MICROCHIP

AN2655

LAN9252 SDK Firmware API Guide

Author: *Riyas Kattukandan*
Microchip Technology Inc.

1.0 INTRODUCTION

This document describes the firmware APIs used to integrate the Microchip LAN9252 EtherCAT® slave controller (ESC) with PIC32MX, which assist developers porting their EtherCAT slave stack application with LAN9252.

1.1 Terms and Abbreviations

- ETG - EtherCAT® Technology Group
- ESC - EtherCAT® Slave Controller
- EVB - Engineering Validation Board
- HAL - Hardware Abstraction Layer
- HBI - Host Bus Interface
- IDE - Integrated Development Environment
- PDI - Process Data Interface
- SDK - Software Development Kit
- SPI - Serial Protocol Interface
- SQI - Serial Quad Interface
- SSC - Slave Stack Code

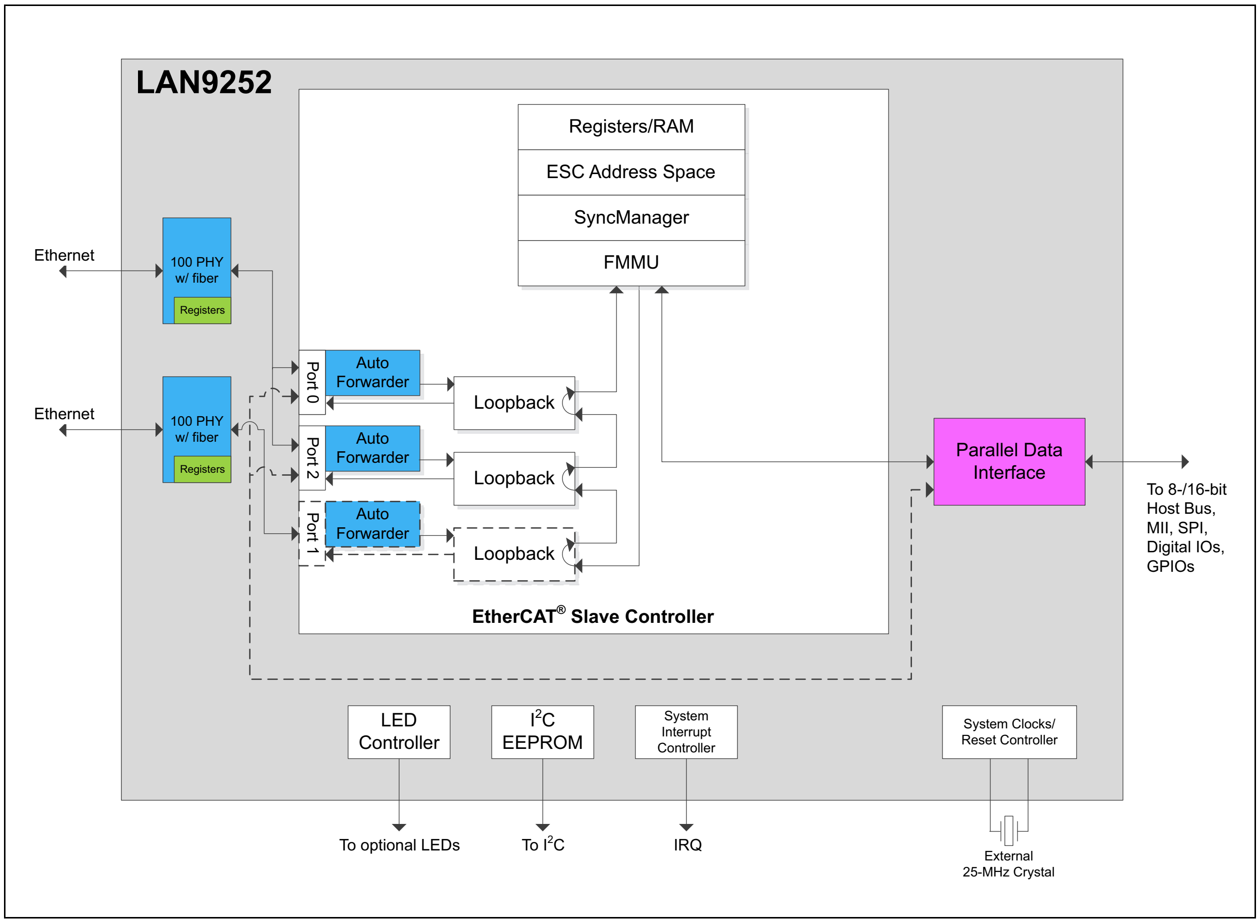
1.2 References

The following documents should be referenced when using this application note. See your Microchip representative for availability.

- *LAN9252 Data Sheet*
- *AN1907 - Microchip LAN9252 Migration from Beckhoff ET1100*
- *AN1916 - AN1916 Integrating Microchip's LAN9252 SDK with Beckhoff's EtherCAT ®SSC*
- *AN1995 - LAN9252 SOC Porting Guidelines*
- *EtherCAT Slave Stack Code (SSC) ET9300 - www.ethercat.org*

2.0 LAN9252 GENERAL DESCRIPTION

FIGURE 2-1: LAN9252 BLOCK DIAGRAM



The LAN9252 is a 2/3-port EtherCAT slave controller with dual integrated Ethernet PHYs that each contain a full-duplex 100BASE-TX transceiver and support 100-Mbps (100BASE-TX) operation.

LAN9252-based solutions can be implemented in the following modes:

Microcontroller Mode: The LAN9252 communicates with the microcontroller through an SRAM-like slave interface. The simple, yet highly functional host bus interface provides a glue-less connection to most common 8- or 16-bit micro-processors and microcontrollers, as well as 32-bit microprocessors with an 8- or 16-bit external bus.

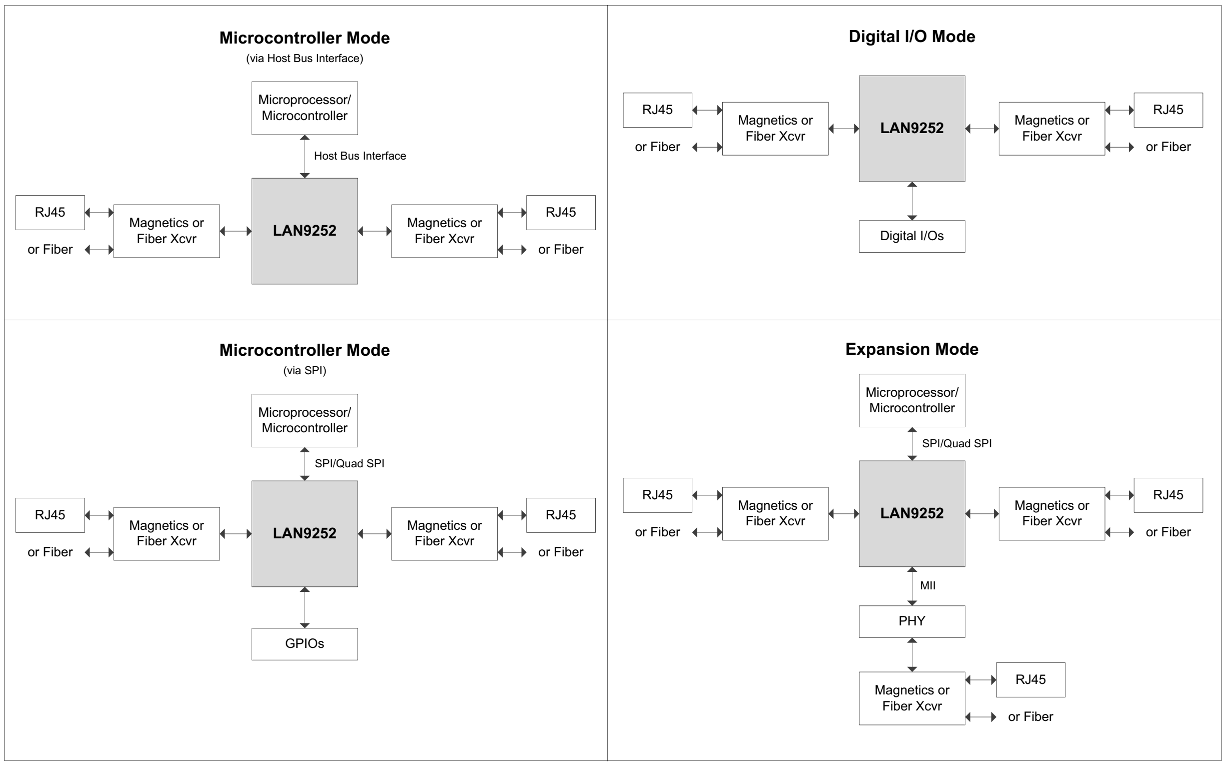
Alternatively, the device can be accessed via SPI or Quad SPI, while also providing up to 16 inputs or outputs for general purpose usage.

