

## **AX58100 Software API Design Guide**

Revision 1.00  
Nov. 19<sup>th</sup>, 2019

## Revision History

Revision	Date	Description
0.10	2019/07/12	Preliminary release
1.00	2019/11/19	1. Add Section 5 “AX58100 Extended Function Driver API”.

## Table of Contents

<b>1. Introduction .....</b>	<b>8</b>
<b>2. AX58100 Overview.....</b>	<b>8</b>
2-1. Block Diagram .....	9
2-2. Typical Applications .....	9
2-3. ESC/Function Registers Memory Map.....	10
<b>3. EtherCAT SSC Overview.....</b>	<b>12</b>
3-1. Code Structure.....	12
3-2. Execution Structure.....	14
3-3. Interrupt Handling.....	15
3-4. Process Data Handling .....	16
<b>4. SSC Hardware Access Implementation.....</b>	<b>18</b>
4-1. Interrupt Handler Functions .....	21
4-1-1. <i>ECAT_CheckTimer</i> .....	21
4-1-2. <i>PDI_Isr</i> .....	21
4-1-3. <i>Sync0_Isr</i> .....	21
4-1-4. <i>Sync1_Isr</i> .....	21
4-2. Interface Functions/Marcos .....	22
4-2-1. <i>HW_Init</i> .....	22
4-2-2. <i>HW_Release</i> .....	22
4-2-3. <i>HW_GetALEventRegister</i> .....	22
4-2-4. <i>HW_GetALEventRegister_Isr</i> .....	22
4-2-5. <i>HW_ResetALEventMask</i> .....	22
4-2-6. <i>HW_SetALEventMask</i> .....	23
4-2-7. <i>HW_SetLed</i> .....	23
4-2-8. <i>HW_RestartTarget</i> .....	23
4-2-9. <i>HW_DisableSyncManChannel</i> .....	23
4-2-10. <i>HW_EnableSyncManChannel</i> .....	23
4-2-11. <i>HW_GetSyncMan</i> .....	24
4-2-12. <i>HW_GetTimer</i> .....	24
4-2-13. <i>HW_ClearTimer</i> .....	24
4-2-14. <i>HW_EepromReload</i> .....	24
4-3. Read Access Functions .....	25
4-3-1. <i>HW_EscRead</i> .....	25
4-3-2. <i>HW_EscReadIsr</i> .....	25
4-3-3. <i>HW_EscReadDWord</i> .....	25
4-3-4. <i>HW_EscReadDWordIsr</i> .....	26

---

4-3-5.	<i>HW_EscReadWord</i> .....	26
4-3-6.	<i>HW_EscReadWordIsr</i> .....	26
4-3-7.	<i>HW_EscReadByte</i> .....	26
4-3-8.	<i>HW_EscReadByteIsr</i> .....	27
4-3-9.	<i>HW_EscReadMbxMem</i> .....	27
4-4.	Write Access Functions .....	28
4-4-1.	<i>HW_EscWrite</i> .....	28
4-4-2.	<i>HW_EscWriteIsr</i> .....	28
4-4-3.	<i>HW_EscWriteDWord</i> .....	28
4-4-4.	<i>HW_EscWriteDWordIsr</i> .....	29
4-4-5.	<i>HW_EscWriteWord</i> .....	29
4-4-6.	<i>HW_EscWriteWordIsr</i> .....	29
4-4-7.	<i>HW_EscWriteByte</i> .....	29
4-4-8.	<i>HW_EscWriteByteIsr</i> .....	30
4-4-9.	<i>HW_EscWriteMbxMem</i> .....	30
<b>5.</b>	<b>AX58100 Extended Function Driver API.....</b>	<b>31</b>
5-1.	Bridge Module .....	32
5-1-1.	<i>Bridge Module Function Description</i> .....	33
5-1-1-1.	<i>BRG_MotorControlEscAccess</i> .....	33
5-1-1-2.	<i>BRG_EncoderEscAccess</i> .....	33
5-1-1-3.	<i>BRG_EmIowdEscAccess</i> .....	33
5-1-1-4.	<i>BRG_SpiMasterEscAccess</i> .....	33
5-1-1-5.	<i>BRG_AllModulesEscAccess</i> .....	33
5-2.	MISC Module .....	34
5-2-1.	<i>MISC Module Definition/Macro Description</i> .....	35
5-2-1-1.	<i>MISC_GET_HOST_INTF_STATUS</i> .....	35
5-2-1-2.	<i>MISC_GET_INTR_CONTROL</i> .....	35
5-2-1-3.	<i>MISC_GET_INTR_STATUS</i> .....	35
5-2-1-4.	<i>MISC_CLR_INTR_STATUS</i> .....	35
5-2-2.	<i>MISC Module Function Description</i> .....	35
5-2-2-1.	<i>MISC_InterruptsEnable</i> .....	35
5-2-2-2.	<i>MISC_InterruptsDisable</i> .....	35
5-2-2-3.	<i>MISC_GetIRQ</i> .....	36
5-2-2-4.	<i>MISC_EfuseRead</i> .....	36
5-2-2-5.	<i>MISC_MotorControlOverride</i> .....	36
5-2-2-6.	<i>MISC_SpiMasterOverride</i> .....	36
5-2-2-7.	<i>MISC_BridgeOverride</i> .....	36
5-3.	IOWD Module .....	37
5-3-1.	<i>IOWD Module Definition/Macro Description</i> .....	38
5-3-1-1.	<i>IOWDEM_OBJECT</i> .....	38
5-3-2.	<i>IOWD Module Function Description</i> .....	39
5-3-2-1.	<i>IOWDEM_Init</i> .....	39
5-3-2-2.	<i>IOWDEM_DeInit</i> .....	39

---

---

5-3-2-3.	<i>IOWD_Start</i> .....	39
5-3-2-4.	<i>IOWD_Stop</i> .....	39
5-3-2-5.	<i>IOWD_SetTimeout</i> .....	39
5-3-2-6.	<i>IOWD_GetTimePeakValue</i> .....	39
5-3-2-7.	<i>IOWD_ClearTimerPeakValue</i> .....	40
5-3-2-8.	<i>IOWD_UnlockIO</i> .....	40
5-3-2-9.	<i>EM_Start</i> .....	40
5-3-2-10.	<i>EM_Stop</i> .....	40
5-3-2-11.	<i>EM_UnlockIO</i> .....	40
5-4.	<b>SPIM Module</b> .....	41
5-4-1.	<i>SPIM Module Definition/Macro Description</i> .....	42
5-4-1-1.	<i>SPIM_STATE</i> .....	42
5-4-1-2.	<i>SPIM_STATUS</i> .....	42
5-4-1-3.	<i>SPIM_CLK_MODE</i> .....	43
5-4-1-4.	<i>SPIM_POLARITY</i> .....	43
5-4-1-5.	<i>SPIM_SLAVE_ENABLE</i> .....	43
5-4-1-6.	<i>SPIM_SLAVE_ID</i> .....	44
5-4-1-7.	<i>SPIM_CHANNELS</i> .....	44
5-4-1-8.	<i>SPIM_CFG_OBJECT</i> .....	45
5-4-1-9.	<i>SPIM_SLAVE_INFO</i> .....	46
5-4-1-10.	<i>SPIM_OBJECT</i> .....	46
5-4-2.	<i>SPIM Module Function Description</i> .....	47
5-4-2-1.	<i>SPIM_Init</i> .....	47
5-4-2-2.	<i>SPIM_DeInit</i> .....	47
5-4-2-3.	<i>SPIM_Config</i> .....	47
5-4-2-4.	<i>SPIM_SetRxStartByte</i> .....	47
5-4-2-5.	<i>SPIM_ClearInterrupt</i> .....	47
5-4-2-6.	<i>SPIM_OneShotStart</i> .....	48
5-4-2-7.	<i>SPIM_ContinueStart</i> .....	48
5-4-2-8.	<i>SPIM_ContinueStop</i> .....	48
5-4-2-9.	<i>SPIM_ReadyLatchPulseCtrl</i> .....	48
5-4-2-10.	<i>SPIM_ClearSlaveAllocation</i> .....	48
5-4-2-11.	<i>SPIM_SwitchSlave</i> .....	49
5-4-2-12.	<i>SPIM_WriteTxBuf</i> .....	49
5-4-2-13.	<i>SPIM_Start</i> .....	49
5-4-2-14.	<i>SPIM_CheckComplete</i> .....	49
5-4-2-15.	<i>SPIM_ReadRxBuf</i> .....	50
5-4-2-16.	<i>SPIM_TransmitReceive</i> .....	50
5-5.	<b>MC Module</b> .....	51
5-5-1.	<i>MC Module Definition/Macro Description</i> .....	53
5-5-2.	<i>PWM Module Definition/Macro Description</i> .....	53
5-5-2-1.	<i>PWM_Polarity</i> .....	53
5-5-2-2.	<i>PWM_AlignMode</i> .....	53
5-5-2-3.	<i>PWM_Channels</i> .....	53
5-5-2-4.	<i>PWM_OBJECT</i> .....	53
5-5-3.	<i>STEP Module Definition/Macro Description</i> .....	54
5-5-3-1.	<i>STEP_OBJECT</i> .....	54

---

5-5-4. <i>PWM Mode Function Description</i> .....	54
5-5-4-1. <i>PWM_Init</i> .....	54
5-5-4-2. <i>PWM_DeInit</i> .....	55
5-5-4-3. <i>PWM_SetHighPulseWidth</i> .....	55
5-5-4-4. <i>PWM_SetAPulsePosition</i> .....	55
5-5-4-5. <i>PWM_Driving</i> .....	55
5-5-4-6. <i>PWM_H_Output</i> .....	55
5-5-4-7. <i>PWM_L_Output</i> .....	56
5-5-4-8. <i>PWM_A_Output</i> .....	56
5-5-4-9. <i>PWM_B_Output</i> .....	56
5-5-4-10. <i>PWM_C_Output</i> .....	56
5-5-4-11. <i>PWM_Z_Output</i> .....	56
5-5-4-12. <i>PWM_AB_Output</i> .....	56
5-5-4-13. <i>PWM_FuncEnable</i> .....	57
5-5-4-14. <i>PWM_FuncDisable</i> .....	57
5-5-4-15. <i>PWM_StopAllOutput</i> .....	57
5-5-4-16. <i>PWM_H_Polarity</i> .....	57
5-5-4-17. <i>PWM_L_Polarity</i> .....	57
5-5-5. <i>STEP Mode Function Description</i> .....	58
5-5-5-1. <i>STEP_Init</i> .....	58
5-5-5-2. <i>STEP_DeInit</i> .....	58
5-5-5-3. <i>STEP_Start</i> .....	58
5-5-5-4. <i>STEP_CheckCompleted</i> .....	58
5-5-5-5. <i>STEP_ClearStepCounter</i> .....	58
5-5-5-6. <i>STEP_GetStepCounter</i> .....	59
5-5-5-7. <i>STEP_StartFreeRun</i> .....	59
5-5-5-8. <i>STEP_StopFreeRun</i> .....	59
5-6. <i>ENC Module</i> .....	60
5-6-1. <i>ENC Module Definition/Macro Description</i> .....	61
5-6-2. <i>ABZ Module Definition/Macro Description</i> .....	61
5-6-2-1. <i>ENC_ABZ_RATIO</i> .....	61
5-6-2-2. <i>ENC_CLR_EVENT_AT_IZ_EDGE</i> .....	61
5-6-2-3. <i>ENC_ABZ_OBJECT</i> .....	61
5-6-3. <i>CW/CCW Module Definition/Macro Description</i> .....	62
5-6-3-1. <i>ENC_CWCCW_OBJECT</i> .....	62
5-6-4. <i>CLK/DIR Module Definition/Macro Description</i> .....	62
5-6-4-1. <i>ENC_CLKDIR_OBJECT</i> .....	62
5-6-5. <i>ABZ Mode Function Description</i> .....	63
5-6-5-1. <i>ENC_ABZ_Init</i> .....	63
5-6-5-2. <i>ENC_ABZ_DeInit</i> .....	63
5-6-6. <i>CW/CCW Mode Function Description</i> .....	63
5-6-6-1. <i>ENC_CWCCW_Init</i> .....	63
5-6-6-2. <i>ENC_CWCCW_DeInit</i> .....	63
5-6-7. <i>CLKDIR Mode Function Description</i> .....	64
5-6-7-1. <i>ENC_CLKDIR_Init</i> .....	64
5-6-7-2. <i>ENC_CLKDIR_DeInit</i> .....	64
5-6-8. <i>HALL Mode Function Description</i> .....	64

