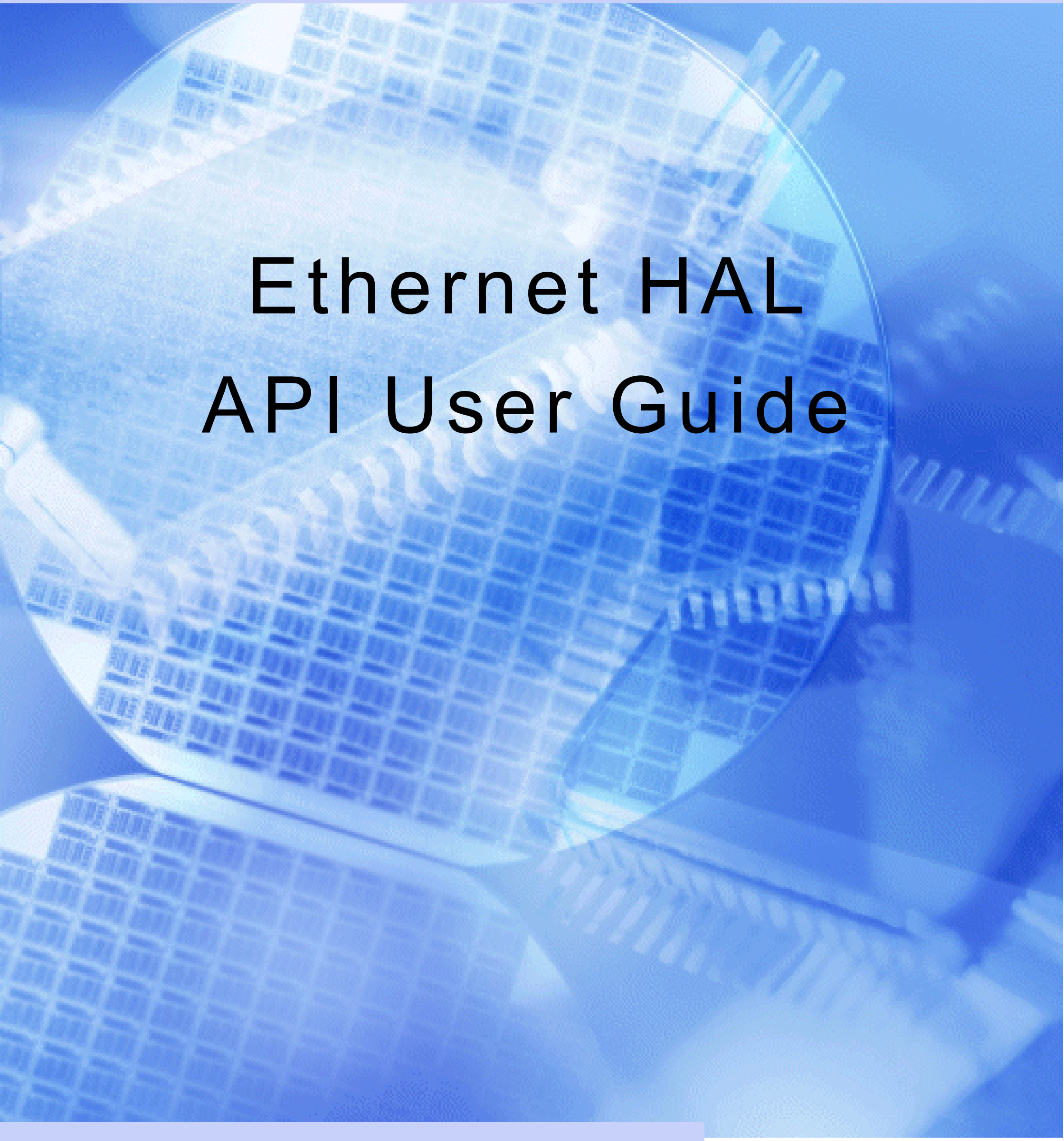
User Guide, v2.4, 24 Jan 2006



Ethernet HAL API User Guide

Microcontrollers

Never stop thinking.

Edition v2.4, 24 Jan 2006

Published by Infineon Technologies India pvt. Ltd.,
ITPL,
Bangalore, INDIA

User Guide, v2.4, 24 Jan 2006

Ethernet HAL API User Guide

Microcontrollers

Never stop thinking.

Ethernet
Revision History 24 Jan 2006 v2.4

Table 1

Page	Subjects (major changes since last revision)
ALL	First Draft
ALL	Updated for eLinux (Section 4), Application Note on Interrupt Service Routine (Section 6) added and version number changed to 2.0 for the Implementation Release SW_M7.
Section 3	Version 2.2.0 Removed the DMUR _ALIGN configuration parameter.
Previous version 2.2.0	Added Limitation chapter 11 and change in ETH_MAC_TX_CONF and ETH_MAC_RX_CONF structure.
Version 2.4	Details regarding data cache fix is added, and details with respect to eLinux is removed.

Author:
Jayashree Badarinath

1 Ethernet HAL Introduction . . . 1

2 Ethernet HAL Application Program Interface . . . 2
2.1 ETH_API_V_MAJ macro . . . 2
2.2 ETH_API_V_MIN macro . . . 2
2.3 ETH_CAM_TBL_LEN macro . . . 2
2.4 ETH_MACADDR_CONV_FACTOR . . . 3
2.5 ETH_DEVICE typedef . . . 3
2.6 ETH_CTRL_CODE . . . 6
2.7 ETH_MAC_CONF Structure . . . 8
2.8 ETH_MAC_TX_CONF Structure . . . 9
2.9 ETH_MAC_RX_CONF Structure . . . 10
2.10 ETH_PHY_CONF Structure . . . 11
2.11 ETH_PHY_CTRL Structure . . . 12
2.12 ETH_CAM_DATA Structure . . . 13
2.13 ETH_CAM_COMMAND . . . 15
2.14 ETH_CAM_CONF Structure . . . 16
2.15 ETH_CAM_UDATACONF Union . . . 17
2.16 ETH_CAM_CTRL Structure . . . 18
2.17 ETH_CAM_PROG Structure . . . 18
2.18 ETH_RB_CONF Structure . . . 20
2.19 ETH_TB_CONF Structure . . . 21
2.20 ETH_RXDESC Structure . . . 21
2.21 ETH_DMUR_CONF Structure . . . 24
2.22 ETH_DMUT_CONF Structure . . . 25
2.23 ETH_TXDESC Structure . . . 26
2.24 ETH_COM_PARAMS Structure . . . 28
2.25 ETH_RETURN_NUM Structure . . . 30
2.26 ETH_TRANSFER Structure . . . 31
2.27 ETH_STAT Structure . . . 32
2.28 ETH_STAT_INF Structure . . . 35
2.29 ETH_gpio_alloc . . . 36
2.30 ETH_gpio_free . . . 36