Expansion Mode: While the device is in SPI or Quad SPI mode, a third networking port can be enabled to provide an additional MII port. This port can be connected to an external PHY to enable star or tree network topologies, or con-nected to another LAN9252 to create a four-port solution. This port can be configured for the upstream or downstream direction.

Digital I/O Mode: For simple digital modules without microcontrollers, the LAN9252 can operate in Digital I/O Mode where 16 digital signals can be controlled or monitored by the EtherCAT master. Six control signals are also provided.

Figure 2-2 shows the block diagram of the operation modes.

FIGURE 2-2: LAN9252 MODES OF OPERATION



Slave stack code (SSC) is applicable in Microcontroller and Expansion modes where the Microprocessor/Microcontroller process data interface (PDI) is available.

SDK 1.3 only supports 32-bit architecture. However, for 8-bit and 16-bit architectures, it is necessary to change the source code according to the compiler's requirements.

System On Chip (SOC) MPUs/MCs can access LAN9252 using different types of PDIs like HBI, SPI, or SQI. Accessibility of each ESC register using PDI is defined in the LAN9252 data sheet.

2.1 LAN9252 Register Classification

There are two types of registers available in the LAN9252:

- Directly accessible registers (LAN9252 Control and Status Registers (CSR))

TABLE 2-1: SYSTEM CONTROLS AND STATUS REGISTERS

Address	Register Name (Symbol)
000h-01Ch	EtherCAT Process RAM Read Data FIFO (ECAT_PRAM_RD_DATA)
020h-03Ch	EtherCAT Process RAM Write Data FIFO (ECAT_PRAM_WR_DATA)
050h	Chip ID and Revision (ID_REV)
054h	Interrupt Configuration Register (IRQ_CFG)
058h	Interrupt Status Register (INT_STS)
05Ch	Interrupt Enable Register (INT_EN)
064h	Byte Order Test Register (BYTE_TEST)
074h	Hardware Configuration Register (HW_CFG)
084h	Power Management Control Register (PMT_CTRL)
08Ch	General Purpose Timer Configuration Register (GPT_CFG)
090h	General Purpose Timer Count Register (GPT_CNT)
09Ch	Free Running 25MHz Counter Register (FREE_RUN)
Reset Register	
1F8h	Reset Control Register (RESET_CTL)
EtherCAT Registers	
300h	EtherCAT CSR Interface Data Register (ECAT_CSR_DATA)
304h	EtherCAT CSR Interface Command Register (ECAT_CSR_CMD)
308h	EtherCAT Process RAM Read Address and Length Register (ECAT_PRAM_RD_ADDR_LEN)
30Ch	EtherCAT Process RAM Read Command Register (ECAT_PRAM_RD_CMD)
310h	EtherCAT Process RAM Write Address and Length Register (ECAT_PRAM_WR_ADDR_LEN)
314h	EtherCAT Process RAM Write Command Register (ECAT_PRAM_WR_CMD)

- Indirectly accessible registers – (EtherCAT Status and Control registers - ESC). All EtherCAT core registers reside under this group. The application can read and write EtherCAT core registers using LAN9252 CSR registers.

Note: Refer to the LAN9252 data sheet for a more detailed description of each register.

2.2 Interrupts

All interrupts must be configured during hardware initialization. Since SSC accesses ESC registers from both interrupt context and polling mode, ECAT_CSR_CMD and ECAT_CSR_DATA registers must be protected; otherwise, it may corrupt the SSC state machine routines.

For example, if an interrupt fires during reading of 0x120 (ESC) register under polling mode, soon after updating 0x120 (Address) into the ECAT_CSR_CMD register, then any ESC access (read/write) from the interrupt routine overwrites the ECAT_CSR_CMD register (some other address, for example, 0x220), which overrides the ECAT_CSR_DATA register.

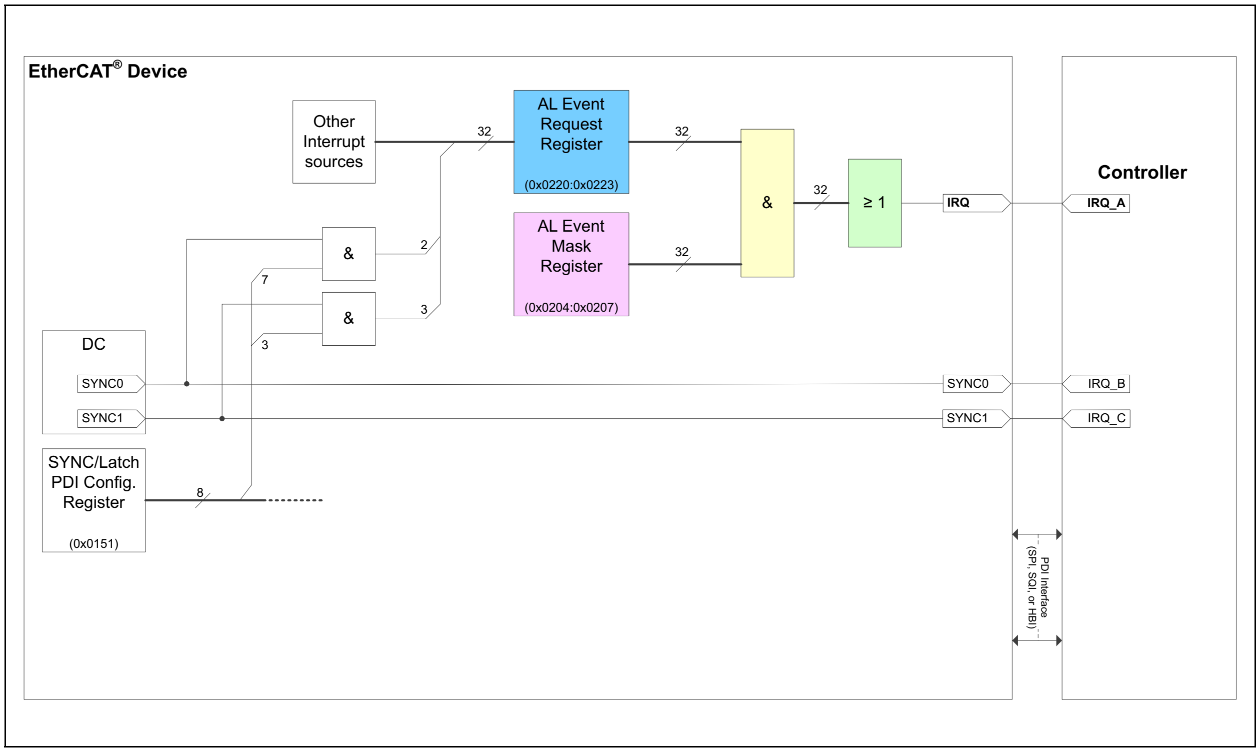
Corruption of data because of interrupt fire can be avoided by disabling the interrupt routine while accessing any ESC registers. However, any missed interrupts because of disabling the interrupts can be monitored by the interrupt line (GPIO) after reading/writing EtherCAT core register.

Interrupt handler in SDK 1.x must be modified according to the host SOC/µC.