5-6-8-1. <i>ENC_HALL_Init</i> .....	64
5-6-8-2. <i>ENC_HALL_DeInit</i> .....	64
5-6-8-3. <i>ENC_HALL_GetStatus</i> .....	64
5-6-8-4. <i>ENC_HALL_GetPhaseDuration</i> .....	65
5-6-8-5. <i>ENC_HALL_ClearOverrun</i> .....	65
5-6-9. <i>ENC General Function Description</i> .....	65
5-6-9-1. <i>ENC_GetLatchedValue</i> .....	65
5-6-9-2. <i>ENC_GetCounter</i> .....	65
5-6-9-3. <i>ENC_ClearCounter</i> .....	65

## List of Figures

Figure 2-1. AX58100 Features .....	8
Figure 2-2. AX58100 Block Diagram.....	9
Figure 2-3. AX58100 Typical Applications.....	9
Figure 2-4. ESC Memory and Function Registers Mirror Mapping Table .....	11
Figure 3-1. EtherCAT SSC Code Structure .....	12
Figure 3-2. EtherCAT SSC Files Association.....	13
Figure 3-3. Basic SSC Execution Structure .....	14
Figure 3-4. SSC MainLoop Execution.....	14
Figure 3-5. ESC Interrupts Basics.....	15
Figure 3-6. PDO Output Mapping Example .....	16
Figure 3-7. PDO Input Mapping Example.....	17
Figure 4-1. SSC Hardware Related Defines .....	18
Figure 4-2. SSC uC Hardware Configuration Examples .....	19
Figure 4-3. SSC Hardware Functions Schema.....	20
Figure 5-1. AX58100 Extended Function Driver APIs.....	31
Figure 5-2. Bridge Module Block Diagram .....	32
Figure 5-3. MISC Module Block Diagram .....	34
Figure 5-4. IOWD Module Block Diagram .....	37
Figure 5-5. SPIM Module Block Diagram.....	41
Figure 5-6. MC Module Block Diagram.....	52
Figure 5-7. ENC Module Block Diagram.....	60

## 1. Introduction

This document describes the EtherCAT Slave Stack Code (SSC) software API to assist developers to porting their EtherCAT slave stack applications with AX58100 EtherCAT Slave Controller (ESC).

## 2. AX58100 Overview

The AX58100 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 AX58100 on different EtherCAT industrial fieldbus applications without extra microcontroller.

The AX58100 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
AX58100	2 x Internal PHY 1 x MII	8	8	9	64-bit	32	Yes

Part No.	Local Bus	SPI Master	PWM	ABZ/ Hall Encoder	Emergency Stop Input	Package	Temperature Range (°C)
AX58100	8/16 bit Async	Yes	3 Channels	Yes	Yes	LQFP-80	-40 ~ +105

Figure 2-1. AX58100 Features

## 2-1. Block Diagram

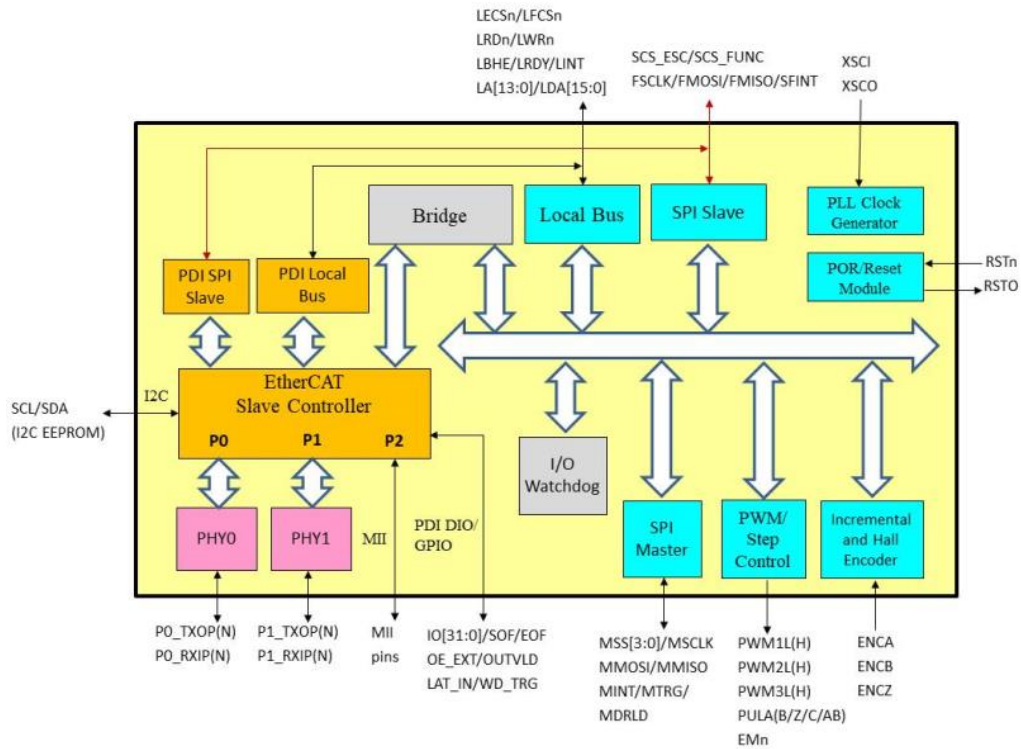


Figure 2-2. AX58100 Block Diagram

## 2-2. Typical Applications

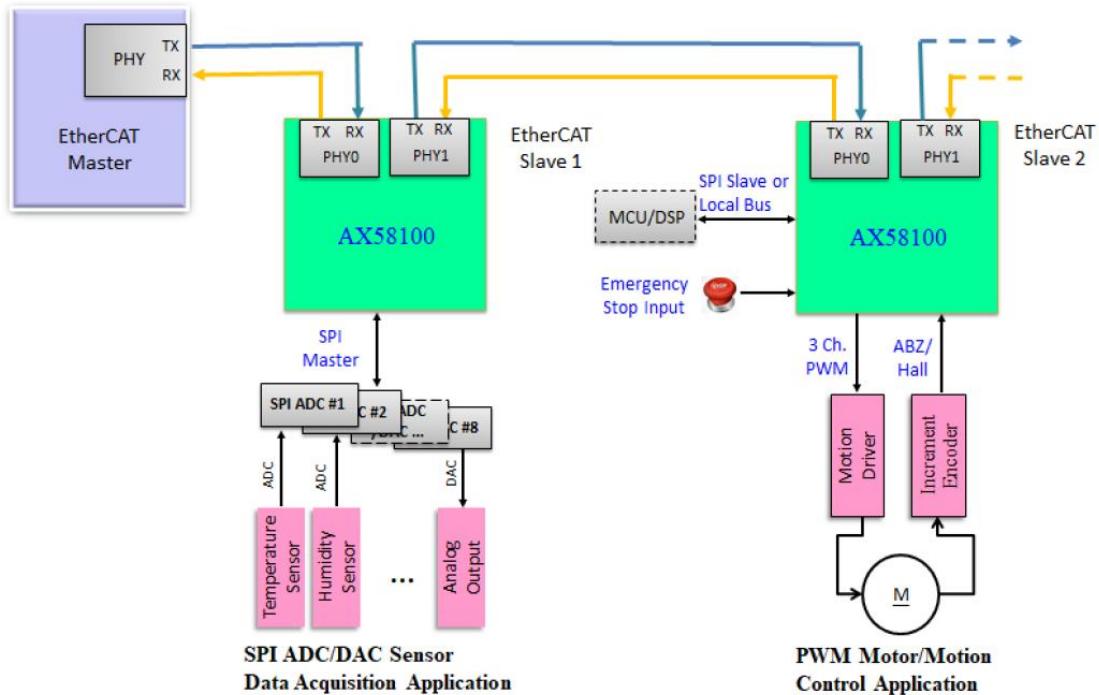


Figure 2-3. AX58100 Typical Applications

### 2-3. ESC/Function Registers Memory Map

Please refer to AX58100 datasheet for details.

Function Address	ESC Address		Name	Description
	R/W	RO		
0x000	0x3000	-	MCTLR	Motor Control Register
0x002	0x3002	-	PXCFGR	PWM Pulse X Configure Register
0x004	0x3004	-	PTAPPR	PWM Trigger A Pulse Position Register
0x006	0x3006	-	PTBPPR	PWM Trigger B Pulse Position Register
0x008	0x3008	-	PPCR	PWM Period Cycle Register
0x00A	0x300A	-	PBBMR	PWM Pulse Break Before Make Register
0x00C	0x300C	-	P1CTRLR	PWM1Control Register
0x00E	0x300E	-	P1SHR	PWM1 Counter Shift Register
0x010	0x3010	-	P1HPWR	PWM1 High Pulse Width Register
0x012	0x3012	-	P2CTRLR	PWM2 Control Register
0x014	0x3014	-	P2SHR	PWM2 Shift Register
0x016	0x3016	-	P2HPWR	PWM2 High Pulse Width Register
0x018	0x3018	-	P3CTRLR	PWM3 Control Register
0x01A	0x301A	-	P3SHR	PWM3 Counter Shift Register
0x01C	0x301C	-	P3HPWR	PWM3 High Pulse Width Register
0x020	0x3020	-	SGTLR	Step Gap Time Low Register
0x022			SGTHR	Step Gap Time High Register
0x024	0x3024	-	SHPWR	Step High Pulse Width Register
0x026	0x3026	-	TDLYR	direction Transform Delay step Register
0x028	0x3028	-	STNLR	Step Target Number Low Word Register
0x02A			STNHR	Step Target Number High Word Register
0x02C	0x302C	-	SCFGR	Step Configure Register
0x02E	0x302E	-	SCTRLR	Step Control Register
0x030	-	0x3230	SCNTLR	Step Counter Content Low Register
0x032			SCNTHR	Step Counter Content High Register
0x040	0x3040	-	ECNTVLR	Encoder Counter value Low Register
0x042			ECNTVHR	Encoder Counter value High Register
0x044	0x3044	-	ECNSTLR	Encoder Constant Low Register
0x046			ECNSTHR	Encoder Constant High Register
0x048	-	0x3248	ELATLR	Encoder Latched Low Register
0x04A			ELATHR	Encoder Latched High Register
0x04C	0x304C	-	EMODR	Encoder Mode Configuration Register
0x04E	0x304E	-	ECLRR	Encoder Clear Register
0x050	-	0x3250	HALSTR	Hall State Register
0x060	0x3060	-	WTLR	Watchdog Timer Low Register
0x062			WTHR	Watchdog Timer High Register
0x064	0x3064	-	WCFGR	Watchdog Configure Register
0x066	0x3066	-	WTPVCR	Watchdog Timer Peak Value Clear Register
0x068	0x3068	-	WMPLR	Watchdog Monitored Polarity Low Register
0x06A			WMPHR	Watchdog Monitored Polarity High Register