3 Ethernet API Functions . . . 38
3.1 ETH_initialise_dev . . . 38
3.2 ETH_initialise_mac . . . 39
3.3 ETH_initialise_phy . . . 40
3.4 ETH_check_link . . . 41
3.5 ETH_read_phy . . . 43
3.6 ETH_camctrl_caseAodd . . . 44
3.7 ETH_camctrl_caseAeven . . . 45
3.8 ETH_camctrl_caseB . . . 46
3.9 ETH_initialise_rb . . . 47

3.10 ETH_initialise_tb 48
3.11 ETH_initialise_dmur 49
3.12 ETH_initialise_dmut 50
3.13 ETH_terminate_dev 51
3.14 ETH_terminate_mac 52
3.15 ETH_terminate_rb 53
3.16 ETH_terminate_tb 53
3.17 ETH_terminate_dmur 54
3.18 ETH_terminate_dmut 55
3.19 ETH_abort 56
3.20 ETH_status_dev 57
3.21 ETH_control_dev 58
3.22 ETH_ctrl_cam 60
3.23 ETH_ctrl_mac_tx 61
3.24 ETH_ctrl_mac_rx 62
3.25 ETH_ctrl_mac_phy 63
3.26 ETH_ctrl_tb 64
3.27 ETH_ctrl_dmur 65
3.28 ETH_ctrl_dmut 66
3.29 ETH_read 67
3.30 ETH_reg_tx_cb 68
3.31 ETH_write 69

4 Configure/Optimize the Ethernet HAL 71
4.1 Configuring the Ethernet HAL 71
4.2 Ethernet Driver HAL configuration parameters 71
4.2.1 ETH_CFG_DEV_CHK macro 71
4.2.2 ETH_CFG_INIT_CHK macro 72
4.2.3 ETH_CFG_STAT_LOG macro 72
4.2.4 ETH_CFG_USE_CAM macro 73
4.3 API Function Exclusion 73
4.3.1 ETH_CFG_FUNC_TERMINATE macro 74
4.3.2 ETH_CFG_FUNC_ABORT macro 74
4.3.3 ETH_CFG_FUNC_STATUS macro 75
4.3.4 ETH_CFG_FUNC_CONTROL macro 75
4.3.5 ETH_CFG_FUNC_READ macro 76
4.3.6 ETH_CFG_FUNC_WRITE macro 76
4.3.7 ETH_CFG_FUNC_CTRL_MAC macro 77
4.3.8 ETH_CFG_FUNC_CTRL_CAM macro 77
4.3.9 ETH_CFG_FUNC_CTRL_MAC_TX macro 78
4.3.10 ETH_CFG_FUNC_CTRL_MAC_RX macro 78
4.3.11 ETH_CFG_FUNC_CTRL_MAC_PHY macro 79
4.3.12 ETH_CFG_FUNC_CTRL_TB macro 79

4.3.13 ETH_CFG_FUNC_CTRL_DMUR macro 80
4.3.14 ETH_CFG_FUNC_CTRL_DMUT macro 80
4.3.15 ETH_CFG_FULLDUPLEX macro 81
4.3.16 ETH_CFG_MAC_LOOPBACK macro 81
4.3.17 ETH_CFG_PHYDEV_LOOPBACK macro 82
4.3.18 ETH_CFG_SPEED macro 82
4.3.19 ETH_CFG_USE_PAUSE macro 83
4.3.20 ETH_CFG_PAUSE_TIME macro 83
4.3.21 ETH_CFG_USE_CAM macro 83
4.3.22 ETH_CFG_NEGATIVECAM macro 84
4.3.23 ETH_CFG_UNICAST macro 84
4.3.24 ETH_CFG_MULTICAST macro 85
4.3.25 ETH_CFG_BROADCAST macro 85
4.3.26 ETH_CFG_MACADDR_STRT_LOC macro 86
4.3.27 ETH_CFG_NUM_OF_MACADDR macro 86
4.3.28 ETH_CFG_ENBL_MACADDRS macro 86
4.3.29 ETH_CFG_ADDPAD macro 86
4.3.30 ETH_CFG_ADDCRC macro 87
4.3.31 ETH_CFG_XDEFER macro 87
4.3.32 ETH_CFG_SQECHECK macro 87
4.3.33 ETH_CFG_RCVSHORT macro 88
4.3.34 ETH_CFG_RCVLONG macro 88
4.3.35 ETH_CFG_STRIPCRC macro 88
4.3.36 ETH_CFG_PASSCTRLPKTS macro 89
4.3.37 ETH_CFG_NOTCHECKCRC macro 89
4.3.38 ETH_CFG_AUTONEGOTIATE macro 89
4.3.39 ETH_CFG_TBFTC macro 90
4.3.40 ETH_CFG_TBRTC macro 90
4.3.41 ETH_CFG_BUFFSIZE macro 91
4.3.42 ETH_CFG_DMUR_ENDIAN macro 91
4.3.43 ETH_CFG_DMUT_ENDIAN macro 92
4.3.44 ETH_CFG_DMUR_DESC_NUM macro 92
4.3.45 ETH_CFG_DMUR_DBUFF_SIZE macro 92
4.3.46 ETH_CFG_DMUT_DESC_NUM macro 92