2.2.1 PDI INTERRUPT

The programmable system interrupts are generated internally by various device sub-modules and can be configured to generate a single external host interrupt via the IRQ interrupt output pin. The programmable nature of the host interrupt provides the user with the ability to optimize performance dependent upon the application requirements. Buffer type, polarity, and deassertion interval of the IRQ interrupt are modifiable. The IRQ interrupt can be configured as an open-drain output to facilitate the sharing of interrupts with other devices. All internal interrupts are maskable and capable of triggering the IRQ interrupt.

FIGURE 2-3: FUNCTIONAL INTERRUPT MECHANISM



If the application running on the SOC requires AL Event Interrupt, then the IRQ line should be connected to the micro-controller input interrupt. The configuration of IRQ can be done using INTERRUPT CONFIGURATION REGISTER (IRQ_CFG) - 0x54 and INTERRUPT ENABLE REGISTER (INT_EN) - 0x5C. For more details, refer to the LAN9252 data sheet.

2.2.2 DC - SYNC0 AND SYNC1

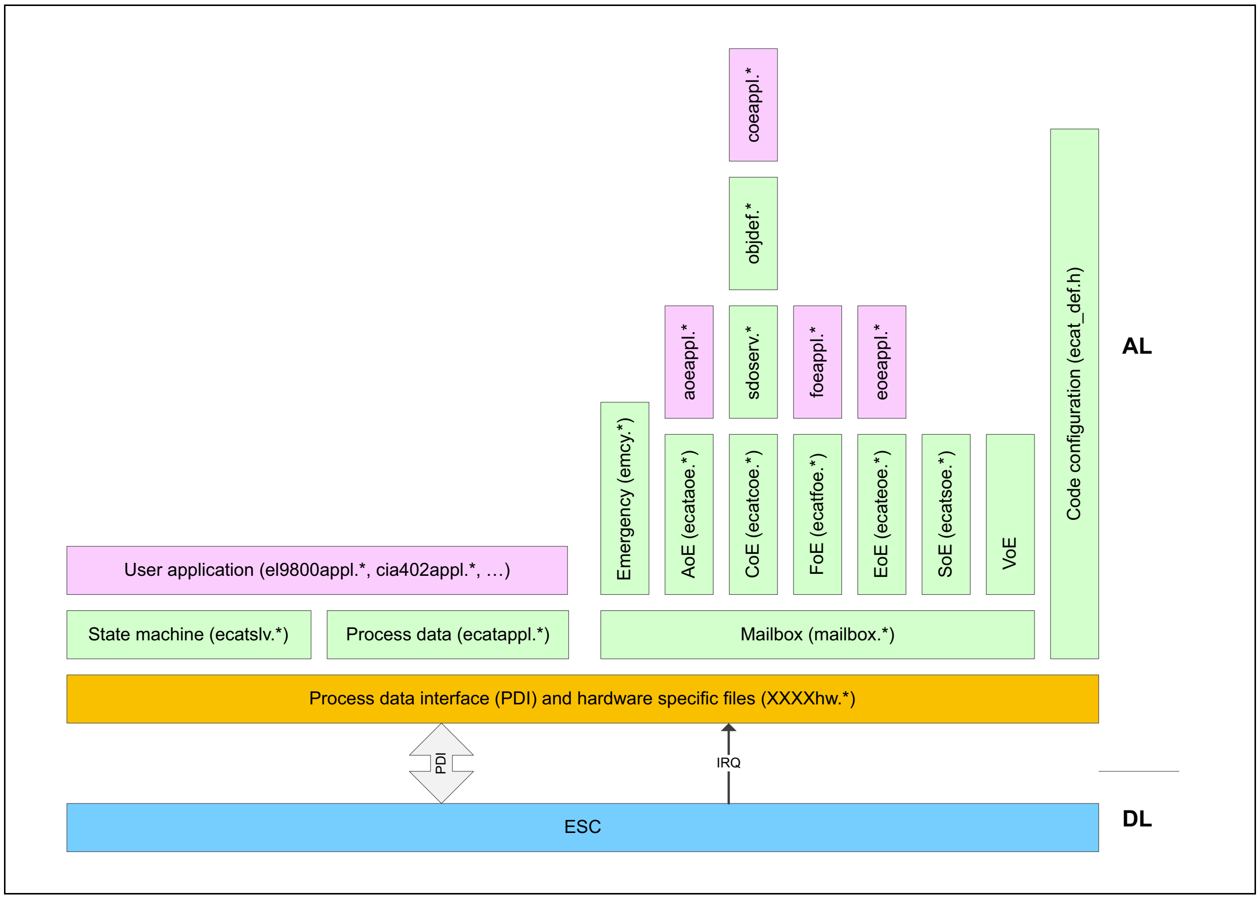
If the application running on the SOC requires a Distributed Clock (DC), then SYNC0 and SYNC1 should be connected to the interrupts lines of the microcontroller. Refer to the LAN9252 data sheet for the configuration of SYNC0 and SYNC1.

2.2.3 TIMER

SSC has a variable that counts every one millisecond, which can be implemented either using timer interrupt or polling method. The interrupt or polling mode can be selected in the SSC Tool before generating the SSC.

3.0 LAN9252 SDK 1.X

FIGURE 3-1: SSC OVERVIEW

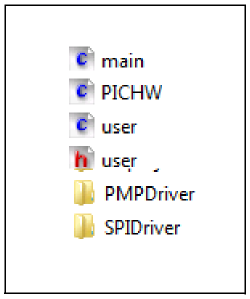

coeappl.*
objdef.*
aoeappl.*
sdoserv.*
foeappl.*
eoeappl.*
Code configuration (ecat_def.h)
AL
Emergency (emcy.*)
AoE (ecataoe.*)
CoE (ecatcoe.*)
FoE (ecatfoe.*)
EoE (ecateoe.*)
SoE (ecatsoe.*)
VoE
User application (el9800appl.*, cia402appl.*, …)
State machine (ecatslv.*)
Process data (ecatappl.*)
Mailbox (mailbox.*)
Process data interface (PDI) and hardware specific files (XXXXhw.*)
PDI
IRQ
ESC
DL

Note: For more information about the SSC tool, refer to EtherCAT Slave Stack Code (SSC) ET9300 web page (www.ethercat.org).

The PDI and hardware-specific files are part of the LAN9252 SDK 1.x. The user application, aoeappl, coeappl, foeappl, and eoeappl have to be defined as per user applications. The PDI interface in SDK 1.3 is based on PIC32MX795F512L. Thus, for any other SOCs, these files must be modified.

SDK 1.3 has two root folders:

- **Common** - The SSC tool generated files can be copied here. The LAN9252_HW has generic PDI APIs defined as per ET9300. This can be easily ported to any other architecture.
- **PIC32** - This folder contains the APIs that are platform-dependent. For controllers other than PIC32MX795F512L, all the APIs must be ported as per the SOCs used.



PMPDriver has the HBI driver (PIC32MX PMP driver) files, whereas SPIDriver has the APIs related to PIC32MX SPI related APIs.

PICHW has the platform-dependent APIs that are defined by ET9300.