0x06C	0x306C	-	WMMLR	Watchdog Monitored Mask Low Register
0x06E			WMMHR	Watchdog Monitored Mask High Register
0x070	0x3070	-	WOMLR	Watchdog Output Mask Low Register
0x072			WOMHR	Watchdog Output Mask High Register
0x074	0x3074	-	WOELR	Watchdog Output Enable Low Register
0x076			WOEHR	Watchdog Output Enable High Register
0x078	0x3078	-	WOPLR	Watchdog Output Polarity Low Register
0x07A			WOPHR	Watchdog Output Polarity High Register
0x07C	-	0x327C	WTPVLR	Watchdog Timer Peak Value Low Register
0x07E			WTPVHR	Watchdog Timer Peak Value High Register
0x080	0x3080	-	SPICFGR	SPI Configure Register
0x082	0x3082	-	SPIBRR	SPI Baud Rate Register
0x084	0x3084	-	SPIDBSR	SPI Delay Byte and SS Register
0x086	0x3086	-	SPIDTR	SPI Delay Transfer Register
0x088	0x3088	-	SPIRPTR	SPI RDY / Pulse Time Register
0x08A	0x308A	-	SPILTR	SPI LDAC Time Register
0x08C	0x308C	-	SPIPLR	SPI Pulse/ RDY/ LDAC Register
0x090	0x3090	-	SPI01BCR	SPI 0/1 Byte Count Register
0x092	0x3092	-	SPI23BCR	SPI 2/3 Byte Count Register
0x094	0x3094	-	SPI45BCR	SPI 4/5 Byte Count Register
0x096	0x3096	-	SPI67BCR	SPI 6/7 Byte Count Register
0x098	0x3098	-	SPI03SSR	SPI 0/1/2/3 slave Select Register
0x09A	0x309A	-	SPI47SSR	SPI 4/5/6/7 slave Select Register
0x0A8	-	0x32A8	SPINTSR	SPI Interrupt Status Register
0x0AA	-	0x32AA	SPITSR	SPI Timeout Status Register
0x0AC	-	0x32AC	SPIOSR	SPI Pulse Overrun Status Register
0x0AE	-	0x32AE	SPIDSR	SPI Data Status Register
0x0B0	0x30B0	0x32B0	SPIC0DR	SPI Channel 0 Data Register
0x0B8	0x30B8	0x32B8	SPIC1DR	SPI Channel 1 Data Register
0x0C0	0x30C0	0x32C0	SPIC2DR	SPI Channel 2 Data Register
0x0C8	0x30C8	0x32C8	SPIC3DR	SPI Channel 3 Data Register
0x0D0	0x30D0	0x32D0	SPIC4DR	SPI Channel 4 Data Register
0x0D8	0x30D8	0x32D8	SPIC5DR	SPI Channel 5 Data Register
0x0E0	0x30E0	0x32E0	SPIC6DR	SPI Channel 6 Data Register
0x0E8	0x30E8	0x32E8	SPIC7DR	SPI Channel 7 Data Register
0x0F0	-	0x32F0	SPIDSMR	SPI Data Status Mirror Register
0x0F2	0x30F2	-	SPIMCR	SPI Master Control Register
0x100	0x3100	-	INTCR	Interrupt Configure Register
0x102	0x3102	-	INTSR	Interrupt Status Register

Figure 2-4. ESC Memory and Function Registers Mirror Mapping Table

## 3. EtherCAT SSC Overview

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

### 3-1. Code Structure

The EtherCAT Slave 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.

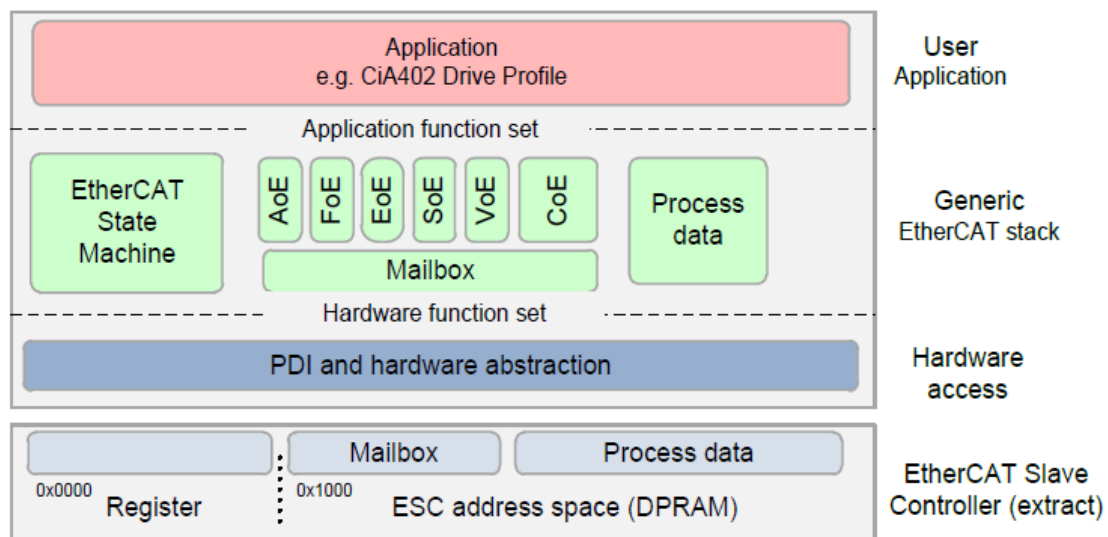


Figure 3-1. EtherCAT SSC Code Structure

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

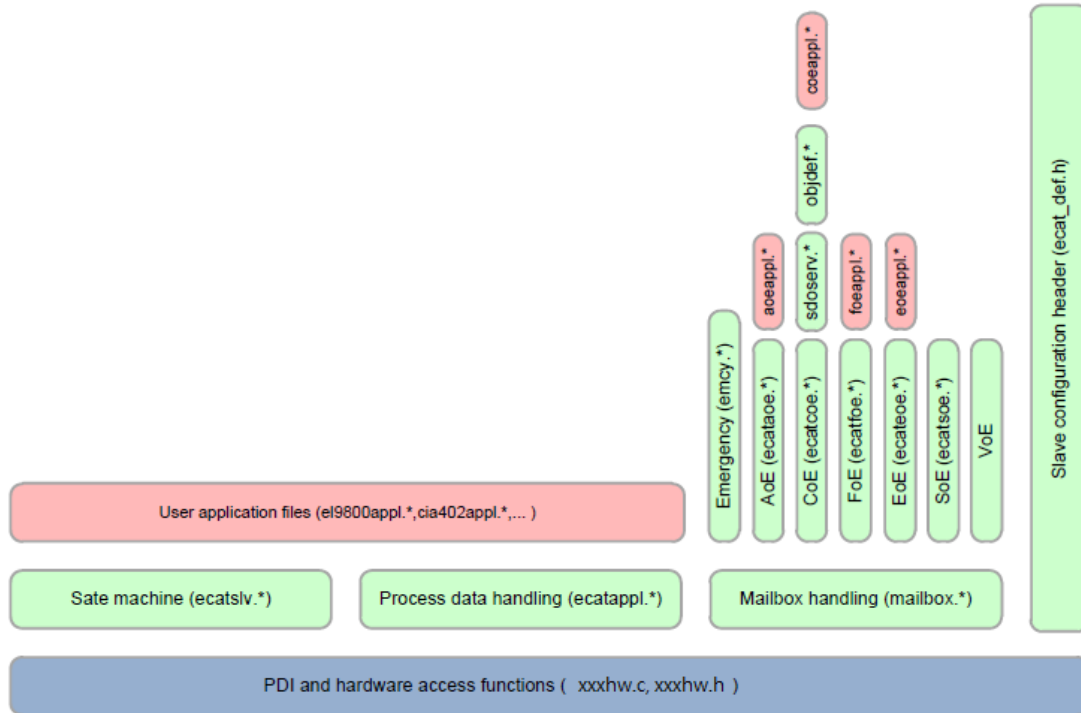


Figure 3-2. EtherCAT SSC Files Association

## 3-2. Execution Structure

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

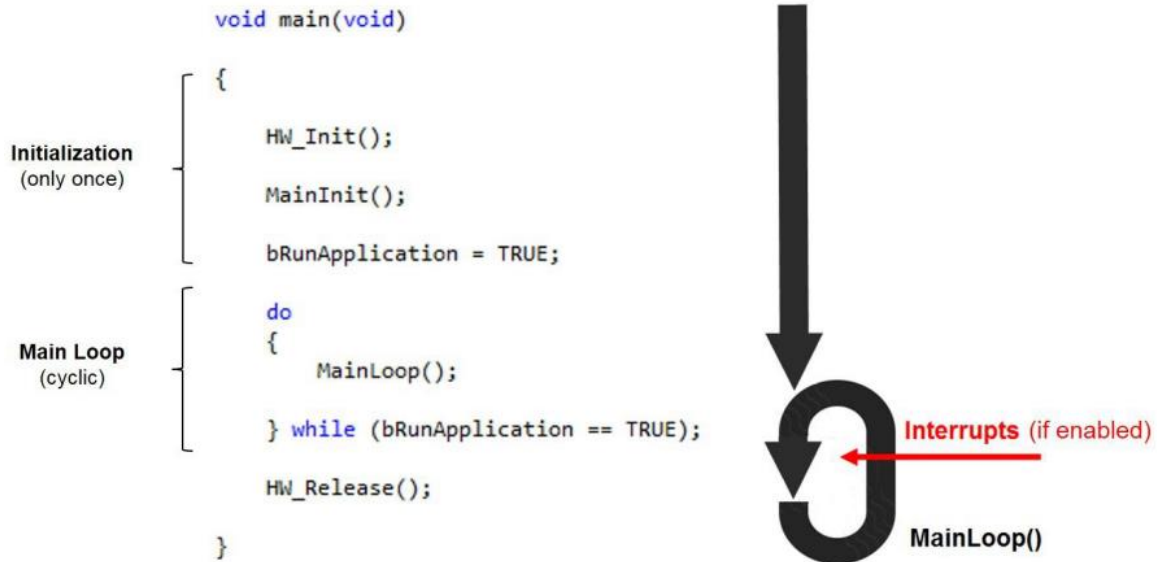


Figure 3-3. Basic SSC Execution Structure

The function MainLoop() contains the main cycle of Slave’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]

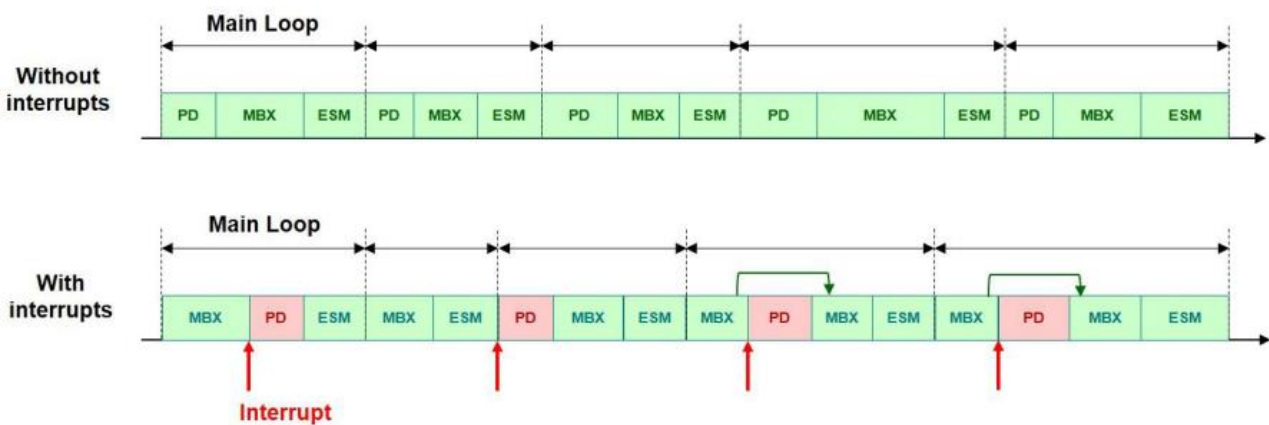


Figure 3-4. SSC MainLoop Execution

## 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 slave application.

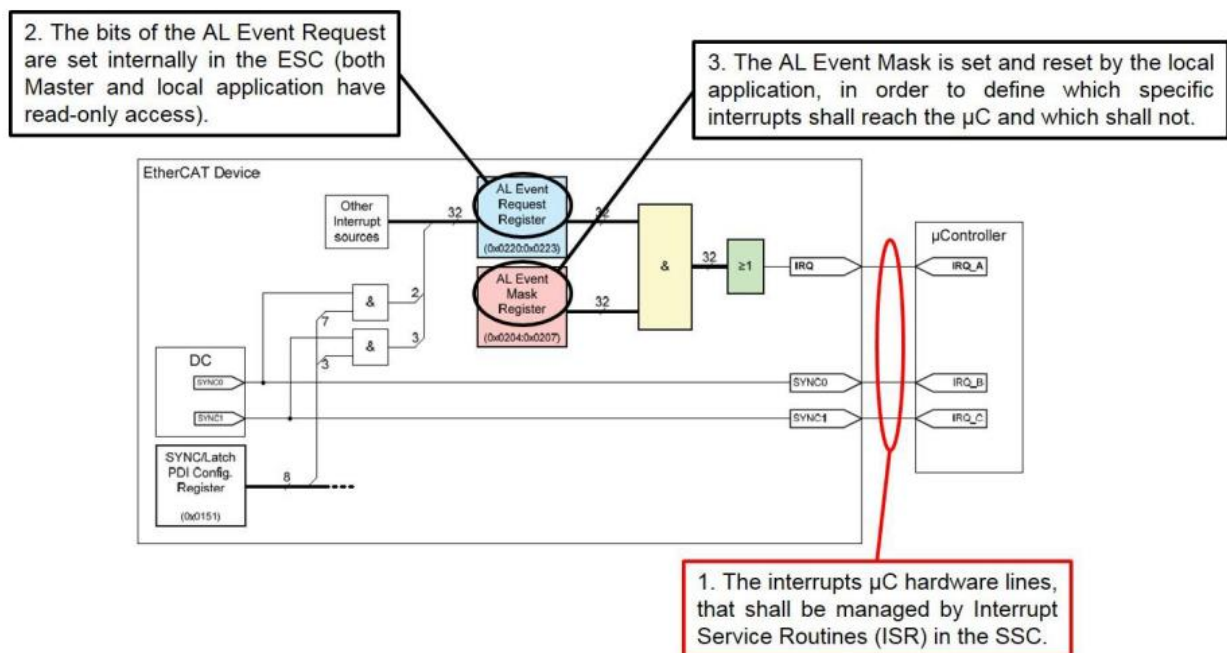


Figure 3-5. ESC Interrupts Basics

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 Pprocess Data SyncManager interrupts, and specifically: SM2 if Process Data Outputs are configured for the slave, 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).