5 Cache memory usage for Ethernet LLD 93

6 Application Examples 95
6.1 Example illustrating the initialisation and basic operation of the driver . . . 95
6.2 Example illustrating the programming of the CAM at run time. 100
6.3 Example illustrating the use of the ‘fe’ bit in the receive path 101
6.4 Changing communication parameters + GPIO cfg example 103
6.5 Application note on the disabling of interrupts in the Interrupt Service Routine
104

7 Related Documentation 105

8 Appendix A - Infineon IFX types 106

9 Appendix B - The System HAL 107
9.1 SYS HAL Configurable Parameters 107
9.1.1 System Clock Frequency 107
9.1.1.1 SYS_CFG_USB_DEVICE_ENABLE macro 107
9.1.1.2 SYS_CFG_USB_ONCHIP_CLK macro 107
9.1.1.3 SYS_CFG_USB_CLK_DIVISOR macro 108
9.1.1.4 SYS_CFG_OSC_FREQ macro 108
9.1.1.5 SYS_CFG_CLK_MODE macro 108
9.1.1.6 SYS_CFG_FREQ_SEL macro 108
9.1.1.7 SYS_CFG_KDIV macro 109
9.1.1.8 SYS_CFG_PDIV macro 109
9.1.1.9 SYS_CFG_NDIV macro 109
9.1.1.10 SYS_CFG_FIX_TC1130A_BUG macro 109
9.1.2 Interrupt Configuration 110
9.1.2.1 SYS_ETH_MACRX0 macro 110
9.1.2.2 SYS_ETH_MACRX1 macro 110
9.1.2.3 SYS_ETH_MACTX0 macro 110
9.1.2.4 SYS_ETH_MACTX1 macro 110
9.1.2.5 SYS_ETH_MACRB0 macro 110
9.1.2.6 SYS_ETH_MACRB1 macro 111
9.1.2.7 SYS_ETH_MACTB0 macro 111
9.1.2.8 SYS_ETH_DMUR macro 111
9.1.2.9 SYS_ETH_MACDMUT macro 111
9.2 General Purpose I/O Configuration 111
9.2.1 SYS_GPIO_ETH_TX0 macro 112
9.2.2 SYS_GPIO_ETH_TX1 macro 112
9.2.3 SYS_GPIO_ETH_TX2 macro 112
9.2.4 SYS_GPIO_ETH_TX3 macro 112
9.2.5 SYS_GPIO_ETH_TXER macro 112
9.2.6 SYS_GPIO_ETH_TXEN macro 112
9.2.7 SYS_GPIO_ETH_MDC macro 113
9.2.8 SYS_GPIO_ETH_RXDV macro 113
9.2.9 SYS_GPIO_ETH_CRS macro 113
9.2.10 SYS_GPIO_ETH_COL macro 113
9.2.11 SYS_GPIO_ETH_RX0 macro 113
9.2.12 SYS_GPIO_ETH_RX1 macro 113
9.2.13 SYS_GPIO_ETH_RX2 macro 113
9.2.14 SYS_GPIO_ETH_RX3 macro 113

10 Limitations 114

1 Ethernet HAL Introduction

The Ethernet HAL (Hardware Abstraction Layer) forms one part out of a larger system of software modules designed to buffer application software from hardware specific implementation details. The Infineon Technologies modular HAL approach however goes beyond simply providing a standardised API for a type of device, each HAL is designed to work as part of a larger system. System resources are reserved by the HAL's using a system hardware abstraction layer meaning that runtime conflicts are avoided and different peripherals may use the same resources at different points in the application. If the peripheral clock is changeable then the HAL's will normally support on the fly clock changing allowing the clock speed to be changed for power saving etc. Data transfer requests from the application software can be queued so that the application does not need to wait for one transfer to end before the next can be started.

In cases where these features are not desirable, possibly due to runtime efficiency or code size constraints, they can simply be removed by modifying a single configuration file and recompiling the HAL, meaning there is no unnecessary code overhead.

In summary the modular HAL system provides the following advantages:

–No extra hardware specific code is required

–Pre-tested code modules are available

–Hardware can be exchanged with little or no application software modifications

–Power saving features incorporated in the HAL

–System resource conflicts are automatically avoided

–Data transfer requests can be queued.

–HAL's are highly configurable

–Porting to different hardware is easy

–Standardised API can be used for development

–Application and hardware dependant developments can go in parallel assuming a standard API

2 Ethernet HAL Application Program Interface

This section defines the interface between the peripheral hardware abstraction layer and the application software. It defines the constants, typedef names, function names and limitations of the HAL. The Ethernet HAL API utilizes a range of constants and typedef names in order to interact in a logical way with the application program. The first section of this chapter will look at these constants and data types.

Please refer to Appendix A - Infineon IFX types for details on the IFX data types.

2.1 ETH_API_V_MAJ macro

#define ETH_API_V_MAJ

Defined in: eth_api.h

API Major Version Number, ETH_API_V_MAJ is defined as the major version of this API which the Ethernet HAL supports. This version is defined as 0.1. Application software may check this field to determine if the HAL API version is acceptable.

2.2 ETH_API_V_MIN macro

#define ETH_API_V_MIN

Defined in: eth_api.h

API Minor Version Number, ETH_API_V_MIN is defined to the minor version of this API which the Ethernet HAL supports. This version is defined as 0.1. Application software may check this field to determine if the HAL API version is acceptable

2.3 ETH_CAM_TBL_LEN macro

#define ETH_CAM_TBL_LEN

Defined in: eth_api.h

CAM Table Length, the CAM contains 20 MAC addresses each of 6 bytes that can be programmed i.e. a total of 120 bytes. Before programming the CAM these addresses are stored in an array of IFX_UINT32. Therefore the 120 bytes are mapped into an integer array mac_addrs[ ] of length ETH_CAM_TBL_LEN. The struct ETH_CAM_DATA uses the mac_addrs[ ].

