP hardware and driving STEP output pins in idle state.

5-5-5-2. STEP_DeInit

Prototype	**AX_STATUS STEP_DeInit(void)**
Parameter	void
Return	void
Description	This function is called to stop STEP signal outputs and force STEP output pins in high impedance state.

5-5-5-3. STEP_Start

Prototype	**AX_STATUS STEP_Start(s32 TargetStepNumber)**
Parameter	"TargetStepNumber" Specify the step pulse number will be generated, note the most significant bit (msb) is used to control direction signal output.
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is used to start STEP signals output with specific step number.

5-5-5-4. STEP_CheckCompleted

Prototype	**AX_STATUS STEP_CheckCompleted(void)**
Parameter	void
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is used to check the STEP pulses output has done or not.

5-5-5-5. STEP_ClearStepCounter

Prototype	**void STEP_ClearStepCounter(void)**
Parameter	void
Return	void
Description	This function is used to clear STEP counter register.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-5-5-6. STEP_GetStepCounter

Prototype	**s32 STEP_GetStepCounter(void)**
Parameter	void
Return	32bits step counter, the most significant bit indicates the direction signal status.
Description	This function is called to get STEP counter register value.

5-5-5-7. STEP_StartFreeRun

Prototype	**AX_STATUS STEP_StartFreeRun(u8 Direction)**
Parameter	"Direction" Set direction in Step Target Number Register (STNLR/STNHR).
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is called to start free run mode.

5-5-5-8. STEP_StopFreeRun

Prototype	**AX_STATUS STEP_StopFreeRun(void)**
Parameter	void
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is called to stop free run mode.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-5. ENC Module

The Encoder module (ENC) is part of AX58101(2) Extended Functions, it supports four encoder modes, they are ABZ, CW/CCW, CLK/DIR and HALL modes. The ENC driver is responsible for configure ENC hardware to be one of four modes at one time and get back measured information from external encoder source.

Related files of Encoder module are explained as below.

ax58101_enc.c –
The major file of ENC module.

ax58101_enc.h –
The header file of ENC module.

The ENC module architecture has shown as below, the general functions can be used regardless which mode has been configured.