## 3-4. Process Data Handling

The EtherCAT slave 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 slave 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 master to the slave.
2. **ECAT\_Application():** Contains the slave application, set general purpose outputs and read general purpose inputs.
3. **PDO\_InputMapping():** Handles the data from the slave to the master.

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 master to local LED of slave.

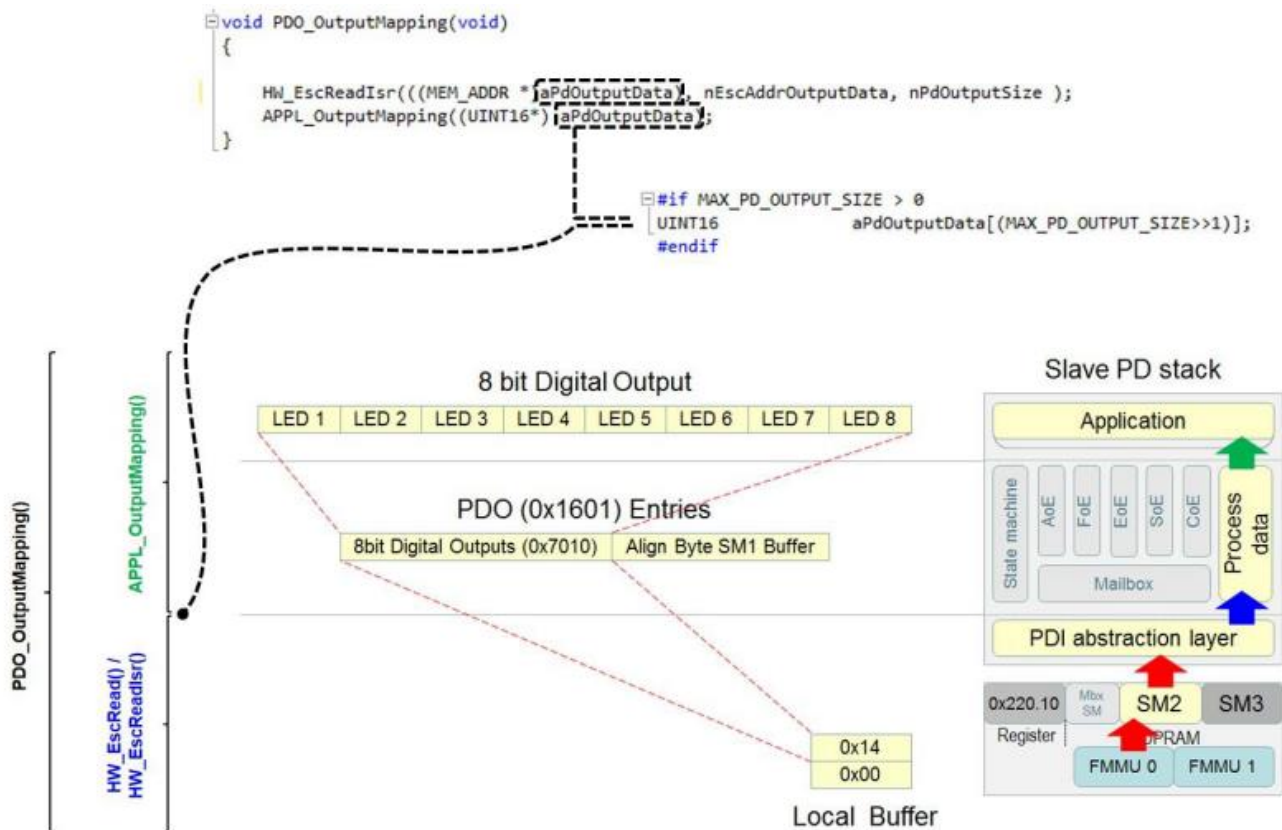


Figure 3-6. PDO Output Mapping Example

Below figure illustrates an example application mapping with PDO\_InputMapping for mapping analog data from slave to master.

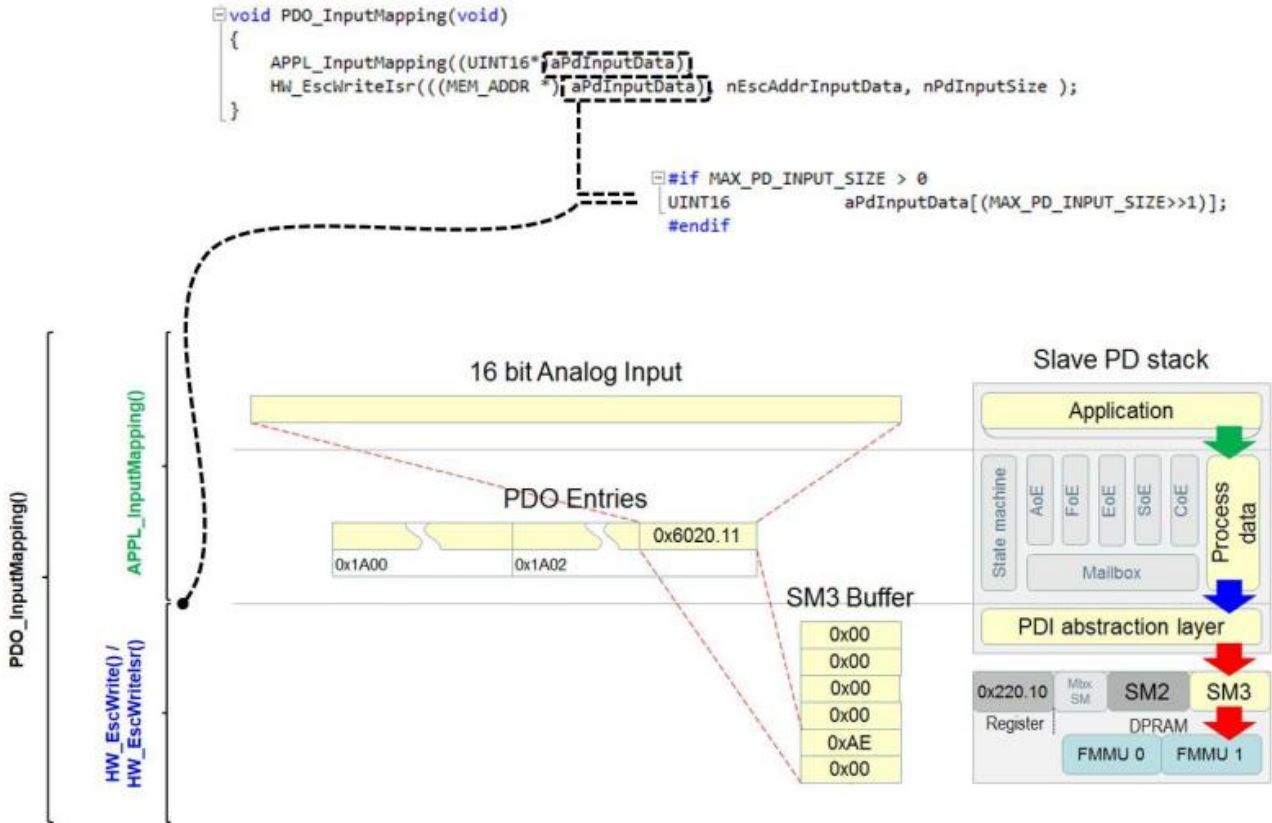


Figure 3-7. PDO Input Mapping Example

## 4. SSC Hardware Access Implementation

The Slave 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 slave 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 $\mu$ C 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 slave card from Beckhoff. Used on Win32 operating system.
CONTROLLER_16BIT	This define shall be used if the slave code is built for a 16Bit $\mu$ C.
CONTROLLER_32BIT	This define shall be used if the slave code is built for a 32Bit $\mu$ C
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 slave code will only access mailbox data 16Bit aligned. If the mailbox data is copied to the local $\mu$ C 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 $\mu$ C 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 $\mu$ C works in big endian format.

Figure 4-1. SSC Hardware Related Defines

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 slave 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	MCL_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

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

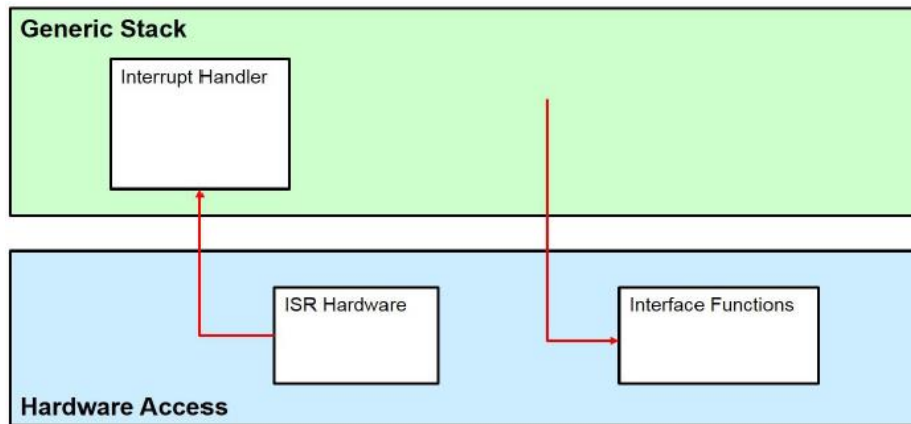


Figure 4-3. SSC Hardware Functions Schema

- **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.

## 4-1. Interrupt Handler Functions

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

### 4-1-1. ECAT\_CheckTimer

Prototype	<b>void ECAT_CheckTimer (void)</b>
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	<b>void PDI_Isr (void)</b>
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	<b>void Sync0_Isr (void)</b>
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	<b>void Sync1_Isr (void)</b>
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.

## 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	<b>UINT16 HW_Init (void)</b>
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	<b>void HW_Release (void)</b>
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	<b>UINT16 HW_GetALEventRegister (void)</b>
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	<b>UINT16 HW_GetALEventRegister_Isr (void)</b>
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	<b>void HW_ResetALEventMask (UINT16 intMask)</b>
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. <b>NOTE:</b> This function is only required for SSC 5.10 or older.

## 4-2-6. HW\_SetALEventMask

Prototype	<b>void HW_SetALEventMask (UINT16 intMask)</b>
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. <b>NOTE:</b> This function is only required for SSC 5.10 or older.

## 4-2-7. HW\_SetLed

Prototype	<b>void HW_SetLed (UINT8 RunLed, UINT8 ErrLed)</b>
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	<b>void HW_RestartTarget (void)</b>
Parameter	void
Return	void
Description	Resets the hardware. This function is only required if “BOOTSTRAPMODE_SUPPORTED” is set.