2.4 ETH_MACADDR_CONV_FACTOR

This factor is used to convert 6 bytes MAC addresses to UINT arrays macro

#define ETH_MACADDR_CONV_FACTOR, This factor is used to convert 6 bytes MAC addresses to UINT arrays

Defined in: eth_api.h

2.5 ETH_DEVICE typedef

This indicates the Device ID

Defined in: eth_api.h

Comments

ETH_DEVICE is used in the API wherever a device must be selected. This is required because many ETH peripherals may be implemented in the same system. ETH_DEVICE is simply a typedef name which is defined as IFX_UINT8

ETH_STATUS

enum ETH_STATUS {

ETH_ERR,

ETH_ERR_RES,

ETH_ERR_RES_INT,

ETH_ERR_RES_MEM,

ETH_ERR_RES_IO,

ETH_ERR_NOT_SUPPORTED,

ETH_ERR_NOT_SUPPORTED_HW,

ETH_ERR_UNKNOWN_DEV,

ETH_ERR_NOT_INITIALISED,

ETH_ERR_NOT_INITIALISED is returned if an API function is called before the HAL has been successfully initialized. This checking may be configured out to improve runtime performance, see Configuring the Ethernet HAL for information

};

Ethernet Status Enumeration

Defined in: eth_api.h

Members

ETH_ERR

ETH_SUCCESS indicates that an operation completed successfully.

ETH_ERR_RES

ETH_ERR is used to indicate that an unspecified error was encountered by the HAL. ETH_ERR will only be used as a last resort when the HAL is unable o describe the error using a more specific error code.

ETH_ERR_RES_INT

ETH_ERR_RES is used to indicate that the Ethernet HAL was unable to allocate a system resource required to carry out the requested operation. This will only be used when the resource is not covered by the other ETH_ERR_RES constants.See also ETH_ERR_RES_INT, ETH_ERR_RES_MEM and ETH_ERR_RES_IO.

ETH_ERR_RES_MEM

ETH_ERR_RES_INT is used to indicate that a required interrupt number priority are currently unavailable for use by the HAL. This error will be encountered either when an attempt is made to change an interrupt number priority during run time or when ETH_initialise_dev is called. If interrupt numbers/priorities cannot be dynamically changed due to hardware limitations then ETH_ERR_NOT_SUPPORTED_HW will be returned.

ETH_ERR_RES_IO

ETH_ERR_RES_MEM is used to indicate that the HAL was unable to allocate enough memory to complete the requested operation.

ETH_ERR_NOT_SUPPORTED

ETH_ERR_RES_IO is used to indicate that one or more physical connection lines are unavailable. This may be because a line is shared with another peripheral (and has been allocated) or if it is currently in use as a general purpose I/O line.

ETH_ERR_NOT_SUPPORTED_HW

ETH_ERR_NOT_SUPPORTED is used to indicate that a requested operation cannot be performed because it is not supported in software. This may be because a required software module has been compiled out (see configuring the Ethernet HAL).

ETH_ERR_UNKNOWN_DEV

ETH_ERR_NOT_SUPPORTED_HW is used to indicate that a requested operation cannot be performed because a required feature is not supported in hardware.

ETH_ERR_NOT_INITIALISED

ETH_ERR_UNKNOWN_DEV indicates that a device ID passed to an API function was not valid. **ETH_ERR_NOT_INITIALISED is returned if an API function is called before the HAL has been successfully initialized. This checking may be configured out to improve runtime performance, see Configuring the Ethernet HAL for information**

Comments

The ETH_STATUS enum constants will be used to return the status from the functions.If the function execute successfully then it will return the ETH_SUCCESS or it will return the specific error code. This return code will be useful for the application to learn about failure.

2.6 ETH_CTRL_CODE

enum ETH_CTRL_CODE {

ETH_CTRL_MAC_TX,

ETH_CTRL_MAC_RX,

ETH_CTRL_PHY,

ETH_CTRL_CAM,

ETH_CTRL_TB,

ETH_CTRL_DMUR,

ETH_CTRL_DMUT,

This enumeration constant is used with ETH_control_dev API function. ETH_CTRL_DMUT may be used to provide basic control to reset DMUT configuration parameters.

};

Ethernet Control Enumeration

Defined in: eth_api.h

Members

ETH_CTRL_MAC_TX

This enumeration constant is used with ETH_control_dev API function. ETH_CTRL_MAC may be used to provide basic control to modify MAC parameters, other than MAC transmitter and receiver

ETH_CTRL_MAC_RX

This enumeration constant is used with ETH_control_dev API function. ETH_CTRL_MAC_TX may be used to provide basic control to modify MAC transmitter parameters

ETH_CTRL_PHY

This enumeration constant is used with ETH_control_dev API function.

ETH_CTRL_MAC_RX may be used to provide basic control to modify MAC receiver parameters

ETH_CTRL_CAM

This enumeration constant is used with ETH_control_dev API function. ETH_CTRL_PHY may be used to provide basic control to modify Physical device parameters

ETH_CTRL_TB

This enumeration constant is used with ETH_control_dev API function. ETH_CTRL_CAM may be used to provide basic control to modify CAM parameters

ETH_CTRL_DMUR

This enumeration constant is used with ETH_control_dev API function. ETH_CTRL_TB may be used to provide basic control to reset transmit buffer configuration parameters

ETH_CTRL_DMUT

This enumeration constant is used with ETH_control_dev API function. ETH_CTRL_DMUR may be used to provide basic control to reset DMUR configuration parameters

This enumeration constant is used with ETH_control_dev API function. ETH_CTRL_DMUT may be used to provide basic control to reset DMUT configuration parameters.

Comments

The ETH_CTRL_CODE enum will be used in ETH_control_dev (IOCTL) function. The enum specifies which particular unit or functional entity would be changed at run time.

The user application will pass this enum as an argument to ETH_control_dev function, the ETH_control_dev() will then branch to the appropriate function, which would change the configuration according to the data provided.