4.0 LAN9252 HARDWARE ABSTRACTION LAYER

The functions to be defined to integrate with SSC as per ET9300 are:

- UINT8 HW_Init(void);
- void HW_Release(void);
- UINT16 HW_GetALEventRegister(void);
- UINT16 HW_GetALEventRegister_Isr(void);
- void HW_ResetALEventMask(UINT16 intMask);
- void HW_SetALEventMask(UINT16 intMask);
- void HW_EscRead(MEM_ADDR * pData, UINT16 Address, UINT16 Len);
- void HW_EscReadIsr(MEM_ADDR *pData, UINT16 Address, UINT16 Len);
- void HW_EscWrite(MEM_ADDR *pData, UINT16 Address, UINT16 Len);
- void HW_EscWriteIsr(MEM_ADDR *pData, UINT16 Address, UINT16 Len);
- void HW_EscReadMbxMem(MEM_ADDR *pData, UINT16 Address, UINT16 Len)
- void HW_EscWriteMbxMem(MEM_ADDR *pData, UINT16 Address, UINT16 Len)
- void HW_DisableSyncManChannel(UINT8 channel);
- void HW_EnableSyncManChannel(UINT8 channel);
- TSYNCMAN ESCMEM *HW_GetSyncMan(UINT8 channel);
- UINT16 MainInit(void)
- void MainLoop(void)
- Interrupts for IRQ, SYNC0 and SYNC1
- Timer Interrupt

5.0 LAN9252 SDK APIS

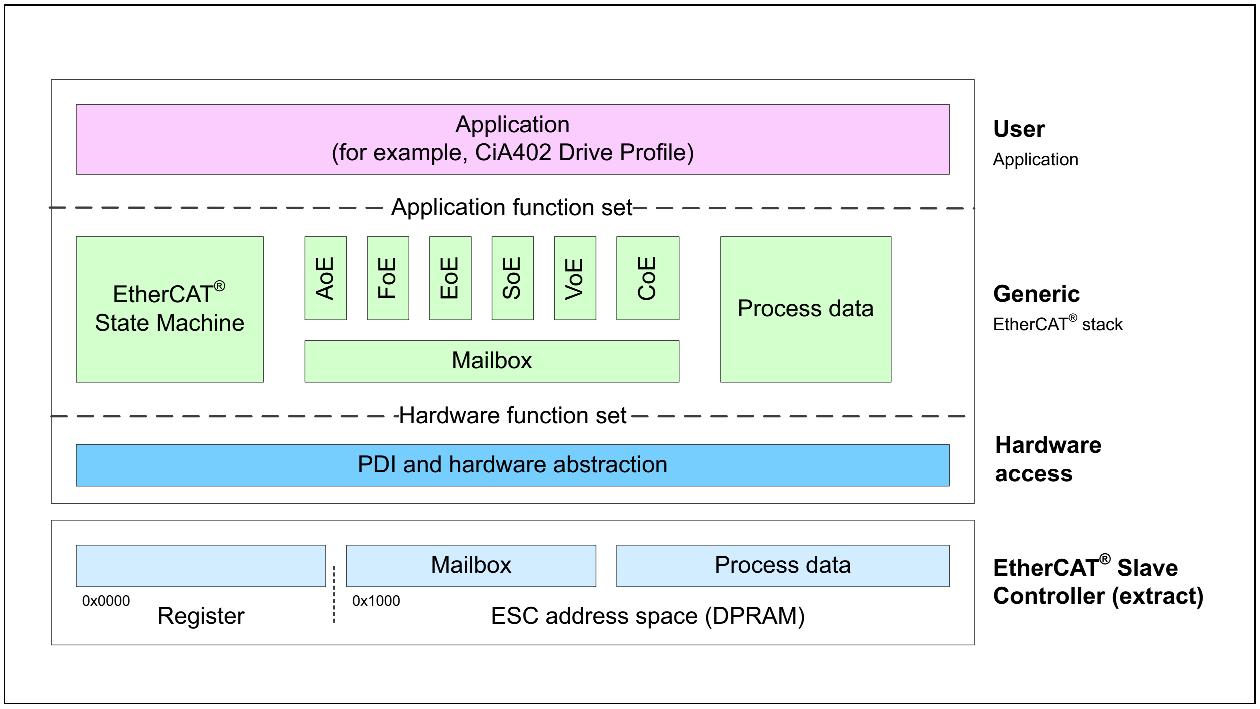
Refer to the ET9300 application note and the SSC tool from ETG for more details about generating the SSC code. SSC defines predefined macros to create appropriate settings for creating the source code for any companion ESCs and SOCs. Table 5-1 has some example macros.

TABLE 5-1: EXAMPLES OF SSC PREDEFINED MACROS

Macro	Description
CONTROLLER_16BIT	This setting is used if the slave code is built for a 16-bit µC.
CONTROLLER_32BIT	This setting is used if the slave code is built for a 32-bit µC.
ESC_16BIT_ACCESS	If this setting is set, then only 16-bit aligned accesses will be performed on the ESC.
ESC_32BIT_ACCESS	If this setting is set, then only 32-bit aligned accesses will be performed on the ESC.
MBX_16BIT_ACCESS	If this setting is set, then the slave code will only access mailbox data 16-bit aligned. If the mailbox data is copied to the local µC memory and CONTROLLER_16BIT is set, then this definition should also be set.

The Microchip_LAN9252_SSC_Config.xml has the predefined macros that are required for generating the SSC code with respect to LAN9252 ESC.

FIGURE 5-1: SSC FUNCTIONAL HIERARCHY



The SSC consists of three parts:

- PDI/HAL
  - Generic LAN9252 driver
  - Low-level PDI driver (specific to host uController)
- Generic EtherCAT stack
- User application

The behavior of the generic EtherCAT stack is described in ETG.1000 Specification [2]. In general, the hardware access implementation needs to support the following features:

- ESC read/write access
- Timer supply (at least 1 ms base tick)

- Calling of timer handler every 1 ms (only required if timer interrupt handling is supported; ECAT_TIMER_INT is set to 1)
- Calling of interrupt specific functions (only required if synchronization is supported)
  - PDI ISR (required if AL_EVENT_SUPPORTED is set to 1)
  - SYNC0 ISR (required if DC_SUPPORTED is set to 1)

The functions and macros that should be provided by the HAL to access LAN9252 and the function that should be provided by the application layer are defined in the following sections.

5.1 LAN9252 Driver

As per the ET9300 application note (EtherCAT Slave Stack Code), the hardware access APIs need to access Generic EtherCAT stack as follows.

5.1.1 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 (PDI) and allocates resources that are required for hardware access.

This function should call from the slave project for initialization of ESC. This function is replaced by UINT16 LAN9252_Init(void).

5.1.2 LAN9252_INIT

Prototype	UINT16 LAN9252_Init(void)
Parameter	void
Return	0 if initialization was successful
	> 0 if error has occurred while initialization
Description	Initializes the host controller (PDI) and allocates resources that are required for hardware access.

This function should call from the slave project for initialization of ESC. Before calling this function, the PDI interface of the slave microcontroller should be initialized.

For initialization of LAN9252, read BYTE-ORDER register (0x64) until it reads 0x87654321. Enable AL Event masking of the AL Event Request register events for mapping to PDI IRQ signal (set 0x93).

This function initializes the LAN9252 PDI interface, configures the LAN9252 IRQ polarity, and provides a function to enable various interrupts, IRQ, SYNC, and Timer.

- PDI_Init_SYNC_Interrupts() - Configure and enable SYNC interrupt
- PDI_Timer_Interrupt() - Configure and enable Timer interrupt
- PDI_IRQ_Interrupt() - Configure and enable IRQ interrupt
- PDI_Enable_Global_interrupt() - Global interrupt

These functions depend on the slave microcontroller.

Refer to the “Interrupt Registers” section of the LAN9252 data sheet.

5.1.3 HW_RELEASE

Prototype	void HW_Release(void)
Parameter	void
Return	void
Description	Release allocated resources

This function is implemented if hardware resources must be released when the sample application stops.

5.1.4 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).

Note: Interrupt should be disabled while reading AL Event register (0x220-0x221).

5.1.5 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).

5.1.6 HW_RESETALEVENTMASK

Prototype	void HW_ResetALEventMask(UINT16 intMask)
Parameter	“intMask” Interrupt mask (disabled interrupt is zero)
Return	void
Description	Performs a logical AND with the AL Event Mask register (0x0204 : 0x0205).

5.1.7 HW_SETALEVENTMASK

Prototype	void HW_SetALEventMask(UINT16 intMask)
Parameter	“intMask” Interrupt mask (enabled interrupt is 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 version 5.10 or previous versions.

5.1.8 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).	

Note: LAN9252 does not support error LED; thus, this feature should be enabled by PDI SOC if needed. The argument RunLed will be neglected (LAN9252 has an in-built support for Run LED).

5.1.9 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.