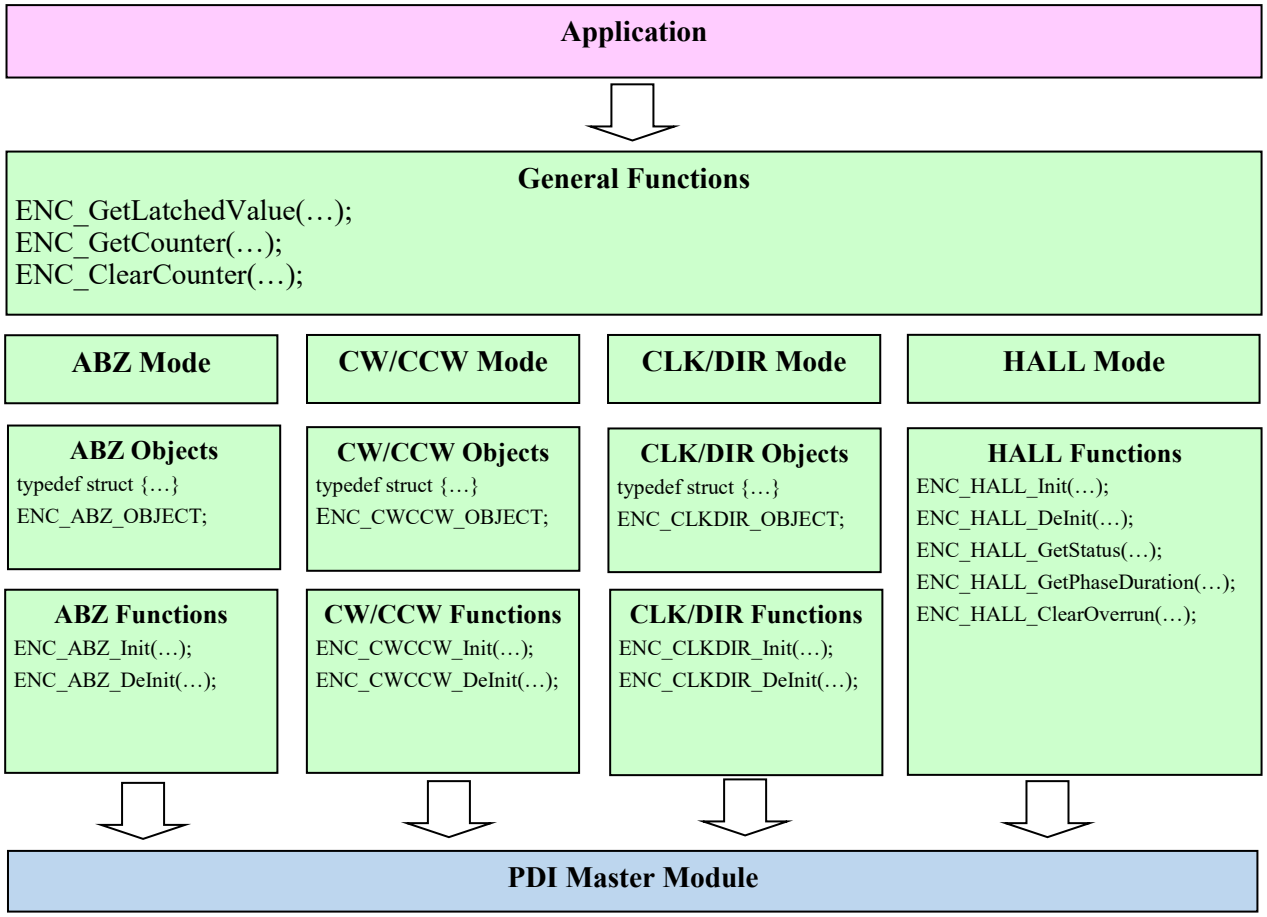

Figure 5-6. ENC Module Block Diagram

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-6-1. ENC Module Definition/Macro Description

AX58101(2) ENC Module driver supports ABZ, CW/CCW, CLK/DIR and HALL mode. Some definitions/structures should be defined/declared. These definitions/structures are declared in the ax58101_enc.h file.

5-6-2. ABZ Module Definition/Macro Description

Following subsections describe the enumerations, structures and functions for ABZ mode use.

5-6-2-1. ENC_ABZ_RATIO

Prototype	**typedef enum {…} ENC_ABZ_RATIO**
Member	ENC_ABZ_RATIO_4X – Set ratio to 4x. ENC_ABZ_RATIO_2X – Set ratio to 2x. ENC_ABZ_RATIO_3X – Set ratio to 3x.
Description	This definition is used to define ratio for ABZ mode.

5-6-2-2. ENC_CLR_EVENT_AT_IZ_EDGE

Prototype	**typedef enum {…} ENC_CLR_EVENT_AT_IZ_EDGE**
Member	ENC_NO_CLR_EVENT – Disable cleared event. ENC_GEN_CLR_AT_IZ_ASSERT – Trigger cleared event at asserting of internal Z. ENC_GEN_CLR_AT_IZ_DEASSERT – Trigger cleared event at de-asserting of internal Z. ENC_GEN_CLR_AT_BOTH – Trigger cleared event at asserting and de-asserting of internal Z.
Description	This definition is used to select which edge of internal Z that cleared event will be triggered.

5-6-2-3. ENC_ABZ_OBJECT

Prototype	**typedef struct {…} ENC_ABZ_OBJECT**
Member	u8 A_InputPolarityPositive – 0: Set input A polarity to be negative. 1: Set input A polarity to be positive. u8 B_InputPolarityPositive – 0: Set input B polarity to be negative. 1: Set input B polarity to be positive. u8 Z_InputPolarityPositive – 0: Set input Z polarity to be negative. 1: Set input Z polarity to be positive. u8 IgnoreAB_Inputs – 0: Considering A and B inputs for clear event generation. 1: Ignore A and B inputs for clear event generation.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

	ENC_ABZ_RATIO Ratio – Specify the ratio of ABZ encoder, please refer to ENC_ABZ_RATIO enumeration definition. ENC_CLR_EVENT_AT_IZ_EDGE ClearEventGenerateEdge – Specify the triggered edge of cleared event, please refer to ENC_CLR_EVENT_AT_IZ_EDGE enumeration definition. u8 EnableDecimalMode – 0: Disable decimal mode for internal counting. 1: Enable decimal mode for internal counting. u16 ConstantFractional – Specify fractional counting base. u16 ConstantInteger – Specify integer counting base.
Description	This object is used as an ABZ encoder instance for further configuration or control.