2.7 ETH_MAC_CONF Structure

```
typedef struct {
IFX_UINT8 full_duplex:1;
IFX_UINT8 loop_back:1;
} ETH_MAC_CONF;
```

MAC Configuration Info

Defined in: eth_api.h

Members

full_duplex:1

Ethernet controller is in full duplex mode

loop_back:1

Ethernet controller works in loop back mode

Comments

The ETH_MAC_CONF structure is used to program the MAC controller configuration. This structure is a part of ETH_COM_PARAMS, which has the startup configuration of Ethernet driver.

This structure is also used to change the run time MAC controller configuration through ETH_ctrl_mac function.

2.8 ETH_MAC_TX_CONF Structure

```
typedef struct {
IFX_UINT8 no_crc:1;
IFX_UINT8 no_defer:1;
IFX_UINT8 sqe_check:1;
IFX_UINT8 send_pause:1;
IFX_UINT8 tx_enable:1;
 ETH_MAC_TX_CONF;
} ETH_MAC_TX_CONF;
```

MAC Transmission Configuration Info

Defined in: eth_api.h

Members

no_crc:1

No pad added to the out going packet if packet length less than 64 bytes

no_defer:1

No CRC added to the out going packet

sqe_check:1

Defer will not be checked to the out going frame

send_pause:1

Signal quality error checking in MII 10 mbps mode

tx_enable:1

for sdpause bit enable

ETH_MAC_TX_CONF

for txEn

Comments

The ETH_MAC_TX_CONF structure is used to program the MAC controller transmission configurations. This structure is a part of ETH_COM_PARAMS, which has the startup configuration of Ethernet driver.

This structure is also used to change the run time MAC TX configuration through ETH_ctrl_mac_tx function.

2.9 ETH_MAC_RX_CONF Structure

```
typedef struct {
IFX_UINT8 accept_short:1;
IFX_UINT8 strip_crc:1;
IFX_UINT8 pass_ctrl_pkts:1;
IFX_UINT8 ignore_crc:1;
IFX_UINT8 rx_enable:1;
 ETH_MAC_RX_CONF;
} ETH_MAC_RX_CONF;
```

MAC Receive Configuration Info

Defined in: eth_api.h

Members

accept_short:1

Accept long frames longer than 1518 bytes

strip_crc:1

Accept short frames of size less than 64 bytes

pass_ctrl_pkts:1

Strip CRC from received frame and no CRC checking at DMUR

ignore_crc:1

Pass the MAC control (pause) frames to the logical link layer (LLC)

rx_enable:1

Do not check CRC of a receiving frame

ETH_MAC_RX_CONF
for rxEn

Comments

The ETH_MAC_RX_CONF structure is used to program the receive MAC controller configurations. This structure is a part of ETH_COM_PARAMS, which has the startup configuration of Ethernet driver.

This structure is also used to change the run time MAC RX configuration settings through ETH_ctrl_mac_rx function.

2.10 ETH_PHY_CONF Structure

```
typedef struct {
IFX_UINT8 loopback:1;
IFX_UINT8 full_duplex:1;
IFX_UINT8 auto_negotiate:1;
 ETH_PHY_CONF;
```

} ETH_PHY_CONF;

Physical Device programming Info

Defined in: eth_api.h

Members

loopback:1

Speed 0-->10 Mbps, 1-->100 Mbps

full_duplex:1

Ethernet controller operates in PHY device loop back mode

auto_negotiate:1

PHY device operates in full duplex mode

ETH_PHY_CONF

Enable auto negotiation feature of PHY device

Comments

The ETH_PHY_CONF structure is used to program the PHY device control register configurations.

This structure is a part of ETH_COM_PARAMS, which has the startup configuration of Ethernet driver.

2.11 ETH_PHY_CTRL Structure

typedef struct {

IFX_UINT16 phy_num:5;

IFX_UINT16 reg_num:5;

IFX_UINT32 * data;
 ETH_PHY_CTRL;
} ETH_PHY_CTRL;

Physical Device Control Info

Defined in: eth_api.h

Members

phy_num:5

To read from or write to PHY device registers 0-->read, 1-->write

reg_num:5

PHY device number

data

Register number inside PHY device

ETH_PHY_CTRL

Data want to read/write from/to the PHY register

Comments

This PHY device configuration control structure is used to configure the PHY device using the ETH_PHY_CTRL structure the user will be able to read /write any data from/ into any of PHY device registers The structure ETH_PHY_CTRL is used to change the PHY device configuration settings at run time through ETH_ctrl_mac_phy function.

2.12 ETH_CAM_DATA Structure

typedef struct {

IFX_UINT8 no_of_mac_addrs;

IFX_UINT32 * mac_addrs;

IFX_UINT32 addr_loc_enable;

 ETH_CAM_DATA;

} ETH_CAM_DATA;

CAM programming info

Defined in: eth_api.h

Members

no_of_mac_addrs

Starting location of MAC address ranges from 0x0 to 0x13

mac_addrs

The number of MAC addresses to be programmed into CAM

addr_loc_enable

Pointer to the MAC addresses which are to be programmed into CAM

ETH_CAM_DATA

The addresses are to be enabled to filter the received frames

Comments

CAM can be programmed to support up to 20 (0-19) MAC addresses each of which are 6 bytes long. The MAC addresses are placed into mac_addrs field of the structure defined below. The MAC addresses are to be programmed to the CAM is first written into mac_addrs[]. Then the addresses in mac_addrs[] are programmed into the CAM.

E.g. If the MAC addresses to be programmed into CAM are 0x010203040506, 0xAABBCCDDEEFF, then the contents of the mac_addrs[] are as follows.

mac_addrs[0] = 0x01020304

mac_addrs[1] = 0x0506AABB

mac_addrs[2] = 0xCCDDEEFF

start_mac_addr = 0

no_of_mac_addrs = 2

This structure is included in ETH_CAM_CTRL to add MAC addresses to CAM and enable the particular MAC addresses in CAM at run time.