## 4-2-9. HW\_DisableSyncManChannel

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

## 4-2-10. HW\_EnableSyncManChannel

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

### 4-2-11. HW\_GetSyncMan

Prototype	<b>TSYNCMAN * HW_GetSyncMan (UINT8 channel)</b>
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. <b>NOTE:</b> This function is only required for SSC 5.10 or older.

### 4-2-12. HW\_GetTimer

Prototype	<b>UINT32 HW_GetTimer (void)</b>
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	<b>void HW_ClearTimer (void)</b>
Parameter	void
Return	void
Description	Clears the hardware timer value.

### 4-2-14. HW\_EepromReload

Prototype	<b>UINT16 HW_EepromReload (void)</b>
Parameter	void
Return	0 <> Error during EEPROM reload 0 = EEPROM load correct
Description	This function is called if an EEPROM reload request is triggered by the master. 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.

## 4-3. Read Access Functions

### 4-3-1. HW\_EscRead

Prototype	<b>void HW_EscRead (MEM_ADDR *pData, UINT16 Address, UINT16 Len)</b>
Parameter	<p>“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).</p> <p>“Address” EtherCAT Slave 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).</p> <p>“Len” Access size in Bytes</p>
Return	void
Description	Reads from the EtherCAT Slave Controller. This function is used to access ESC registers and the DPRAM area.

### 4-3-2. HW\_EscReadIsr

Prototype	<b>void HW_EscReadIsr (MEM_ADDR *pData, UINT16 Address, UINT16 Len)</b>
Parameter	<p>“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).</p> <p>“Address” EtherCAT Slave 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).</p> <p>“Len” Access size in Bytes</p>
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 Slave Controller. This function is used to access ESC registers and the DPRAM area.

### 4-3-3. HW\_EscReadDWord

Prototype	<b>void HW_EscReadDWord (UINT32 DWordValue, UINT16 Address)</b>
Parameter	<p>“DWordValue” Local 32Bit variable where the register value shall be stored.</p> <p>“Address” EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid 32Bit addresses are used.</p>
Return	void
Description	Reads two words from the specified address of the EtherCAT Slave 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)”

## 4-3-4. HW\_EscReadDWordIsr

Prototype	<b>void HW_EscReadDWordIsr (UINT32 DWordValue, UINT16 Address)</b>
Parameter	<p>“DWordValue” Local 32Bit variable where the register value shall be stored.</p> <p>“Address” EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid 32Bit addresses are used.</p>
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 Slave Controller.

## 4-3-5. HW\_EscReadWord

Prototype	<b>void HW_EscReadWord (UINT16 WordValue, UINT16 Address)</b>
Parameter	<p>“WordValue” Local 16Bit variable where the register value shall be stored.</p> <p>“Address” EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid 16Bit addresses are used.</p>
Return	void
Description	Reads one word from the specified address of the EtherCAT Slave 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	<b>void HW_EscReadWordIsr (UINT16 WordValue, UINT16 Address)</b>
Parameter	<p>“WordValue” Local 16Bit variable where the register value shall be stored.</p> <p>“Address” EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid 16Bit addresses are used.</p>
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 Slave Controller. Only required if “ESC_32_BIT_ACCESS” is not set.

## 4-3-7. HW\_EscReadByte

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

### 4-3-8. HW\_EscReadByteIsr

Prototype	<b>void HW_EscReadByteIsr (UINT8 ByteValue, UINT16 Address)</b>
Parameter	<p>“ByteValue” Local 8Bit variable where the register value shall be stored.</p> <p>“Address” EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes.</p>
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 Slave Controller. Only required if “ESC_16BIT_ACCESS” and “ESC_32BIT_ACCESS” are not set.

### 4-3-9. HW\_EscReadMbxMem

Prototype	<b>void HW_EscReadMbxMem (MEM_ADDR *pData, UINT16 Address, UINT16 Len)</b>
Parameter	<p>“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).</p> <p>“Address” EtherCAT Slave 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).</p> <p>“Len” Access size in Bytes</p>
Return	void
Description	Reads data from the ESC and copies to slave mailbox memory. If the local mailbox memory is also located in the application memory this function is equal to “HW_EscRead”.

## 4-4. Write Access Functions

### 4-4-1. HW\_EscWrite

Prototype	<b>void HW_EscWrite (MEM_ADDR *pData, UINT16 Address, UINT16 Len)</b>
Parameter	<p>“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).</p> <p>“Address” EtherCAT Slave 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).</p> <p>“Len” Access size in Bytes</p>
Return	void
Description	Writes from the EtherCAT Slave Controller. This function is used to access ESC registers and the DPRAM area.

### 4-4-2. HW\_EscWriteIsr

Prototype	<b>void HW_EscWriteIsr (MEM_ADDR *pData, UINT16 Address, UINT16 Len)</b>
Parameter	<p>“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).</p> <p>“Address” EtherCAT Slave 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).</p> <p>“Len” Access size in Bytes</p>
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 Slave Controller. This function is used to access ESC registers and the DPRAM area.

### 4-4-3. HW\_EscWriteDWord

Prototype	<b>void HW_EscWriteDWord (UINT32 DWordValue, UINT16 Address)</b>
Parameter	<p>“DWordValue” Local 32Bit variable which contains the data to be written to the ESC memory area.</p> <p>“Address” EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid 32Bit addresses are used.</p>
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 Slave Controller. This function is used to access ESC registers and the DPRAM area.

## 4-4-4. HW\_EscWriteDWordIsr

Prototype	<b>void HW_EscWriteDWordIsr (UINT32 DWordValue, UINT16 Address)</b>
Parameter	<p>“DWordValue” Local 32Bit variable which contains the data to be written to the ESC memory area.</p> <p>“Address” EtherCAT Slave Controller address . Specifies the offset within the ESC memory area in Bytes. Only valid 32Bit addresses are used.</p>
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 Slave Controller.

## 4-4-5. HW\_EscWriteWord

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

## 4-4-6. HW\_EscWriteWordIsr

Prototype	<b>void HW_EscWriteWordIsr (UINT16 WordValue, UINT16 Address)</b>
Parameter	<p>“WordValue” Local 16Bit variable which contains the data to be written to the ESC memory area.</p> <p>“Address” EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid 16Bit addresses are used.</p>
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 Slave Controller. Only required if “ESC_32BIT_ACCESS” is not set.

## 4-4-7. HW\_EscWriteByte

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

### 4-4-8. HW\_EscWriteByteIsr

Prototype	<b>void HW_EscWriteByteIsr (UINT8 ByteValue, UINT16 Address)</b>
Parameter	<p>“ByteValue” Local 8Bit variable which contains the data to be written to the ESC memory area</p> <p>“Address” EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes.</p>
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 Slave Controller. Only defined if “ESC 16BIT ACCESS” and “ESC 32BIT ACCESS” are disabled.

### 4-4-9. HW\_EscWriteMbxMem

Prototype	<b>void HW_EscWriteMbxMem (MEM_ADDR *pData, UINT16 Address, UINT16 Len)</b>
Parameter	<p>“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).</p> <p>“Address” EtherCAT Slave 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).</p> <p>“Len” Access size in Bytes.</p>
Return	void
Description	Writes data from the slave mailbox memory to ESC memory. If the local mailbox memory is also located in the application memory this function is equal to “HW_EscWrite”.

## 5. AX58100 Extended Function Driver API

The AX58100 EtherCAT slave 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 AX58100 extended functions, this section is mainly to introduce the software drivers of AX58100 extended functions.

The AX58100 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 Slave Stack Code (SSC) is generated by SSC tool, and the EtherCAT Application is user defined.

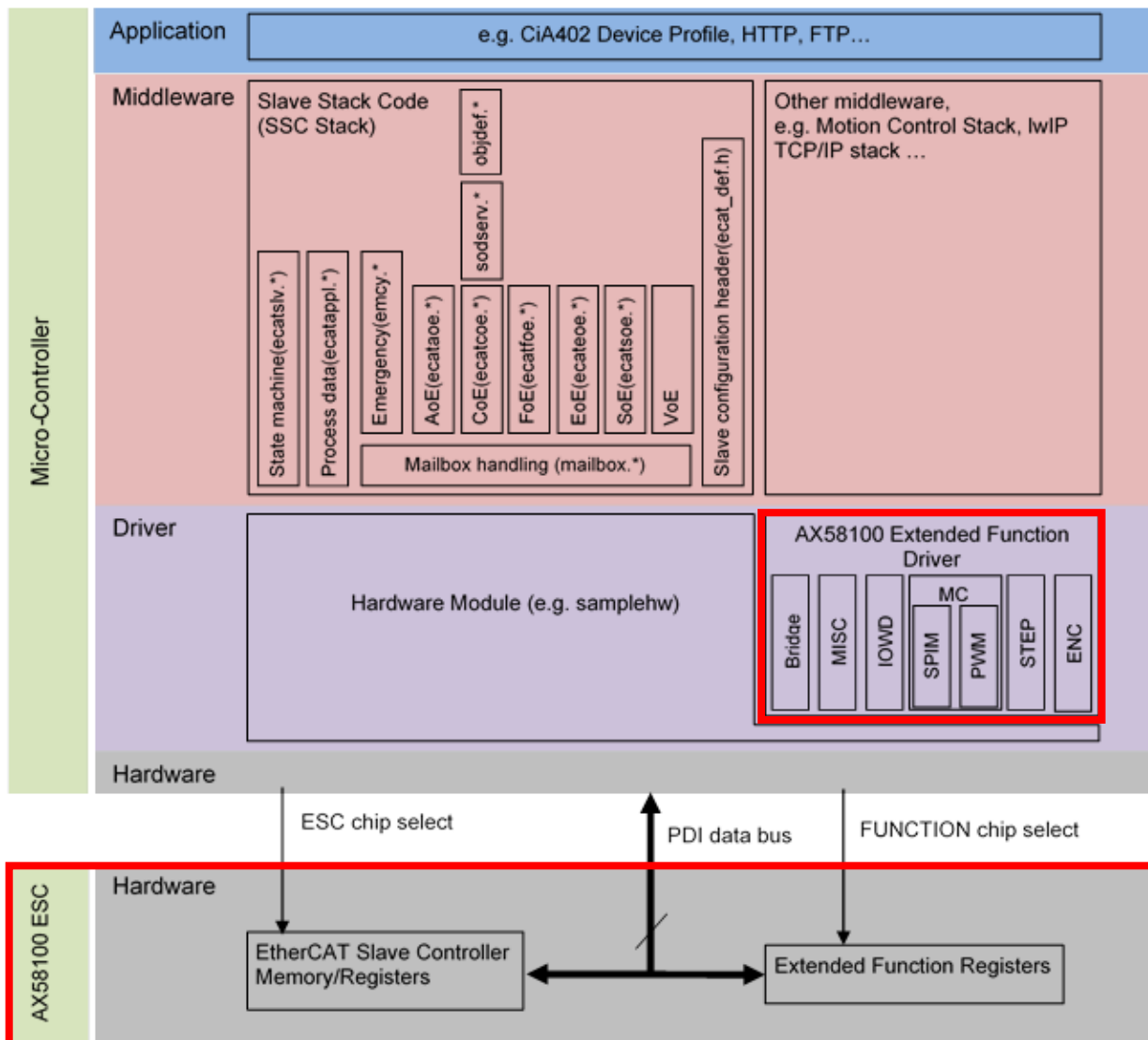


Figure 5-1. AX58100 Extended Function Driver APIs

## 5-1. Bridge Module

Typically, the access control value of AX58100 Extended Function registers are loaded from EEPROM by using Category 1 section, but can also be modified at run time by MCU firmware. Bridge module provides APIs to specify access control value and enable/disable register mirroring.

Related files of Bridge module are explained as below.

### **ax58100\_bridge.c** –

The major file of Bridge module, it is used to switch access right of extended function registers.

### **ax58100\_bridge.h** –

The header file of Bridge module.

The Bridge Module Functions are illustrated as below figure.