5.1.10 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 register.	

Find the sync manager offset register for the corresponding “channel” and disable it.

Note: This function is not supported. This function is only required for SSC version 5.10 or previous versions.

5.1.11 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.	

Find the sync manager offset register for the corresponding “channel” and disable it.

Note: This function is not supported. This function is only required for SSC version 5.10 or previous versions.

5.1.12 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 not supported. This function is only required for SSC version 5.10 or previous versions.

5.1.13 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 returns the counter value of a multimedia timer. The timer ticks value (increments/ms) is defined in ECAT_TIMER_INC_P_MS.

This function provides the “PDI_GetTimer()” function to get the current hardware timer value. ECAT_TIMER_INC_P_MS must be defined as per slave microcontroller implementation. SSC calculates different timeouts as per the hardware time ticks and ECAT_TIMER_INC_P_MS.

Note: This function is required if no timer interrupt is supported (ECAT_TIMER_INT = 0) and to calculate the bus cycle time.

5.1.14 HW_CLEARTIMER

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

This function provides the “PDI_ClearTimer” function to clear the hardware timer value.

5.1.15 HW_EEPROMRELOAD

This is only required if EEPROM Emulation is supported. LAN9252 does not support EEPROM Emulation.

5.1.16 READ/WRITE ACCESS

The EtherCAT CSRs provide register level access to the various parameters of the EtherCAT core. LAN9252-related registers can be classified into two categories based on their method of access: direct and indirect.

The directly accessible registers (EtherCAT CSR and Process Data RAM Access Registers) are part of the main system CSRs (LAN9252). These registers provide data or command registers for access to the indirect EtherCAT core registers. Refer to the “EtherCAT CSR and Process Data RAM Access Registers (Directly Addressable)” section of the LAN9252 data sheet for more details.

The indirectly accessible EtherCAT core registers reside within the EtherCAT core and must be accessed indirectly via the EtherCAT CSR Interface Data Register (ECAT_CSR_DATA) and EtherCAT CSR Interface Command Register (ECAT_CSR_CMD). The indirectly accessible EtherCAT core CSRs provide full access to the many configurable parameters of the EtherCAT core. The indirectly accessible EtherCAT core CSRs are accessed at address 0h through 0FFFh and are detailed in the “EtherCAT Core CSR Registers (Indirectly Addressable)” section of the LAN9252 data sheet.

The EtherCAT Core Process Data RAM can be accessed indirectly via ECAT_CSR_DATA and ECAT_CSR_CMD, starting at 1000h.

The EtherCAT Core Process Data RAM can also be accessed more efficiently using the EtherCAT Process RAM Read Data FIFO (ECAT_PRAM_RD_DATA) and EtherCAT Process RAM Write Data FIFO (ECAT_PRAM_WR_DATA). This method provides for multiple DWORDS to be transferred via a FIFO mechanism using a single command and fewer status reads.

For more details, refer to the “ETHERCAT PROCESS RAM READS” and “ETHERCAT PROCESS RAM WRITES” sections of the LAN9252 data sheet.

5.1.16.1 HW_EscRead

Prototype	void HW_EscRead(MEM_ADDR *pData, UINT16 Address, UINT16 Len)	
Parameter	“pData”	Pointer to local destination buffer. The type of pointer depends on the host controller architecture (specified in ecat_def.h or the SSC Tool).
	“Address”	EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in bytes. Only valid addresses are used depending on 8-/16-bit 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 Slave Controller. This function is used to access ESC registers and the DPRAM area.	

This API provides the PDIReadReg() function to access the ESC core register and Process RAM. The implementation of this API depends on the PDI interface and slave microcontroller method to access LAN9252 registers (for example, DMA or accessing the data buffers).

Note: To protect the CSR command and data register (a read or write event from interrupt routine overwrites the CSR command and data register), the interrupt has to be disabled before accessing any ESC memory.

5.1.16.2 HW_EscReadIsr

Prototype	void HW_EscReadIsr(MEM_ADDR *pData, UINT16 Address, UINT16 Len)	
Parameter	"pData"	Pointer to local destination buffer. The type of pointer depends on the host controller architecture (specified in ecat_def.h or the SSC Tool).
	"Address"	EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in bytes. Only valid addresses are used depending on 8-/16-bit 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 Slave Controller. This function is used to access ESC registers and the DPRAM area.	

5.1.16.3 HW_EscReadDWord

Prototype	void HW_EscReadDWord(UINT32 DWordValue, UINT16 Address)	
Parameter	"DWordValue"	Local 32-bit variable where the register value will be stored.
	"Address"	EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in bytes. Only valid 32-bit addresses are used.
Return	void	
Description	Reads two words from the specified address of the EtherCAT Slave Controller.	

5.1.16.4 HW_EscReadDWordIsr

Prototype	void HW_EscReadDWordIsr(UINT32 DWordValue, UINT16 Address)	
Parameter	"DWordValue"	Local 32-bit variable where the register value will be stored.
	"Address"	EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in bytes. Only valid 32-bit 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 Slave Controller.	

5.1.16.5 HW_EscReadWordIsr

Prototype	void HW_EscReadWordIsr(UINT16 WordValue, UINT16 Address)	
Parameter	"WordValue"	Local 16-bit variable where the register value will be stored.
	"Address"	EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in bytes. Only valid 16-bit 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 Slave Controller.	

Note: Only required if ESC_32_BIT_ACCESS is not set.

5.1.16.6 HW_EscReadByte

Prototype	void HW_EscReadByte(UINT8 ByteValue, UINT16 Address)	
Parameter	“ByteValue”	Local 8-bit variable where the register value will be stored.
	“Address”	EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in bytes.
Return	void	
Description	Reads one byte from the EtherCAT Slave Controller.	

Note: Only required if ESC_16BIT_ACCESS and ESC_32BIT_ACCESS are not set.

5.1.16.7 HW_EscReadByteIsr

Prototype	void EscReadByteIsr (UINT8 ByteValue, UINT16 Address)	
Parameter	“ByteValue”	Local 8-bit variable where the register value will be stored.
	“Address”	EtherCAT Slave 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 Slave Controller.	

Note: Only required if ESC_16BIT_ACCESS and ESC_32BIT_ACCESS are not set.

5.1.16.8 HW_EscReadMbxMem

Prototype	void HW_EscReadMbxMem(MEM_ADDR *pData, UINT16 Address, UINT16 Len)	
Parameter	“pData”	Pointer to local destination mailbox buffer. The type of pointer depends on the host controller architecture (specified in ecat_def.h or the SSC Tool).
	“Address”	EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in bytes. Only valid addresses are used depending on 8-/16-bit 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 the slave mailbox memory. If the local mailbox memory is also located in the application memory, this function is equal to HW_EscRead.	

5.1.16.9 HW_EscWrite

Prototype	void HW_EscWrite(MEM_ADDR *pData, UINT16 Address, UINT16 Len)	
Parameter	“pData”	Pointer to local source buffer. The type of pointer depends on the host controller architecture (specified in ecat_def.h or the SSC Tool).
	“Address”	EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in bytes. Only valid addresses are used depending on 8-/16-bit 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 Slave Controller. This function is used to access ESC registers and the DPRAM area.	

This API provides the PDIWriteReg() function to access ESC core register and Process RAM. The implementation of this API depends on the PDI interface and the slave microcontroller method to access LAN9252 registers (for example, DMA or direct access of the data buffers).

Note: To protect the CSR command and data register (a read or write event from interrupt routine overwrites the CSR command and data register), the interrupt must be disabled before accessing any ESC memory.

5.1.16.10 HW_EscWriteIsr

Prototype	void HW_EscWriteIsr (MEM_ADDR *pData, UINT16 Address, UINT16 Len)	
Parameter	“pData”	Pointer to local source buffer. The type of pointer depends on the host controller architecture (specified in ecat_def.h or the SSC Tool).
	“Address”	EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in bytes. Only valid addresses are used depending on 8-/16-bit 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 Slave Controller. This function is used to access ESC registers and the DPRAM area.	