2.13 ETH_CAM_COMMAND

```
enum ETH_CAM_COMMAND {
ETH_ADD_MAC_ADDR,
ETH_ENABLE_LOC,
ETH_RESET_CONF
};
```

Ethernet CAM Command

Defined in: eth_api.h

Members

ETH_ADD_MAC_ADDR

This is used to add new MAC address to CAM

ETH_ENABLE_LOC

This is used to enable MAC addresses in CAM which are programmed

ETH_RESET_CONF

This is used to reset MAC CAM configuration

Comments

The ETH_CAM_COMMAND enum is used in ETH_CAM_CTRL structure to distinguish the provided data and branch to the specific part of code in ETH_ctrl_mac_cam function.

2.14 ETH_CAM_CONF Structure

```
typedef struct {
IFX_UINT8 uni_cast:1;
IFX_UINT8 multi_cast:1;
IFX_UINT8 broad_cast:1;
IFX_UINT8 negative_cam:1;
IFX_UINT8 cam_compare_enable:1;
} ETH_CAM_CONF;
```

CAM Configuration Info

Defined in: eth_api.h

Members

uni_cast:1

Accept unicast address packets

multi_cast:1

Accept Multicast address packets

broad_cast:1

Accept Broadcast address packets

negative_cam:1

Receive the frames which are not recognized by CAM

cam_compare_enable:1

Enable CAM controller to filter the received frames

Comments

The ETH_CAM_CONF structure is used to program the CAM controller configurations. This structure is a part of ETH_COM_PARAMS, which has the startup configuration of Ethernet driver.

This structure is also a part of ETH_CAM_CTRL to change the CAM configuration settings at run time through ETH_ctrl_mac_cam function.

2.15 ETH_CAM_UDATACONF Union

```
typedef union {
ETH_CAM_DATA mac_addr_data;
ETH_CAM_CONF conf_params;
} ETH_CAM_UDATACONF;
```

A union contains CAM programming info. This union is used in the structure ETH_CAM_CTRL

Defined in: eth_api.h

Members

mac_addr_data

Write MAC addresses into CAM and enable the programmed MAC addresses

conf_params

To change CAM configuration parameters

2.16 ETH_CAM_CTRL Structure

```
typedef struct {
ETH_CAM_COMMAND command;
ETH_CAM_UCONFDATA data;
} ETH_CAM_CTRL;
```

CAM Control Info

Defined in: eth_api.h

Members

command

Contains the CAM command information

data

Union contains CAM programming info

Comments

The structure ETH_CAM_CTRL is used to add the MAC addresses to CAM, enable the MAC addresses in CAM or to change the CAM configuration parameters at runtime through ETH_ctrl_mac_cam function

2.17 ETH_CAM_PROG Structure

```
typedef struct {
```

```
IFX_UINT8 ETH_cam_addr_hw;
IFX_UINT8 ETH_cam_addr_user;
IFX_UINT8 ETH_user_array_position;
IFX_UINT8 ETH_num_addr;
ETH_CAM_CTRL * ETH_ctrl_data;
} ETH_CAM_PROG;
```

CAM Program Info

Defined in: eth_api.h

Members

ETH_cam_addr_hw

Index of the MAC address in the CAM

ETH_cam_addr_user

Index of the MAC address in the user array

ETH_user_array_position

Flag indicating the start of the MAC address in the user array

ETH_num_addr

Number of addresses to be programmed

ETH_ctrl_data

Pointer to the ETH_CAM_CTRL structure, containing the user settings

Comments

The structure ETH_CAM_PROG is used to add the MAC addresses to CAM

2.18 ETH_RB_CONF Structure

```
typedef struct {
IFX_UINT16 fwd_threshold:2;
IFX_UINT16 actq_threshold:6;
IFX_UINT16 free_pool_threshold:8;
} ETH_RB_CONF;
```

Receive Buffer Configuration

Defined in: eth_api.h

Members

fwd_threshold:2

Receive buffer forward threshold code

actq_threshold:6

Receive buffer action queue threshold

free_pool_threshold:8

Receive buffer free pool threshold

Comments

The ETH_RB_CONF structure is used to program the Receive Buffer configuration values. This structure is a part of ETH_COM_PARAMS, which has the startup configuration of Ethernet driver.

This structure is also used to change the Receive Buffer configuration vals at runtime through ETH_ctrl_rb function.

2.19 ETH_TB_CONF Structure

```
typedef struct {
IFX_UINT16 ind_tx_buff_size;
IFX_UINT16 refill_threshold:3;
IFX_UINT16 fwd_threshold:3;
} ETH_TB_CONF;
```

Transmit Buffer Configuration

Defined in: eth_api.h

Members

ind_tx_buff_size

Individual transmitter buffer size

refill_threshold:3

Transmit buffer refill threshold value

fwd_threshold:3

Transmit forward threshold value

Comments

The ETH_TB_CONF structure is used to program the Transmit Buffer configuration values. This structure is a part of ETH_COM_PARAMS, which has the startup configuration of Ethernet driver.

This structure is also used to change the Transmit Buffer configuration values at runtime through ETH_ctrl_tb function.

2.20 ETH_RXDESC Structure

```
typedef struct {
```

```
IFX_UINT32 desc_id:6;
IFX_UINT32 reserved:4;
IFX_UINT32 rhi:1;
IFX_UINT32 hold:1;
IFX_UINT32 reserved1:1;
IFX_UINT8        * data;
IFX_UINT32 bno:16;
IFX_UINT32 rab:1;
IFX_UINT32 il_len:1;
IFX_UINT32 crc:1;
IFX_UINT32 rx_frm_ovflow:1;
IFX_UINT32 max_frm_len:1;
IFX_UINT32 reserved2:9;
IFX_UINT32 frm_end:1;
 ETH_RXDESC;
} ETH_RXDESC;
```

Receive Descriptor

Defined in: eth_api.h

Members

desc_id:6

Maximum data it can hold

reserved:4

Descriptor id

rhi:1

Extra header data inside data buffer

hold:1

Receive hold indication bit

reserved1:1

Hold bit, set to 1 for last descriptor

data

Pointer to next descriptor

bno:16

Pointer to data buffer

rab:1

Data buffer size

il_len:1

DMUR abort

crc:1

Invalid length

rx_frm_ovflow:1

CRC error

max_frm_len:1

Receive frame over flow

reserved2:9

Packet exceed the max frame length

frm_end:1

DMUR completely filled the data buffer associated with current descriptor

ETH_RXDESC

Full frame received

Comments

The ETH_RXDESC structure is used to define the DMUR descriptors. These descriptors are maintained by DMUR. Each descriptor associated with one data buffer. The data received from PHY medium placed into these descriptors by DMUR. Either the data buffer is filled up or a single frame is received the DMUR branches to next descriptor in descriptor list provided hold bit is not set.