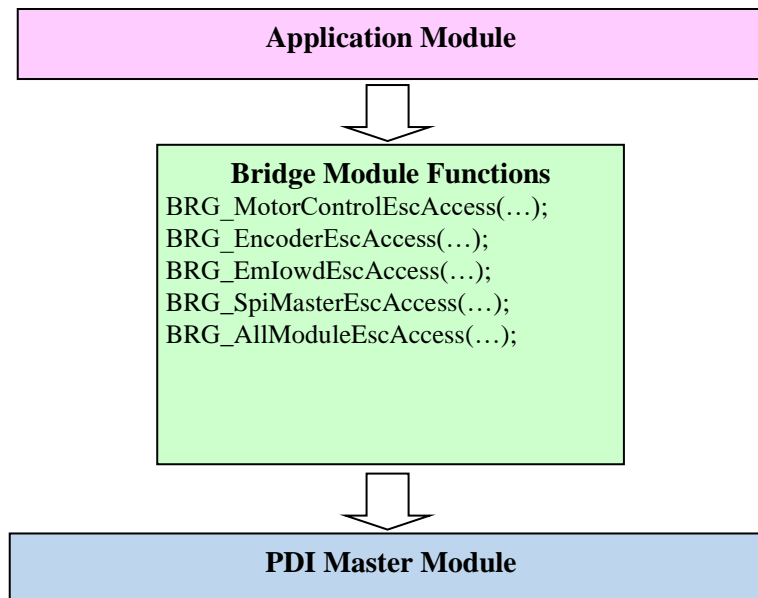


Figure 5-2. Bridge Module Block Diagram

## 5-1-1. Bridge Module Function Description

The Bridge module provides following API functions.

### 5-1-1-1. BRG\_MotorControlEscAccess

Prototype	<b>void BRG_MotorControlEscAccess(u8 Enable)</b>
Parameter	“Enable” 0: Switch access right to function side. 1: Switch access right to ESC side.
Return	void
Description	This function is used to switch access right of Motor Control (MC) registers.

### 5-1-1-2. BRG\_EncoderEscAccess

Prototype	<b>void BRG_EncoderEscAccess(u8 Enable)</b>
Parameter	“Enable” 0: Switch access right to function side. 1: Switch access right to ESC side.
Return	void
Description	This function is used to switch access right of Encoder (ENC) registers.

### 5-1-1-3. BRG\_EmIowdEscAccess

Prototype	<b>void BRG_EmIowdEscAccess(u8 Enable)</b>
Parameter	“Enable” 0: Switch access right to function side. 1: Switch access right to ESC side.
Return	void
Description	This function is used to switch access right of I/O Watch-Dog (IOWD) registers.

### 5-1-1-4. BRG\_SpiMasterEscAccess

Prototype	<b>void BRG_SpiMasterEscAccess(u8 Enable)</b>
Parameter	“Enable” 0: Switch access right to function side. 1: Switch access right to ESC side.
Return	void
Description	This function is used to switch access right of SPI Master (SPIM) registers.

### 5-1-1-5. BRG\_AllModulesEscAccess

Prototype	<b>void BRG_AllModulesEscAccess(u8 Enable)</b>
Parameter	“Enable” 0: Switch access right to function side. 1: Switch access right to ESC side.
Return	void
Description	This function is used to switch access right of entire Extended Function registers.

## 5-2. 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.

ax58100\_misc.c –

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

ax58100\_misc.h –

The header file of MISC module.

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

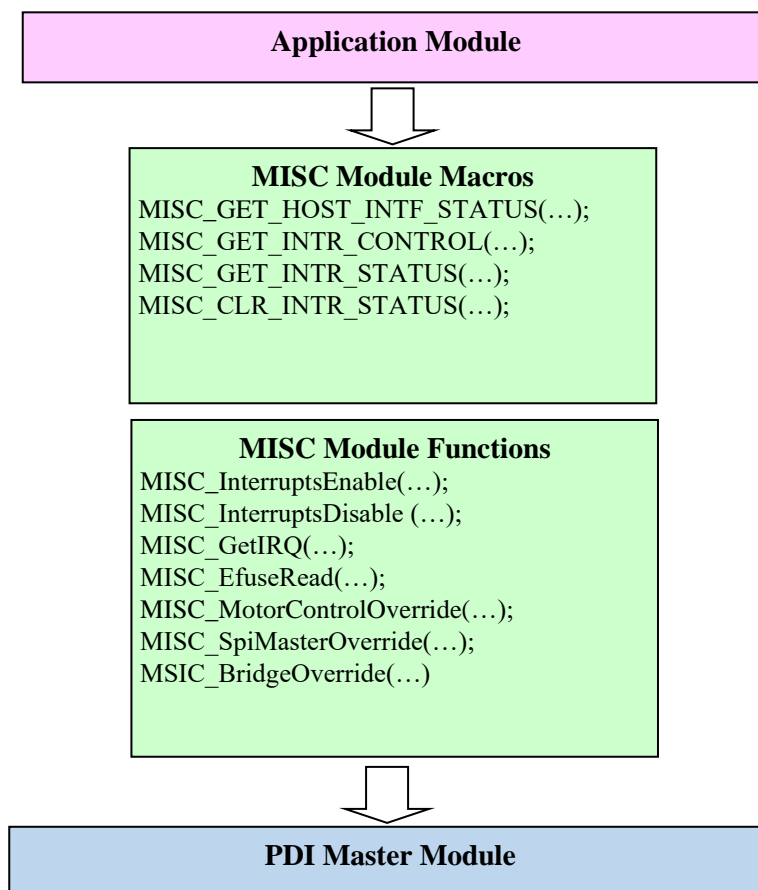


Figure 5-3. MISC Module Block Diagram

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

### 5-2-1-1. MISC\_GET\_HOST\_INTF\_STATUS

Prototype	<b>MISC_GET_HOST_INTF_STATUS(pcStu)</b>
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	<b>MISC_GET_INTR_CONTROL(pwCtrl)</b>
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	<b>MISC_GET_INTR_STATUS(pwFlags)</b>
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	<b>MISC_CLR_INTR_STATUS(pwFlags)</b>
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	<b>void MISC_InterruptsEnable(u16 EnableMask)</b>
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	<b>void MISC_InterruptsDisable(u16 DisableMask)</b>
Parameter	“DisableMask” b0: Don’t care. b1: Select the interrupts to be disabled.
Return	void

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

### 5-2-2-3. MISC\_GetIRQ

Prototype	<b>u16 MISC_GetIRQ(void)</b>
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	<b>AX_STATUS MISC_EfuseRead(u8 Addr, u8 *pBuf, u8 Size)</b>
Parameter	<p>“Addr” The start address of Efuse that you want to read.</p> <p>*pBuf The empty buffer used to store Efuse data.</p> <p>Size Byte size that you want to read.</p>
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	<b>void MISC_MotorControlOverride(u8 Enable)</b>
Parameter	<p>“Enable” 0: Disable MC override, MC hardware only operate at EtherCAT OP mode.</p> <p>1: Enable MC override, MC hardware operate no matter what EtherCAT state is.</p>
Return	void
Description	This function is used to Enable/Disable MC hardware for debugging.

### 5-2-2-6. MISC\_SpiMasterOverride

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

### 5-2-2-7. MISC\_BridgeOverride

Prototype	<b>void MISC_BridgeOverride (u8 Enable)</b>
Parameter	<p>“Enable” 0: Disable Bridge override, Bridge hardware only operate at EtherCAT OP mode.</p> <p>1: Enable Bridge override, Bridge hardware operate no matter what EtherCAT state is.</p>
Return	void
Description	This function is used to Enable/Disable Bridge hardware for debugging.

### 5-3. IOWD Module

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

Related files of IOWD module are explained as below.

**ax58100\_iowd.c** –

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

**ax58100\_iowd.h** –

The header file of IOWD sub-module.

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

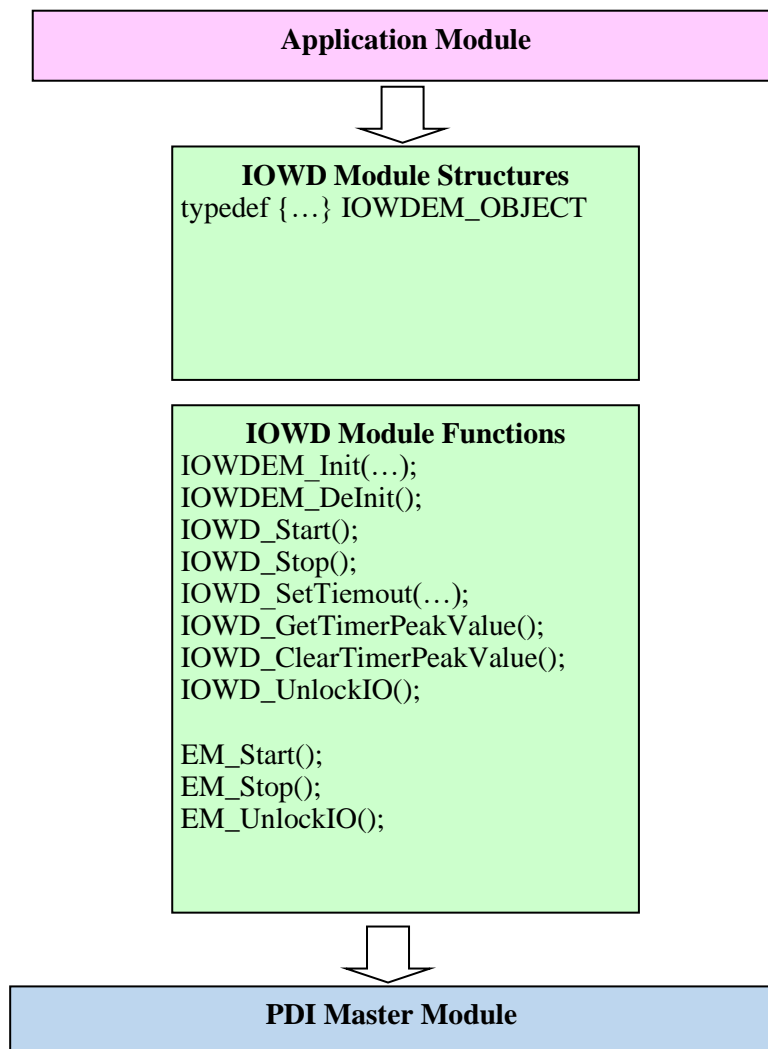


Figure 5-4. IOWD Module Block Diagram

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

### 5-3-1-1. IOWDEM\_OBJECT

Prototype	<b>typedef struct {...} IOWDEM_OBJECT</b>
Member	<p>u32 InputMonitorMask –  b0: Disable monitoring on the input.  b1: Enable monitoring on the input.  This property is bit-wise operation.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>u8 EnableSofInputMonitor –  0: Disable monitoring on SOF event.  1: Enable monitoring on SOF event.</p> <p>u8 EnableEscCsInputMonitor –  0: Disable monitoring on ESC chip select event.  1: Enable monitoring on ESC chip select event.</p> <p>u8 EnableFunCsInputMonitor –  0: Disable monitoring of Function chip select event.  1: Enable monitoring of Function chip select event.</p> <p>u8 DetectInputRisingEdge –  0: Detect falling edge on monitored inputs.  1: Detect rising edge on monitored inputs.</p>
Description	This structure is used setup IOWD properties.

## 5-3-2. IOWD Module Function Description

The IOWD module provides following API functions.

### 5-3-2-1. IOWDEM\_Init

Prototype	<b>AX_STATUS IOWDEM_Init(IOWDEM_OBJECT *pIowdEmObj)</b>
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	<b>void IOWDEM_DeInit(void)</b>
Parameter	void
Return	void
Description	This function is called to de-initiate IOWD/EM peripheral hardware module.

### 5-3-2-3. IOWD\_Start

Prototype	<b>void IOWD_Start(void)</b>
Parameter	void
Return	void
Description	This function is used to start IOWD function.

### 5-3-2-4. IOWD\_Stop

Prototype	<b>void IOWD_Stop(void)</b>
Parameter	void
Return	void
Description	This function is called to stop IOWD function.

### 5-3-2-5. IOWD\_SetTimeout

Prototype	<b>void IOWD_SetTimeout(u32 Timeout)</b>
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	<b>u32 IOWD_GetTimePeakValue(void)</b>
Parameter	void
Return	Time peak value in 10nsec unit.
Description	This function is called to get time peak value.

**5-3-2-7. IOWD\_ClearTimerPeakValue**

Prototype	<b>void IOWD_ClearTimerPeakValue(void)</b>
Parameter	void
Return	void
Description	This function is called to clear up time peak value.