5.1.16.11 HW_EscWriteDWord

Prototype	void HW_EscWriteDWord(UINT32 DWordValue, UINT16 Address)	
Parameter	“DWordValue”	Local 32-bit variable that contains the data to be written to the ESC memory area.
	“Address”	EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in bytes. Only valid 32-bit addresses are used.
Return	void	
Description	Writes one word to the EtherCAT Slave Controller.	

5.1.16.12 HW_EscWriteDWordIsr

Prototype	void HW_EscWriteDWordIsr (UINT32 DWordValue, UINT16 Address)	
Parameter	"DWordValue"	Local 32-bit variable that contains the data to be written to the ESC memory area.
	"Address"	EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in bytes. Only valid 32-bit 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 Slave Controller.	

5.1.16.13 HW_EscWriteWordIsr

Prototype	void HW_EscWriteWordIsr(UINT16 WordValue, UINT16 Address)	
Parameter	"WordValue"	Local 16-bit variable that contains the data to be written to the ESC memory area.
	"Address"	EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in bytes. Only valid 16-bit 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 Slave Controller.	

Note: Only required if ESC_32BIT_ACCESS is not set.

5.1.16.14 HW_EscWriteWord

Prototype	void HW_EscWriteWordIsr(UINT16 WordValue, UINT16 Address)	
Parameter	"WordValue"	Local 16-bit variable that contains the data to be written to the ESC memory area.
	"Address"	EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in bytes. Only valid 16-bit 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 Slave Controller.	

Note: Only required if ESC_32BIT_ACCESS is not set.

5.1.16.15 HW_EscWriteByte

Prototype	void HW_EscWriteByte (UINT8 ByteValue, UINT16 Address)	
Parameter	"ByteValue"	Local 8-bit variable that contains the data to be written to the ESC memory area.
	"Address"	EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in bytes.
Return	void	
Description	Writes one byte to the EtherCAT Slave Controller.	

5.1.16.16 HW_EscWriteByteIsr

Prototype	void HW_ EscWriteByteIsr (UINT8 ByteValue, UINT16 Address)	
Parameter	"ByteValue"	Local 8-bit variable that contains the data to be written to the ESC memory area.
	"Address"	EtherCAT Slave 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 Slave Controller.	

This is only defined if ESC_16BIT_ACCESS and ESC_32BIT_ACCESS are disabled.

5.1.16.17 HW_EscWriteMbxMem

Prototype	void HW_EscWriteMbxMem(MEM_ADDR *pData, UINT16 Address, UINT16 Len)	
Parameter	"pData"	Pointer to local source mailbox buffer. The type of pointer depends on the host controller architecture (specified in ecat_def.h or the SSC Tool).
	"Address"	EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in bytes. Only valid addresses are used depending on 8-/16-bit 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 slave mailbox memory to the ESC memory. If the local mailbox memory is also located in the application memory, this function is equal to HW_EscWrite.	

5.1.17 APPLICATION

All APIs under this section are related to the end application.

5.1.17.1 APPL_Application

Prototype	void APPL_Application(void)
Parameter	Void
Return	void
Description	This function is called by the synchronization ISR or from the main-loop if synchronization is not activated.

5.1.17.2 APPL_GetDeviceID

Prototype	UINT16 APPL_GetDeviceID (void)
Parameter	Void
Return	Explicit Device ID that is written to the AL Status Code register.
Description	This function is called if the master requests the Explicit Device ID. Only required if the slave supports Explicit Device ID handling (EXPLICIT_DEVICE_ID).

5.1.17.3 pAPPL_EEPROM_Read

Prototype: UINT16 (* pAPPL_EEPROM_Read)(UINT32 wordaddr)

Note: LAN9252 does not support EEPROM Emulation.

5.1.17.4 pAPPL_EEPROM_Write

Prototype: UINT16 (* pAPPL_EEPROM_Write)(UINT32 wordaddr)

Note: LAN9252 does not support EEPROM Emulation.

5.1.17.5 pAPPL_EEPROM_Reload

Prototype: UINT16 (* pAPPL_EEPROM_Reload)(void)

Note: LAN9252 does not support EEPROM Emulation.

5.1.17.6 APPL_StartMailboxHandler

Prototype	UINT16 APPL_StartMailboxHandler(void)
Parameter	Void
Return	See the generic ESM return code description.
Description	This function is called during the state transition from INIT to PREOP or INIT to BOOT.

5.1.17.7 APPL_StopMailboxHandler

Prototype	UINT16 APPL_StopMailboxHandler(void)
Parameter	Void
Return	See the generic ESM return code description.
Description	This function is called during the state transition from PREOP to INIT or BOOT to INIT.

5.1.17.8 APPL_StartInputHandler

Prototype	UINT16 APPL_StartInputHandler (UINT16 *pIntMask)
Parameter	pIntMask Value for register 0x204 (AL Event Mask)
Return	See the generic ESM return code description.
Description	This function is called during the state transition from PREOP to SAFEOP (even if no input process data is available).

5.1.17.9 APPL_StopInputHandler

Prototype	UINT16 APPL_StopInputHandler (void)
Parameter	Void
Return	See the generic ESM return code description.
Description	This function is called during the state transition from SAFEOP to PREOP (even if no input process data is available).

5.1.17.10 APPL_StartOutputHandler

Prototype	UINT16 APPL_StartOutputHandler (void)
Parameter	Void
Return	See the generic ESM return code description.
Description	This function is called during the state transition from SAFEOP to OP (even if no output process data is available).

5.1.17.11 APPL_StopOutputHandler

Prototype	UINT16 APPL_StopOutputHandler (void)
Parameter	Void
Return	See the generic ESM return code description.
Description	This function is called during the state transition from OP to SAFEOP (even if no output process data is available).

5.1.17.12 APPL_GenerateMapping

Prototype	UINT16 APPL_GenerateMapping (UINT16 *pInputSize, UINT16 *pOutputSize)
Parameter	Pointer to two 16-bit variables to store the process data size. pInputSize: Input process data (Slave -> Master) pOutputSize: Output process data (Master - > Slave).
Return	See the generic ESM return code description.
Description	This function is called when the EtherCAT master requests the transition from PREOP to SAFEOP. This function calculates the process data size in bytes. The values are required to check the SyncManager settings and for the generic process data handling.

5.1.17.13 APPL_AckErrorInd

Prototype	Void APPL_AckErrorInd(UINT16 stateTrans)
Parameter	stateTrans: Indicates the current state transition
Return	Void
Description	This function is called when the master acknowledges an error.

5.1.17.14 APPL_InputMapping

Prototype	void APPL_InputMapping(UINT16 *pData)
Parameter	pData Pointer to the input process data
Return	Void
Description	This function is called after the application call to map the input process data to the generic stack. The generic stack copies the data to the SM buffer.

Note: The pData contains the input buffer from the Master.

5.1.17.15 APPL_OutputMapping

Prototype	void APPL_OutputMapping(UINT16 *pData)
Parameter	pData Pointer to the output process data
Return	Void
Description	This function is called before the application call to get the output process data.

Note: The pData contains the output buffer to the Master.

5.1.18 VARIABLES

5.1.18.1 ApplicationObjDic

Name	ApplicationObjDic
Type	Array of structure TOBJECT
Description	Only required if the slave supports CAN over EtherCAT (CoE). The variable is defined in the application header file. This array contains the application-specific objects. The last element of this array has the 0xFFFF index.

5.1.18.2 pEEPROM

Name	pEEPROM
Type	UINT8 *
Description	Pointer to the EEPROM buffer. Only required if EEPROM emulation is enabled (ESC_EEPROM_EMULATION = 1). This is defined in ecatappl.h and is set by the application during power-up (before MainInit() is called). The size of the EEPROM buffer is defined by the ESC_EEPROM_SIZE (default 2048) setting.

Note: LAN9252 does not support EEPROM emulation.

5.2 PDI Driver

Different types of Process Data Interface (PDI) are available in LAN9252. See Figure 2-2.