2.21 ETH_DMUR_CONF Structure

```
typedef struct {
IFX_UINT8 endian_mode:1;
IFX_UINT8 alignment:1;
} ETH_DMUR_CONF;
```

DMUR Configuration Register

Defined in: eth_api.h

Members

endian_mode:1

Endian mode 0-->little endian, 1-->big endian

alignment:1

Block boundary alignment for burst is enabled

Comments

The ETH_DMUR_CONF structure is used to program the Data Management Unit Receive configuration values. This structure is a part of ETH_COM_PARAMS, which has the startup configuration of Ethernet driver.

This structure is also used to change the Data Management Unit Receive configuration values at runtime through ETH_ctrl_dmur function.

2.22 ETH_DMUT_CONF Structure

```
typedef struct {
IFX_UINT8 endian_mode;
} ETH_DMUT_CONF;
```

DMUT Configuration Register

Defined in: eth_api.h

Members

endian_mode

Endian mode 0->little endian, 1->big endian

Comments

The ETH_DMUT_CONF structure is used to program the Data Management Unit Transmit configuration values. This structure is a part of ETH_COM_PARAMS, which has the startup configuration of Ethernet driver.

This structure is also used to change the Data Management Unit Transmit configuration values at runtime through ETH_ctrl_dmut function.

2.23 ETH_TXDESC Structure

```
typedef struct {
IFX_UINT32 no:16;
IFX_UINT32 desc_id:6;
IFX_UINT32 cmp_en:1;
IFX_UINT32 thi:1;
IFX_UINT32 hold:1;
IFX_UINT32 frm_end:1;
struct ETH_txdesc  * next;
IFX_UINT8         * data;
IFX_UINT32 cmp:1;
void              * os_specific;
IFX_UINT32 dummy;
IFX_UINT32 dummy_1;
IFX_UINT32 dummy_2;
} ETH_TXDESC;
```

Transmit Descriptor

Defined in: eth_api.h

Members

no:16

Max data size of data buffer

desc_id:6

Descriptor id

cmp_en:1

Complete interrupt is enabled

thi:1

TX host indication bit

hold:1

Hold bit, last descriptor set this bit to 1

frm_end:1

Complete frame transmitted

next

Pointer to next descriptor

data

Pointer to data buffer

cmp:1

Complete

os_specific

A cookie field for any OS driver

dummy

required for alignment reasons

dummy_1

required for alignment reasons

dummy_2

required for alignment reasons

Comments

The ETH_TXDESC structure is used to define the DMUT descriptors. The data coming from applications will be buffered into data buffer and associated with these descriptors.

The filled descriptors are added to the DMUT, the DMUT will transfer the data associated with descriptor to the TB (towards PHY medium). Once the data is transferred the DMUT will branch to the next descriptor in list.

2.24 ETH_COM_PARAMS Structure

```
typedef struct {
ETH_MAC_CONF mac_conf;
ETH_CAM_CONF cam_conf;
ETH_MAC_TX_CONF tx_conf;
ETH_MAC_RX_CONF rx_conf;
ETH_PHY_CONF phy_conf;
ETH_CAM_DATA cam_data;
ETH_DMUR_CONF dmur_conf;
ETH_DMUT_CONF dmut_conf;
ETH_RB_CONF rb_conf;
ETH_TB_CONF tb_conf;
```

} ETH_COM_PARAMS;

Ethernet Configuration Setting

Defined in: eth_api.h

Members

mac_conf

MAC configuration data

cam_conf

CAM configuration data

tx_conf

MAC TX configuration data

rx_conf

MAC RX configuration data

phy_conf

PHY device configuration data

cam_data

MAC CAM configuration data

dmur_conf

DMUR configuration data

dmut_conf

DMUT configuration data

rb_conf

Receive buffer configuration data

tb_conf

Transmit buffer configuration data

Comments

The ETH_COM_PARMS structure is used to specify complete configuration settings for the Ethernet driver. The elements of this structure contains the configuration information for the individual units of the Ethernet controller This structure is used with the ETH_initialise_dev() API function during the initialization of the device driver

2.25 ETH_RETURN_NUM Structure

```
typedef struct {
IFX_UINT32 ETH_fe:1;
IFX_UINT32 ETH_rab:1;
IFX_UINT32 ETH_ilen:1;
IFX_UINT32 ETH_crc:1;
IFX_UINT32 ETH_rfod:1;
IFX_UINT32 ETH_mfl:1;
} ETH_RETURN_NUM;
```

Bit Fields for the ETH_return_num field in the ETH_transfer structure.

Defined in: eth_api.h

Members

ETH_fe:1

End of frame reached

ETH_rab:1

Receive Abort bit set

ETH_ilen:1

Illegal Length of incoming data packets

ETH_crc:1

CRC Error

ETH_rfod:1

Receive Frame Overflow

ETH_mfl:1

Inconimg data packet length exceeded the maximum allowed frame length

Comments

These bit fields are used to indicate the status of the data received to the application. Depending on the size of the receive buffer, an ethernet frame may be spread accross more than one descriptor. The 'fe' bit in the ETH_return_num field in the ETH_transfer structure indicates to the application that the end of a frame has reached.