5-6-3. CW/CCW Module Definition/Macro Description

Following subsections describe the enumerations, structures and functions for CW/CCW mode use.

5-6-3-1. ENC_CWCCW_OBJECT

Prototype	**typedef struct {…} ENC_CWCCW_OBJECT**
Member	u8 EnableDecimalMode – 0: Disable decimal mode for internal counting. 1: Enable decimal mode for internal counting. u16 ConstantFractional – Specify fractional counting base. u16 ConstantInteger – Specify integer counting base.
Description	This object is used as a CW/CCW encoder instance for further configuration or control.

5-6-4. CLK/DIR Module Definition/Macro Description

Followed sections describes the enumerations, structures and functions for CLK/DIR mode use.

5-6-4-1. ENC_CLKDIR_OBJECT

Prototype	**typedef struct {…} ENC_CLKDIR_OBJECT**
Member	u8 EnableDecimalMode – 0: Disable decimal mode for internal counting. 1: Enable decimal mode for internal counting. u16 ConstantFractional – Specify fractional counting base. u16 ConstantInteger – Specify integer counting base.
Description	This object is used as a CLK/DIR encoder instance for further configuration or control.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-6-5. ABZ Mode Function Description

5-6-5-1. ENC_ABZ_Init

Prototype	**AX_STATUS ENC_ABZ_Init(ENC_ABZ_OBJECT *pAbzObj)**
Parameter	"*pAbzObj" This is a pointer to ENC_ABZ_OBJECT instance object.
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is called to initiate ENC hardware in ABZ mode.

5-6-5-2. ENC_ABZ_DeInit

Prototype	**void ENC_ABZ_DeInit(void)**
Parameter	void
Return	void
Description	This function is called to de-initiate ENC hardware.

5-6-6. CW/CCW Mode Function Description

5-6-6-1. ENC_CWCCW_Init

Prototype	**AX_STATUS ENC_CWCCW_Init(ENC_CWCCW_OBJECT *pCwccwObj)**
Parameter	"*pCwCCwObj" This is a pointer to ENC_CWCCW_OBJECT instance object.
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is called to initiate ENC hardware in CW/CCW mode.

5-6-6-2. ENC_CWCCW_DeInit

Prototype	**void ENC_CWCCW_DeInit(void)**
Parameter	void
Return	void
Description	This function is called to de-initiate ENC hardware.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-6-7. CLKDIR Mode Function Description

5-6-7-1. ENC_CLKDIR_Init

Prototype	**AX_STATUS ENC_CLKDIR_Init(ENC_CLKDIR _OBJECT *pClkdirObj)**
Parameter	"*pClkdirObj" This is a pointer to ENC_CLKDIR_OBJECT instance object.
Return	Return AX_STATUS_OK indicates the operation is successful, otherwise return AX_STATUS_ERR.
Description	This function is called to initiate ENC hardware in CLK/DIR mode.

5-6-7-2. ENC_CLKDIR_DeInit

Prototype	**void ENC_CLKDIR_DeInit(void)**
Parameter	void
Return	void
Description	This function is called to de-initiate ENC hardware.

5-6-8. HALL Mode Function Description

5-6-8-1. ENC_HALL_Init

Prototype	**void ENC_HALL_Init(void)**
Parameter	void
Return	void
Description	This function is called to initiate ENC hardware in HALL mode.

5-6-8-2. ENC_HALL_DeInit

Prototype	**void ENC_HALL_DeInit(void)**
Parameter	void
Return	void
Description	This function is called to de-initiate ENC hardware.