Host bus interface with two user-selectable options are available:

- Indexed register access
- Multiplexed address/data bus

The HBI supports 8-/16-bit operation with big, little, and mixed-endian operations. Two process data RAM FIFOs interface the HBI to the EtherCAT slave controller and facilitate the transferring of process data information between the host CPU and the EtherCAT slave.

An SPI/Quad SPI slave controller provides a low pin count synchronous slave interface that facilitates communication between the device and a host system. The SPI/Quad SPI slave allows access to the system CSRs, internal FIFOs, and memories. It supports single and multiple registers read and write commands with incrementing, decrementing, and static addressing. Single, Dual and Quad bit lanes are supported with a clock rate of up to 80 MHz.

5.2.1 GENERAL PDI API

All the APIs in this section are designed to work with 32-bit processors or controllers.

5.2.1.1 PDI_Init

Prototype	void PDI_Init();
Parameter	void
Return	void
Description	Initializes PDI interface

Note: This API should be called before accessing the PDI interface. The implementation of this API depends on the slave microcontroller.

5.2.2 READ/WRITE API

The EtherCAT CSRs provide register level access to the various parameters of the EtherCAT core. EtherCAT-related registers can be classified into two main categories based on their method of access: direct and indirect.

The LAN9252SDK 1.x is compatible with PIC32MX795F512L.

5.2.2.1 Read/Write Directly Addressable Registers

5.2.2.1.1 PDIWriteLAN9252DirectReg

Prototype	void PDIWriteLAN9252DirectReg(UINT32 Val, UINT16 Address)	
Parameter	Val	DWORD value
	Address	Address of the directly addressable register
Return	void	
Description	Writes “val” to the directly addressable register.	

Note: Either the HBI or SPI/SQI interface can be used to access LAN9252 registers.

5.2.2.1.2 PDIReadLAN9252DirectReg

Prototype	UINT32 PDIReadLAN9252DirectReg(UINT16 Address)	
Parameter	Address	Address of the directly addressable register
Return	UINT32	The read DWORD value
Description	Reads the directly addressable register.	

Note: Either the HBI or SPI/SQI interface can be used to access LAN9252 registers.

5.2.2.2 Read/Write Indirectly Addressable Registers (EtherCAT® Core Registers)

5.2.2.2.1 PDIReadReg

Prototype	void PDIReadReg(UINT8 *ReadBuffer, UINT16 Address, UINT16 Count)	
Parameter	ReadBuffer	Pointer to the read buffer
	Address	Address of the directly addressable register
	Count	Number of bytes to read
Return	void	
Description	This function reads the ESC registers using LAN9252 CSR or FIFO.	

Note: Either the HBI or SPI/SQI interface can be used to access LAN9252 registers.

5.2.2.2.2 PDIWriteReg

Prototype	void PDIWriteReg(UINT8 *WriteBuffer, UINT16 Address, UINT16 Count)	
Parameter	WriteBuffer	Pointer to the buffer to be written
	Address	Address of the directly addressable register
	Count	Number of bytes to read
Return	void	
Description	Writes the ESC registers using LAN9252 CSR or FIFO.	

Note: Either the HBI or SPI/SQI interface can be used to access LAN9252 registers.

5.2.2.2.3 Host Bus Interface

The following APIs use PMPReadDWord and PMPWriteDWord to read and write LAN9252 registers. This API must be defined as per the host bus interface used in the SSC controller.

Refer to the “HBI Bus Interface” section of the LAN9252 data sheet.

- *PMPReadRegUsingCSR*

Prototype	void PMPReadRegUsingCSR(UINT8 *ReadBuffer, UINT16 Address, UINT8 Count)	
Parameter	ReadBuffer	Pointer to the read buffer
	Address	Address of the directly addressable register
	Count	Number of bytes to read
Return	void	
Description	Reads the EtherCAT core registers using LAN9252 CSR registers.	

Note: For more details, refer to the “EtherCAT Process RAM Reads” section of the LAN9252 data sheet.

- *PMPReadPDRamRegister*

Prototype	void PMPReadPDRamRegister(UINT8 *ReadBuffer, UINT16 Address, UINT16 Count)	
Parameter	ReadBuffer	Pointer to the read buffer
	Address	Address of the directly addressable register
	Count	Number of bytes to read
Return	void	
Description	Reads the PDRAM using LAN9252 FIFO.	

Note: Refer to the “EtherCAT CSR and Process Data RAM Access Registers (Directly Addressable)” section of the LAN9252 data sheet.

- *PMPWriteRegUsingCSR*

Prototype	void PMPWriteRegUsingCSR(UINT8 *WriteBuffer, UINT16 Address, UINT8 Count)	
Parameter	WriteBuffer	Pointer to the buffer to be written
	Address	Address of the directly addressable register
	Count	Number of bytes to read
Return	void	
Description	Writes the ESC registers using LAN9252 CSR.	

Note: For more details, refer to the “EtherCAT Process RAM Writes” section of the LAN9252 data sheet.

- *PMPWritePDRamRegister*

Prototype	void PMPWritePDRamRegister(UINT8 *WriteBuffer, UINT16 Address, UINT16 Count)	
Parameter	WriteBuffer	Pointer to the buffer to be written
	Address	Address of the directly addressable register
	Count	Number of bytes to read
Return	void	
Description	Writes the ESC registers using LAN9252 FIFO.	

Note: Refer to the “EtherCAT CSR and Process Data RAM Access Registers (Directly Addressable)” section of the LAN9252 data sheet.

5.2.2.2.4 SPI

- *SPIReadRegUsingCSR*

Prototype	void SPIReadRegUsingCSR (UINT8 *ReadBuffer, UINT16 Address, UINT8 Count)	
Parameter	ReadBuffer	Pointer to the read buffer
	Address	Address of the directly addressable register
	Count	Number of bytes to read
Return	void	
Description	Reads the EtherCAT core registers using LAN9252 CSR registers.	

Note: For more details, refer to the “EtherCAT Process RAM Reads” section of the LAN9252 data sheet.

- *SPIReadPDRamRegister*

Prototype	void SPIReadPDRamRegister (UINT8 *ReadBuffer, UINT16 Address, UINT16 Count)	
Parameter	ReadBuffer	Pointer to the read buffer
	Address	Address of the directly addressable register
	Count	Number of bytes to read
Return	void	
Description	Reads the PDRAM using LAN9252 FIFO.	

Note: Refer to the “EtherCAT CSR and Process Data RAM Access Registers (Directly Addressable)” section of the LAN9252 data sheet.

- *SPIWriteRegUsingCSR*

Prototype	void SPIWriteRegUsingCSR (UINT8 *WriteBuffer, UINT16 Address, UINT8 Count)	
Parameter	WriteBuffer	Pointer to the buffer to be written
	Address	Address of the directly addressable register
	Count	Number of bytes to read
Return	void	
Description	Writes the ESC registers using LAN9252 CSR.	

Note: For more details, refer to the “EtherCAT Process RAM Writes” section of the LAN9252 data sheet.

- *SPIWritePDRamRegister*

Prototype	void PMPWritePDRamRegister(UINT8 *WriteBuffer, UINT16 Address, UINT16 Count)	
Parameter	WriteBuffer	Pointer to the buffer to be written
	Address	Address of the directly addressable register
	Count	Number of bytes to read
Return	void	
Description	Writes the ESC registers using LAN9252 FIFO.	

Note: Refer to the “EtherCAT CSR and Process Data RAM Access Registers (Directly Addressable)” of the LAN9252 data sheet.

- *SPIReadDWord*

Prototype	UINT32 SPIReadDWord (UINT16 Address)	
Parameter	Address	Address of LAN9252 CSR register
Return	The read value (DWORD)	
Description	This function reads 4 bytes of data from LAN9252 CSR register.	

Note: For more details, refer to the “SPI/SQI Slave” section of the LAN9252 data sheet.