**5-3-2-8. IOWD\_UnlockIO**

Prototype	<b>void IOWD_UnlockIO(void)</b>
Parameter	void
Return	void
Description	This function is called to unlock IOs by IOWD function.

**5-3-2-9. EM\_Start**

Prototype	<b>void EM_Start(void)</b>
Parameter	void
Return	void
Description	This function is used to enable Emergency function.

**5-3-2-10. EM\_Stop**

Prototype	<b>void EM_Stop(void)</b>
Parameter	void
Return	void
Description	This function is called to stop Emergency function.

**5-3-2-11. EM\_UnlockIO**

Prototype	<b>void EM_UnlockIO(void)</b>
Parameter	void
Return	void
Description	This function is called to unlock IOs by Emergency function.

## 5-4. 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.

### ax58100\_spim.c –

The major file of SPIM module.

### ax58100\_spim.h –

The header file of SPIM module.

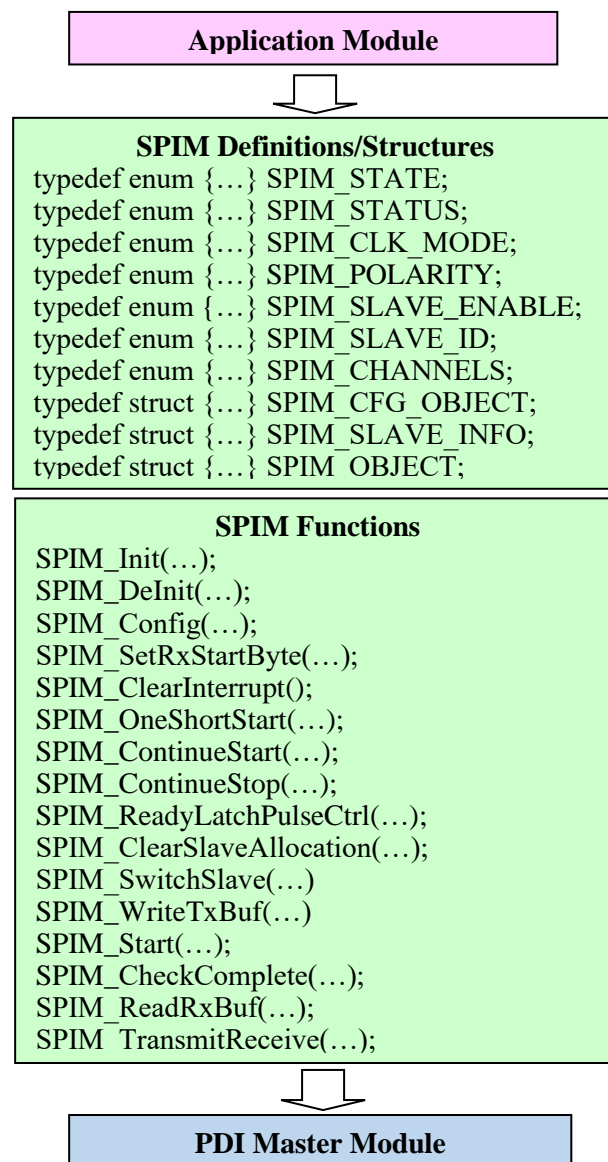


Figure 5-5. SPIM Module Block Diagram

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

AX58100 SPIM Module supports either One Shot or Continue access mode. Some definitions/structures should be defined/declared. These definitions/structures are declared in the ax58100\_spim.h file.

Following subsections describe the enumerations, structures and functions for SPI master use.

### 5-4-1-1. SPIM\_STATE

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

### 5-4-1-2. SPIM\_STATUS

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

### 5-4-1-3. SPIM\_CLK\_MODE

Prototype	<b>typedef enum {...} SPIM_CLK_MODE</b>
Member	<p>SCLK_MODE_0 – Set clock mode 0.</p> <p>SCLK_MODE_1 – Set clock mode 1.</p> <p>SCLK_MODE_2 – Set clock mode 2.</p> <p>SCLK_MODE_3 – Set clock mode 3.</p>
Description	This define the SPIM clock mode that can be used.

### 5-4-1-4. SPIM\_POLARITY

Prototype	<b>typedef enum {...} SPIM_POLARITY</b>
Member	<p>SPIM_PolarityActiveLow – Set polarity to active low.</p> <p>SPIM_PolarityActiveHigh – Set polarity to active high.</p>
Description	This define the polarity of SPIM output signals.

### 5-4-1-5. SPIM\_SLAVE\_ENABLE

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

**5-4-1-6. SPIM\_SLAVE\_ID**

Prototype	<b>typedef enum {...} SPIM_SLAVE_ID</b>
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 define the identification of slave select.

**5-4-1-7. SPIM\_CHANNELS**

Prototype	<b>typedef enum {...} SPIM_CHANNELS</b>
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 define the identification of channel.

**5-4-1-8. SPIM\_CFG\_OBJECT**

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

	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	<b>typedef struct {...} SPIM_SLAVE_INFO</b>
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	<b>typedef struct {...} SPIM_OBJECT</b>
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.

## 5-4-2. SPIM Module Function Description

The SPIM module provides the following SPI master API functions.

### 5-4-2-1. SPIM\_Init

Prototype	<b>AX_STATUS SPIM_Init(SPIM_OBJECT *pSpimObj)</b>
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	<b>void SPIM_DeInit(SPIM_OBJECT *pSpimObj)</b>
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	<b>AX_STATUS SPIM_Config(SPIM_OBJECT *pSpimObj)</b>
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	<b>void SPIM_SetRxStartByte(SPIM_OBJECT *pSpimObj, u8 StartByte)</b>
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	<b>void SPIM_ClearInterrupt(void)</b>
Parameter	void
Return	void
Description	This function is used to clear SPIM complete flag and interrupt flag.

## 5-4-2-6. SPIM\_OneShotStart

Prototype	<b>AX_STATUS SPIM_OneShotStart(SPIM_OBJECT *pSpimObj)</b>
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	<b>AX_STATUS SPIM_ContinueStart(SPIM_OBJECT *pSpimObj)</b>
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	<b>AX_STATUS SPIM_ContinueStop(SPIM_OBJECT *pSpimObj)</b>
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	<b>void SPIM_ReadyLatchPulseCtrl(SPIM_OBJECT *pSpimObj, u8 Enable)</b>
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	<b>AX_STATUS SPIM_ClearSlaveAllocation(SPIM_OBJECT *pSpimObj)</b>
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.

## 5-4-2-11. SPIM\_SwitchSlave

Prototype	<b>AX_STATUS SPIM_SwitchSlave(SPIM_OBJECT *pSpimObj, SPIM_SLAVE_ID SlaveId)</b>
Parameter	<p>“*pSpimObj” This is a pointer to SPIM_OBJECT instance object.</p> <p>“SPIM_SLAVE_ID SlaveId” Select the slave id which will be allocated transfer.</p>
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	<b>s32 SPIM_WriteTxBuf(SPIM_OBJECT *pSpimObj, u8 *pData, u8 Len)</b>
Parameter	<p>“*pSpimObj” This is a pointer to SPIM_OBJECT instance object.</p> <p>“u8 *pData” A pointer to the data will be transmitted.</p> <p>“u8 Len” Specify the byte length of transmit data.</p>
Return	<p>&gt;=0 The actual byte length of data be filled into TX buffer.</p> <p>&lt;0 Fail.</p>
Description	This function is used to fill data into TX buffer.

## 5-4-2-13. SPIM\_Start

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

### 5-4-2-15. SPIM\_ReadRxBuf

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

### 5-4-2-16. SPIM\_TransmitReceive

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

## **5-5. MC Module**

The AX58100 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.

**ax58100\_pwm.c** –

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

**ax58100\_pwm.h** –

The header file of the PWM sub-module.

**ax58100\_step.c** –

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

**ax58100\_step.h** –

The header file of the STEP sub-module.

The MC Module are illustrated as below figure.

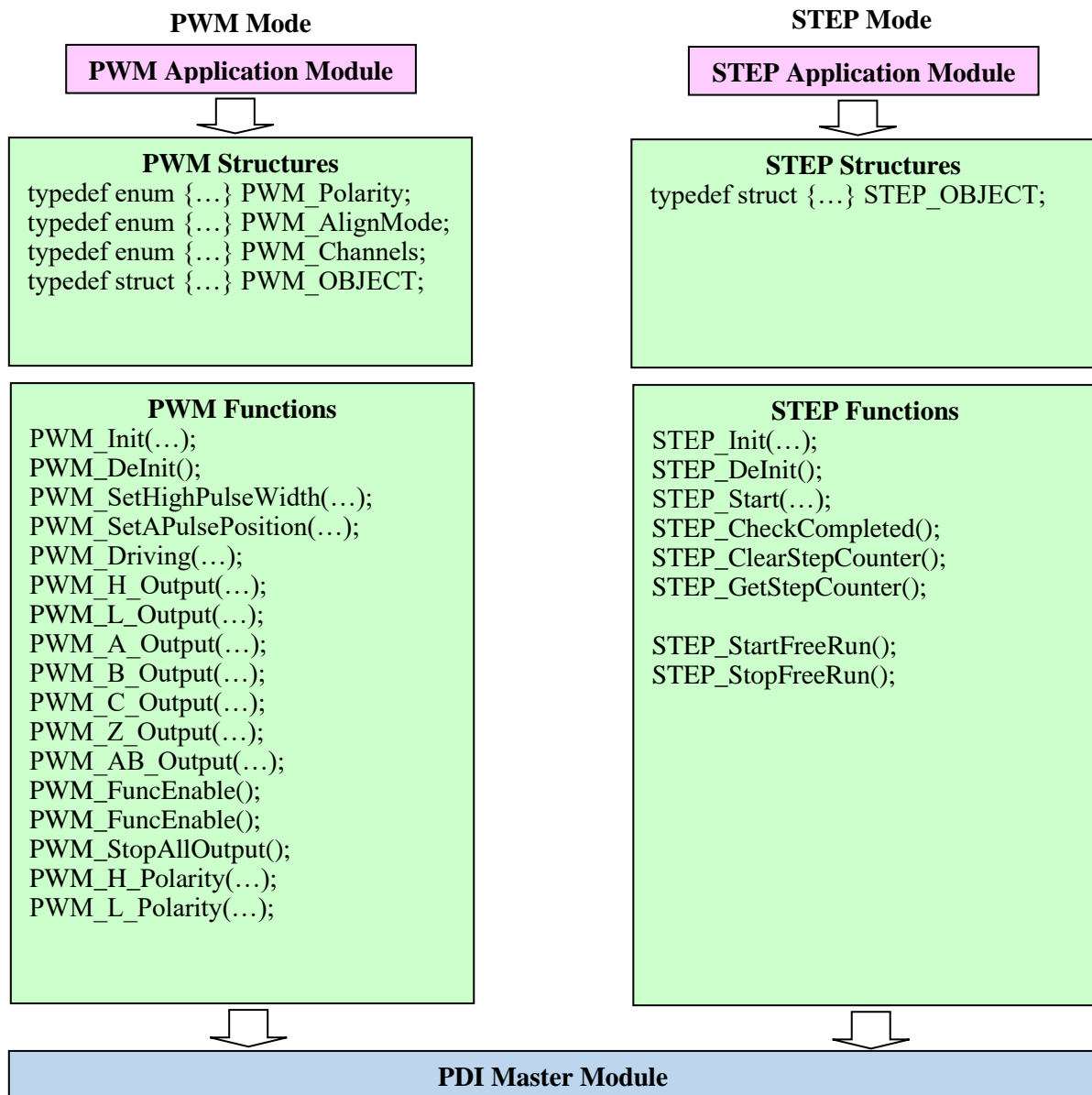


Figure 5-6. MC Module Block Diagram

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

AX58100 MC Module driver supports either PWM or STEP mode. Some definitions/structures should be defined/declared. These definitions/structures are declared in the ax58100\_pwm.h and ax58100\_step.h file.