5-6-8-3. ENC_HALL_GetStatus

Prototype	**void ENC_HALL_GetStatus(u8 *pCurrStatus, u8 *pKeepStatus, u8 *pOverrun)**
Parameter	"*pCurrStatus" Used to contain 3bits HALL current status. "*pKeepStatus" Used to contain 3bits HALL keep status. "*pOverrun" 0: No phase overrun. 1: Phase overrun occurred.
Return	void
Description	This function is used to get HALL encoder status.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

5-6-8-4. ENC_HALL_GetPhaseDuration

Prototype	**u32 ENC_HALL_GetPhaseDuration(void)**
Parameter	void
Return	32bits phase duration time in 10ns unit.
Description	This function is used to get HALL phase duration time.

5-6-8-5. ENC_HALL_ClearOverrun

Prototype	**void ENC_HALL_ClearOverrun(void)**
Parameter	void
Return	void
Description	If HALL duration overrun occurred, calling this function to clear overrun state.

5-6-9. ENC General Function Description

5-6-9-1. ENC_GetLatchedValue

Prototype	**u32 ENC_GetLatchedValue(void)**
Parameter	void
Return	32bits latched value.
Description	This function is used to get latched value.

5-6-9-2. ENC_GetCounter

Prototype	**u32 ENC_GetCounter(void)**
Parameter	void
Return	32bits counter value.
Description	This function is used to get counter value.

5-6-9-3. ENC_ClearCounter

Prototype	**void ENC_ClearCounter(void)**
Parameter	void
Return	void
Description	This function is used to clear counter register.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

Appendix A. PHY Enhancement Solution

This section indicates AX58101(2) PHY enhancement solution to prevent possible Ethernet link unstable state on some special test environments. We strongly recommend the designer to implement this solution in the SOC hardware driver to avoid this kind of possible potential issue. The designer can also refer to below source code driver for more details.

```
#if (ESC_PHY_ENHANCEMENT)
/**
 * @brief  This function operates to read PHY register.
 * @param  PhyAddr      PHY address.
 * @param  RegAddr     Address of the PHY register.
 * @param  *pBuf        Point to the empty buffer will be stored data.
 * @param  Size         Byte size that will be read.
 * @retval Operation result, return zero indicates the operation is successful, otherwise return a error code.
 */
INT32 HW_MiiPhyRead(UINT8 PhyAddr, UINT8 RegAddr, UINT16 *pBuf, UINT8 Size)
{
        UINT8 tmp8;
        UINT16 tmp16;

        /* Check access right */
        HW_EscRead((MEM_ADDR *)&tmp16, 0x0510, 2);
        if ((tmp16 & 0x8002) == 0)
        {
                return -1;
        }

        /* Get access right to PDI */
        tmp8 = 1;
        HW_EscWrite((MEM_ADDR *)&tmp8, 0x0517, 1);

        /* Set PHY address */
        HW_EscWrite((MEM_ADDR *)&PhyAddr, 0x0512, 1);

        while (Size--)
        {
                /* Set register address */
                HW_EscWrite((MEM_ADDR *)&RegAddr, 0x0513, 1);

                /* Set command */
                tmp16 |= 0x0100;
                tmp16 &= ~0x2000;
                HW_EscWrite((MEM_ADDR *)&tmp16, 0x0510, 2);

                /* Check complete */
                do {
                        HW_EscRead((MEM_ADDR *)&tmp16, 0x0510, 2);
```


Copyright (C) 2025 Reserved by ASIX Electronics Corporation

```
            } while (tmp16 & 0x8300);

            /* Check error */
            if (tmp16 & 0x6000)
            {
                return -2;
            }

            /* Read data */
            HW_EscRead((MEM_ADDR *)pBuf, 0x0514, 2);

            RegAddr++;
            pBuf++;
        }

        return 0;
}

/**
 * @brief  This function operates to write PHY register.
 * @param  PhyAddr      PHY address.
 * @param  RegAddr      Address of the PHY register.
 * @param  *pData       Point to the data buffer will be written.
 * @param  Len          Byte size that will be written.
 * @retval Operation result, return zero indicates the operation is successful, otherwise return a error
code.
 */
INT32 HW_MiiPhyWrite(UINT8 PhyAddr, UINT8 RegAddr, UINT16 *pData, UINT8 Len)
{
        UINT8 tmp8;
        UINT16 tmp16;

        /* Check access right */
        HW_EscRead((MEM_ADDR *)&tmp16, 0x0510, 2);
        if ((tmp16 & 0x8002) == 0)
        {
            return -1;
        }

        /* Get access right to PDI */
        tmp8 = 1;
        HW_EscWrite((MEM_ADDR *)&tmp8, 0x0517, 1);

        /* Set PHY address */
        HW_EscWrite((MEM_ADDR *)&PhyAddr, 0x0512, 1);

        while (Len--)
        {
            /* Set register address */
            HW_EscWrite((MEM_ADDR *)&RegAddr, 0x0513, 1);
```