- *SPIWriteDWord*

Prototype	void SPIWriteDWord (UINT16 Address, UINT32 Val)	
Parameter	Address	Address of LAN9252 CSR register (not ESC register)
	Val	4-byte value
Return	void	
Description	This function writes 4 bytes of data to the corresponding address of the LAN9252 CSR register (not ESC register).	

Note: For more details, refer to the “SPI/SQI Slave” section of the LAN9252 data sheet.

- *SPIReadBurstMode*

Prototype	UINT32 SPIReadBurstMode (UINT16 Address)
Parameter	void
Return	The read value(DWORD)
Description	This function reads 4 bytes of data from LAN9252 CSR register.

Note: This function does not assert CS. For more details, refer to the “SPI/SQI Slave” section of the LAN9252 data sheet.

- *SPIWriteBurstMode*

Prototype	UINT32 SPIWriteBurstMode (UINT16 Address)	
Parameter	Address	Address of LAN9252 CSR register
Return	The read value (DWORD)	
Description	This function writes 4 bytes of data to the corresponding address of LAN9252 CSR register.	

Note: This function does not assert CS. For more details, refer to the “SPI/SQI Slave” section of the LAN9252 data sheet.

- *SPIWriteBytes*

Prototype	void SPIWriteBytes(UINT16 Address, UINT8 *Val, UINT8 nLength)	
Parameter	Address	Address of CSR register
	Val	Pointer to the write buffer
	nLength	Length of the buffer
Return	void	
Description	This function writes the LAN9252 CSR registers.	

Note: For more details, refer to the “SPI/SQI Slave” section of the LAN9252 data sheet.

5.3 Slave Controller Dependent APIs

The definition of the following APIs depends on the SSC controller used.

5.3.1 INTERRUPT API

5.3.1.1 PDI_Timer_Interrupt

Prototype	void PDI_Timer_Interrupt()
Parameter	void
Return	void
Description	This function configures and enables the TIMER interrupt for 1 ms.

5.3.1.2 PDI_IRQ_Interrupt

Prototype	void PDI_IRQ_Interrupt()
Parameter	void
Return	void
Description	This function configures and enables the interrupt for IRQ.

5.3.1.3 PDI_Enable_Global_interrupt

Prototype	void PDI_Enable_Global_interrupt()
Parameter	void
Return	void
Description	Enables interrupts in the slave controller.

5.3.1.4 PDI_Disable_Global_Interrupt

Prototype	UINT32 PDI_Disable_Global_Interrupt()
Parameter	void
Return	The previous state of the interrupt Status
Description	This function reads the current interrupt status and disables all interrupt requests.

Note: PDI_Restore_Global_Interrupt uses the previous state of the interrupt status to restore the interrupt status.

5.3.1.5 PDI_Init_SYNC_Interrupts

Prototype	void PDI_Init_SYNC_Interrupts()
Parameter	void
Return	void
Description	The function configures and enable SYNC0 and SYNC1 interrupts.

5.3.2 INTERRUPT HANDLERS

The following functions are provided by the generic SSC (defined in ecatappl.h) and must be called from the hardware access layer.

5.3.2.1 ECAT_CheckTimer

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

Note: The timer ISR must be implemented as per the slave controller.

5.3.2.2 PDI_Isr

Prototype	void PDI_Isr (void)
Parameter	void
Return	void
Description	This function must be called from the PDI ISR. For the PDI-specific pin naming and the interrupt generation logic, please refer to the LAN9252 data sheet. To support PDI interrupt handling, it is also required to set AL_EVENT_ENABLED to 1.

Note: IRQ ISR must be implemented as per the slave controller.

5.3.2.3 Sync0_Isr

Prototype	void Sync0_Isr (void)
Parameter	void
Return	void
Description	This function must 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 must be set.

Note: SYNC0 ISR must be implemented as per the slave controller.

5.3.2.4 Sync1_Isr

Prototype	void Sync1_Isr (void)
Parameter	void
Return	void
Description	This function must 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 must be set.

Note: SYNC1 ISR must be implemented as per the slave controller.

5.3.3 HOST BUS INTERFACE

Host Bus Interface (HBI) can be in the Indexed or Multiplexed mode.

In Multiplexed Address or Data mode, the address and the endianness select inputs are shared with the data bus. Two methods are supported: a single-phase address that utilizes up to 16 address or data pins and a dual-phase address that utilizes only the lower 8 data bits.

In Indexed Address mode, access to the internal registers and memory of the device are indirectly mapped using Index and Data registers. The desired internal address is written into the device at a specific offset. The value written is then used as the internal address when the associate Data register address is accessed. Three Index or Data register sets are provided allowing for multi-threaded operation without the concern of one thread corrupting the Index set by another thread.

For more details, refer to the “HBI Bus Interface” section of the LAN9252 data sheet.

5.3.3.1 PMPReadDWord

Prototype	UINT32 PMPReadDWord (UINT16 Address)	
Parameter	Address	Address of LAN9252 CSR register
Return	The read value (DWORD)	
Description	This function reads 4 bytes of data from LAN9252 CSR register.	

Note: This API must be implemented as per the slave controller if PDI interface is a host bus interface. This function will call after initialization of the HBI of the slave controller.

5.3.3.2 PMPWriteDWord

Prototype	void PMPWriteDWord (UINT16 Address, UINT32 Val)	
Parameter	Address	Address of LAN9252 CSR register (not ESC register)
	Val	4-byte value
Return	void	
Description	This function writes 4 bytes of data to the corresponding address of LAN9252 CSR register (not ESC register).	

Note: This API must be implemented as per the slave controller if PDI interface is a host bus interface. This function will call after initialization of the HBI of the slave controller.

5.3.4 SPI

The SPI/SQI Slave module provides a low pin count synchronous slave interface that facilitates communication between the device and a host system. The SPI/SQI Slave allows access to the System CSRs and internal FIFOs and memories. It supports single and multiple registers read and write commands with incrementing, decrementing, and static addressing.

For more details, refer to the “SPI/SQI SLAVE” section of the LAN9252 data sheet.

5.3.4.1 SPIReadByte

Prototype	void SPIReadByte ()
Parameter	void
Return	void
Description	This function reads 1 byte of data from the LAN9252 CSR register. Refer to the LAN9252 data sheet on how to read using SPI.

5.3.4.2 SPIWriteByte

Prototype	void SPIWriteByte(UINT8 data)	
Parameter	data	Data to write
Return	void	
Description	This function writes 1 byte of data to the corresponding address of LAN9252 CSR register.	

5.3.4.3 Macros

The following macros are related to the SOCs where SSC runs.

- CSLOW () – This macro drives the CS line low.
- CSHIGH () – This macro drives the CS line high.
- DISABLE_ESC_INT – Disables the IRQ interrupt
- ENABLE_ESC_INT – Enables the IRQ interrupt
- ECAT_TIMER_INC_P_MS – The timer ticks value (increments/ms)

6.0 SSC TOOL CONFIGURATION FILE

The Microchip_LAN9252_SSC_Config.xml has the predefined macros that are required for generating the SSC code about LAN9252 ESC. The default values available in this config file can be updated using the SSC tool or modifying the same file itself.

Note: For more information about the SSC tool, refer to EtherCAT Slave Stack Code (SSC) ET9300 web page (www.ethercat.org).

APPENDIX A: REVISION HISTORY

TABLE A-1: REVISION HISTORY

Revision	Section/Figure/Entry	Correction
DS00002655A (02-23-18)	Initial release	

THE MICROCHIP WEB SITE

Microchip provides online support via our WWW site at www.microchip.com. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

CUSTOMER CHANGE NOTIFICATION SERVICE

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at www.microchip.com. Under "Support", click on "Customer Change Notification" and follow the registration instructions.

CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://microchip.com/support

ISBN: 978-1-5224-2711-7

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
== ISO/TS 16949 ==

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

China - Chengdu
Tel: 86-28-8665-5511

China - Chongqing
Tel: 86-23-8980-9588

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

Japan - Osaka
Tel: 81-6-6152-7160

Japan - Tokyo
Tel: 81-3-6880- 3770

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-67-3636

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-7289-7561

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820