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

### 5-5-2-1. PWM\_Polarity

Prototype	<b>typedef enum {...} PWM_Polarity</b>
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	<b>typedef enum {...} PWM_AlignMode</b>
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	<b>typedef enum {...} PWM_Channels</b>
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	<b>typedef struct {...} PWM_OBJECT</b>
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[] –

	<p>Specify the polarity of PWM high side signal.  <b>PWM_Polarity</b> PwmlPolarity[] –          Specify the polarity of PWM low side signal.  <b>u16 PwmShiftTime</b>[] –          Specify the shift time of PWM signal will be generated, unit in 10nsec.  <b>PWM_AlignMode</b> AlignMode –          Specify the alignment mode of PWM signals.  <b>u16 AbczHighPulseWidth</b> –          Specify the pulse width of A/B/C/Z signals.  <b>PWM_Polarity</b> AbczPulsePolarity –          Specify the polarity of A/B/C/Z signals.  <b>u16 A_PulseTrigPosition</b> –          Specify the trigger position of A pulse, unit in 10nsec.  <b>u16 B_PulseTrigPosition</b> –          Specify the trigger position of B pulse, unit in 10nsec.</p>
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	<b>typedef enum {...} STEP_OBJECT</b>
Member	<p><b>u32 GapTime</b> –          Specify the gap time of two STEP pulses, unit in 10nsec.  <b>u16 HighPulseWidth</b> –          Specify the high pulse width of STEP pulse will be generated, unit in 10nsec.  <b>u16 DirXferDelay</b> –          Specify the transfer delay time of direction signal, unit in 10nsec.  <b>STEP_Polarity</b> DirectionSignalPolarity –          Specify the polarity of direction signal.  <b>STEP_Polarity</b> StepPulsePolarity –          Specify the polarity of STEP pulse signal.</p>
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	<b>AX_STATUS PWM_Init(PWM_OBJECT *pPwmObj)</b>
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.

**5-5-4-2. PWM\_DeInit**

Prototype	<b>void PWM_DeInit(void)</b>
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	<b>void PWM_SetHighPulseWidth(PWM_OBJECT *pPwmObj, PWM_Channels Channel, u16 HighPulseWidth)</b>
Parameter	“*pPwmObj” This is a pointer to PWM_OBJECT instance object. “Channel” Specify a channel number that has defined in <b>PWM_Channels</b> 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	<b>void PWM_SetAPulsePosition(PWM_OBJECT *pPwmObj, u16 Position)</b>
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	<b>void PWM_Driving(PWM_Channels Channel, u8 EnableDriving)</b>
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	<b>void PWM_H_Output(PWM_Channels Channel, u8 EnableOutput)</b>
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.

## 5-5-4-7. PWM\_L\_Output

Prototype	<b>void PWM_L_Output(PWM_Channels Channel, u8 EnableOutput)</b>
Parameter	“Channel” Specify a channel number that has defined in <b>PWM_Channels</b> 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	<b>void PWM_A_Output(u8 EnableOutput)</b>
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	<b>void PWM_B_Output(u8 EnableOutput)</b>
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	<b>void PWM_C_Output(u8 EnableOutput)</b>
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	<b>void PWM_Z_Output(EnableOutput)</b>
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	<b>void PWM_AB_Output(EnableOutput)</b>
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.

**5-5-4-13. PWM\_FuncEnable**

Prototype	<b>void PWM_FuncEnable(void)</b>
Parameter	void
Return	void
Description	This function is used to enable PWM function.

**5-5-4-14. PWM\_FuncDisable**

Prototype	<b>void PWM_FuncDisable(void)</b>
Parameter	void
Return	void
Description	This function is used to disable PWM function.

**5-5-4-15. PWM\_StopAllOutput**

Prototype	<b>void PWM_StopAllOutput(void)</b>
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	<b>void PWM_H_Polarity(PWM_Channels Channel, PWM_Polarity HighPolarity)</b>
Parameter	“Channel” Specify a channel number that has defined in <b>PWM_Channels</b> enumeration. “HighPolarity” Specify high side polarity that has defined in <b>PWM_Polarity</b> enumeration.
Return	void
Description	This function is used to change high side polarity.

**5-5-4-17. PWM\_L\_Polarity**

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

## 5-5-5. STEP Mode Function Description

### 5-5-5-1. STEP\_Init

Prototype	<b>AX_STATUS STEP_Init(STEP_OBJECT *pStepObj)</b>
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	<b>AX_STATUS STEP_DeInit(void)</b>
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	<b>AX_STATUS STEP_Start(s32 TargetStepNumber)</b>
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	<b>AX_STATUS STEP_CheckCompleted(void)</b>
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	<b>void STEP_ClearStepCounter(void)</b>
Parameter	void
Return	void
Description	This function is used to clear STEP counter register.

**5-5-5-6. STEP\_GetStepCounter**

Prototype	<b>s32 STEP_GetStepCounter(void)</b>
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	<b>AX_STATUS STEP_StartFreeRun(u8 Direction)</b>
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	<b>AX_STATUS STEP_StopFreeRun(void)</b>
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.

## 5-6. ENC Module

The Encoder module (ENC) is part of AX58100 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.

**ax58100\_enc.c** –

The major file of ENC module.

**ax58100\_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.

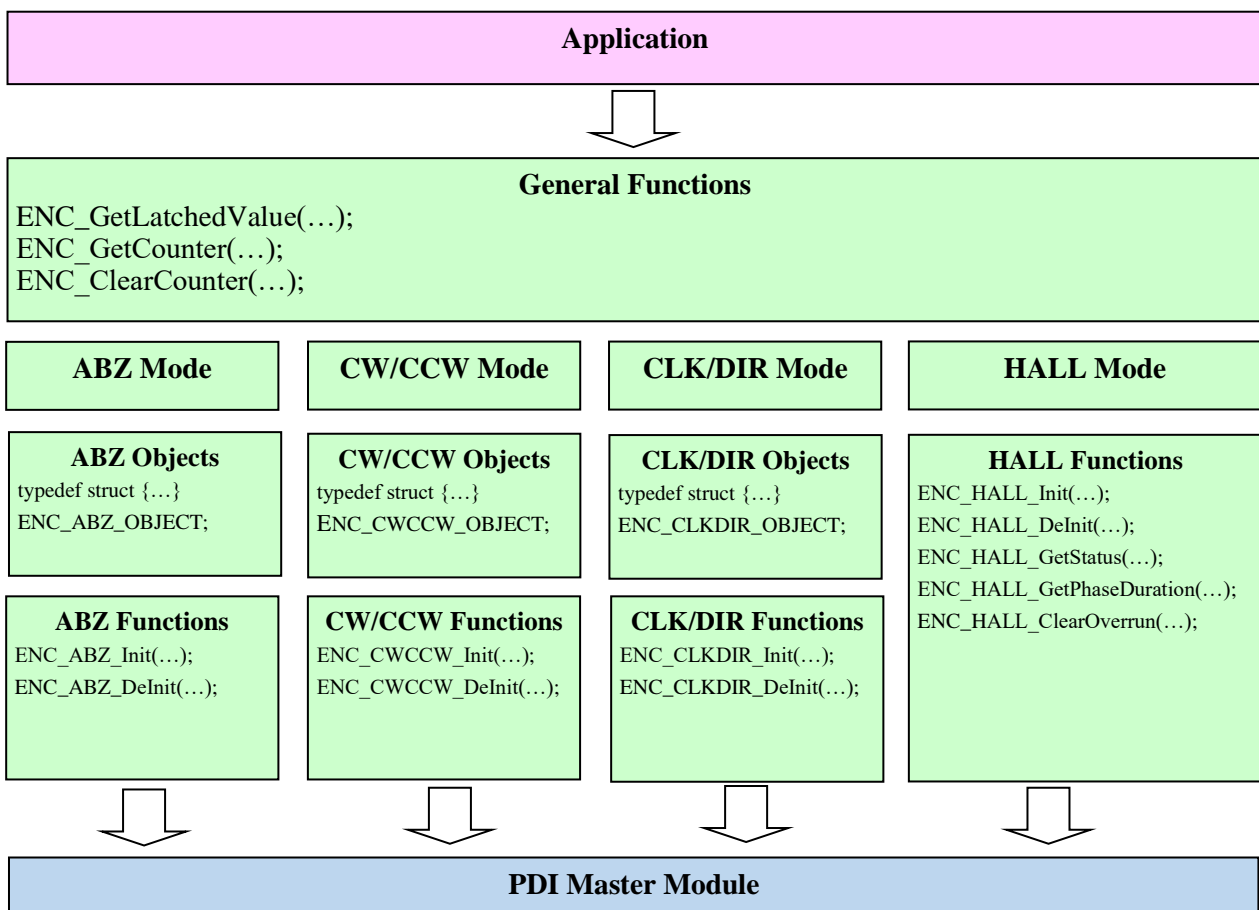


Figure 5-7. ENC Module Block Diagram

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

AX58100 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 ax58100\_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	<b>typedef enum {...} ENC_ABZ_RATIO</b>
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	<b>typedef enum {...} ENC_CLR_EVENT_AT_IZ_EDGE</b>
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	<b>typedef struct {...} ENC_ABZ_OBJECT</b>
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.

	<p>ENC_ABZ_RATIO Ratio – Specify the ratio of ABZ encoder, please refer to ENC_ABZ_RATIO enumeration definition.</p> <p>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.</p> <p>u8 EnableDecimalMode – 0: Disable decimal mode for internal counting. 1: Enable decimal mode for internal counting.</p> <p>u16 ConstantFractional – Specify fractional counting base.</p> <p>u16 ConstantInteger – Specify integer counting base.</p>
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	<b>typedef struct {...} ENC_CWCCW_OBJECT</b>
Member	<p>u8 EnableDecimalMode – 0: Disable decimal mode for internal counting. 1: Enable decimal mode for internal counting.</p> <p>u16 ConstantFractional – Specify fractional counting base.</p> <p>u16 ConstantInteger – Specify integer counting base.</p>
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	<b>typedef struct {...} ENC_CLKDIR_OBJECT</b>
Member	<p>u8 EnableDecimalMode – 0: Disable decimal mode for internal counting. 1: Enable decimal mode for internal counting.</p> <p>u16 ConstantFractional – Specify fractional counting base.</p> <p>u16 ConstantInteger – Specify integer counting base.</p>
Description	This object is used as a CLK/DIR encoder instance for further configuration or control.

## 5-6-5. ABZ Mode Function Description

### 5-6-5-1. ENC\_ABZ\_Init

Prototype	<b>AX_STATUS ENC_ABZ_Init(ENC_ABZ_OBJECT *pAbzObj)</b>
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	<b>void ENC_ABZ_DeInit(void)</b>
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	<b>AX_STATUS ENC_CWCCW_Init(ENC_CWCCW_OBJECT *pCwccwObj)</b>
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	<b>void ENC_CWCCW_DeInit(void)</b>
Parameter	void
Return	void
Description	This function is called to de-initiate ENC hardware.

## 5-6-7. CLKDIR Mode Function Description

### 5-6-7-1. ENC\_CLKDIR\_Init

Prototype	<b>AX_STATUS ENC_CLKDIR_Init(ENC_CLKDIR_OBJECT *pClkdirObj)</b>
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	<b>void ENC_CLKDIR_DeInit(void)</b>
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	<b>void ENC_HALL_Init(void)</b>
Parameter	void
Return	void
Description	This function is called to initiate ENC hardware in HALL mode.

### 5-6-8-2. ENC\_HALL\_DeInit

Prototype	<b>void ENC_HALL_DeInit(void)</b>
Parameter	void
Return	void
Description	This function is called to de-initiate ENC hardware.

### 5-6-8-3. ENC\_HALL\_GetStatus

Prototype	<b>void ENC_HALL_GetStatus(u8 *pCurrStatus, u8 *pKeepStatus, u8 *pOverrun)</b>
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.

**5-6-8-4. ENC\_HALL\_GetPhaseDuration**

Prototype	<b>u32 ENC_HALL_GetPhaseDuration(void)</b>
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	<b>void ENC_HALL_ClearOverrun(void)</b>
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	<b>u32 ENC_GetLatchedValue(void)</b>
Parameter	void
Return	32bits latched value.
Description	This function is used to get latched value.

**5-6-9-2. ENC\_GetCounter**

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

**5-6-9-3. ENC\_ClearCounter**

Prototype	<b>void ENC_ClearCounter(void)</b>
Parameter	void
Return	void
Description	This function is used to clear counter register.



**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](mailto:support@asix.com.tw)**

**Web: <http://www.asix.com.tw>**