Copyright (C) 2025 Reserved by ASIX Electronics Corporation

```
                /* Write data */
                HW_EscWrite((MEM_ADDR *)pData, 0x0514, 2);

                /* Set command */
                tmp16 |= 0x0201;
                HW_EscWrite((MEM_ADDR *)&tmp16, 0x0510, 2);

                /* Check complete */
                do {
                        HW_EscRead((MEM_ADDR *)&tmp16, 0x0510, 2);
                } while (tmp16 & 0x8300);

                /* Check error */
                if (tmp16 & 0x4000)
                {
                        return -2;
                }

                RegAddr++;
                pData++;
        }

        return 0;
}

/**
 * @brief  This function operates to do PHY enhancement.
 */
void HW_PhyEnhancement(void)
{
        UINT8 i;
        UINT16 tmp16;
        static UINT16 bmsr[2][2] = {{0, 0}, {0, 0}};

        HW_DEBUG *pDbgCnt;
        pDbgCnt = HW_GetDebugCounter();

        for (i=0; i<2; i++)
        {
                /* Set page 0 */
                tmp16 = 0;
                HW_MiiPhyWrite(i, 0x1f, &tmp16, 1);

                HW_MiiPhyRead(i, 0x01, &(bmsr[i][0]), 1);
                if ((bmsr[i][1] & 0x0020)==0 && (bmsr[i][0] & 0x0020)!=0)
                {
                        HW_MiiPhyRead(i, 0x1c, &tmp16, 1);
```


Copyright (C) 2025 Reserved by ASIX Electronics Corporation

```
                        if (tmp16)
                        {
                                HW_MiiPhyRead(i, 0x1e, &tmp16, 1);
                                tmp16 |= 0x4000;
                                HW_MiiPhyWrite(i, 0x1e, &tmp16, 1);

                                HW_MiiPhyRead(i, 0x00, &tmp16, 1);
                                tmp16 |= 0x1200;
                                HW_MiiPhyWrite(i, 0x00, &tmp16, 1);

                                pDbgCnt->PhyEnhancementCnt[i]++;
                        }
                }
                bmsr[i][1] = bmsr[i][0];
        }

}
#endif

int main()
{
        …..
        …..
        …..
        while (1)
        {
           /* Main house keeping of the SSC */
          MainLoop();
          ….
          ….
          ….
          #if (ESC_PHY_ENHANCEMENT)
                HW_PhyEnhancement();
          #endif
        }
}
```


Copyright (C) 2025 Reserved by ASIX Electronics Corporation

Copyright © 2025 ASIX Electronics Corporation. All rights reserved.

DISCLAIMER

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose, without the express written permission of ASIX. ASIX may make changes to the product specifications and descriptions in this document at any time, without notice.

ASIX provides this document “as is” without warranty of any kind, either expressed or implied, including without limitation warranties of merchantability, fitness for a particular purpose, and non-infringement.

Designers must not rely on the absence or characteristics of any features or registers marked “reserved”, “undefined” or “NC”. ASIX reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. Always contact ASIX to get the latest document before starting a design of ASIX products.

TRADEMARKS

ASIX, the ASIX logo are registered trademarks of ASIX Electronics Corporation. All other trademarks are the property of their respective owners.

EtherCAT® is a registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany.

Copyright (C) 2025 Reserved by ASIX Electronics Corporation

ASIX Electronics Corporation

4F, No.8, Hsin Ann RD., Hsinchu Science Park,
Hsinchu, Taiwan, R.O.C.

TEL: +886-3-5799500
FAX: +886-3-5799558

Email: support@asix.com.tw
Web: https://www.asix.com.tw

Copyright (C) 2025 Reserved by ASIX Electronics Corporation