2.26 ETH_TRANSFER Structure

typedef struct {

```
IFX_UINT8     * ETH_buffer;
IFX_UINT32 ETH_buffer_size;
ETH_RETURN_NUM ETH_return_num;
} ETH_TRANSFER;
```

Ethernet Data Transfer Structure

Defined in: eth_api.h

Members

ETH_buffer

Pointer to data buffer

ETH_buffer_size

Data size

ETH_return_num

Bit fields (described above) indicating the status of the receive operation.

Comments

This structure is used to exchange data transfer between driver and application.As soon as the destination receives the data it will return the status.

2.27 ETH_STAT Structure

```
typedef struct {
IFX_UINT32 rsrc_err;
IFX_UINT32 tx_def;
IFX_UINT32 tx_lst_car;
IFX_UINT32 tx_good_frms;
IFX_UINT32 tx_under;
IFX_UINT32 tx_excs_coll;
IFX_UINT32 tx_late_coll;
IFX_UINT32 tx_pause;
IFX_UINT32 rx_pause;
IFX_UINT32 rx_align_err;
IFX_UINT32 rx_crc_err;
IFX_UINT32 rx_lng_frm;
IFX_UINT32 rx_over;
IFX_UINT32 rx_good_frms;
} ETH_STAT;
```

Ethernet statistics Structure

Defined in: eth_api.h

Members

rsrc_err

The received frames are discarded because of data buffers are not available both at DMUR and DMUT

tx_def

Transmission is deferred for out going frame

tx_lst_car

Carrier is lost

tx_good_frms

Frames transmitted without any errors

tx_under

MAC TX FIFO under run

tx_excs_coll

Excessive collision occurred while transmitting frames

tx_late_coll

Late collision occurred while transmitting frames

tx_pause

Ethernet controller send control frames to the sources of frames

rx_pause

Received control frames

rx_align_err

Received the frames with alignment error

rx_crc_err

Received frames with CRC error

rx_lng_frm

Received long frames size more than 1518 bytes, when received long frames option is disabled

rx_over

Receiver is overflowed

rx_good_frms

Good frames received

Comments

ETH_STAT is used to maintain the statistics of the received and transmitted frames. Once any counter reaches maximum value it will automatically reset to zero. So the application can keep track of these counters from time to time.

2.28 ETH_STAT_INF Structure

```
typedef struct {
ETH_COM_PARAMS ETH_conf_params;
ETH_STAT ETH_stat_cntrs;
} ETH_STAT_INF;
```

Ethernet driver status info, this struct contains the current configuration values of all the configuration registers, required. It also contains the current values of all the interrupts. This structure is derived from some of the structures defined above.

Defined in: eth_api.h

Members

ETH_conf_params

Driver current configuration parameters

ETH_stat_cntrs

Received and transmit frames status counters

2.29 ETH_gpio_alloc

IFX_UINT8 ETH_gpio_alloc(*void***)**

ETH_gpio_alloc

Defined in: eth_api.h

Return Value

Success/Failure

1

The requested ports are allocated.

0

The requested ports are not allocated.

Parameters

void

None

Implementation Details

Get the ports and reserve the port bits for the Ethernet controller from System HAL, by calling the system HAL provided API.

2.30 ETH_gpio_free

IFX_UINT8 ETH_gpio_free(*void***)**

ETH_gpio_free

Defined in: eth_api.h

Return Value

Success/Failure

1

The ports are being freed and returned to System HAL

0

The ports are not being freed and not returned to System HAL

Parameters

void

None

Implementation Details

- Release and return the ports to the System HAL.

3 Ethernet API Functions

3.1 ETH_initialise_dev

ETH_STATUS ETH_initialise_dev(ETH_DEVICE *ETH_device*, **ETH_COM_PARAMS * ** *ETH_setup***)**

Ethernet Driver Initialization function, this function initializes the internal data structures of the HAL related to the device selected by ETH_DEVICE, allocates any required system resources and configures the peripheral according to the ETH_COM_PARAMS structure. The ETH_COM_PARAMS structure must be initialized by the user before calling ETH_initialise_dev. This function must be called successfully before any of the other API functions are used and if ETH_terminate_dev is called then ETH_initialise_dev must be called again before using the other API functions.

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Initialization is success.

ETH_ERR_RES_INT

Requested priorities are not available

ETH_ERR_RES_IO

Requested ports are not available

ETH_ERR

Functional blocks are not initialized.

Parameters

ETH_device

Ethernet controller hardware module identification value

ETH_setup

Configuration parameters, to be programmed to the Ethernet controller kernel registers. These parameters are ignored.

Implementation Details

- Copy the driver initialization configuration parameters to ETH_COM_PARAMS datastructure.
- Get the interrupt priorities from System HAL, which are defined in SYS_CFG.H file
- Reserve the required port bits for driver defined in GPIO_CFG.H file.
- Check the reserved interrupt priorities and port bits are not clashing with any other module.
- Enable the global interrupt through System HAL.
- Call the functional blocks(DMUT, DMUR, TB, RB and MAC) initialization routines in a specific sequence in order not to lose any frames receiving either from PHY medium or from application.

3.2 ETH_initialise_mac

ETH_STATUS ETH_initialise_mac(ETH_COM_PARAMS * ***ETH_setup*****, Configuration parameters, to be programmed to the MAC kernel registers)**

Ethernet MAC controller initialization function

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

MAC initialization is success

Parameters

ETH_setup

Ethernet controller hardware module identification value

Implementation Details

- MAC initialization routine should not be exposed to application and it will called by only ETH_initialise_dev routine.
- Write the application provided MAC addresses into CAM.
- If ETH_CFG_USE_PAUSE feature is enabled program the pause related addresses and data into CAM.
- Configure the PHY device and program the interrupt mask registers.
- If ETH_CFG_STAT_LOG feature is enabled, the following interrupts will be enabled in their corresponding interrupt mask registers.
- MAC TX - late collision, excessive collision, under run, lost carrier, remote pause and frame sent without errors.
- MAC RX - Alignment error, CRC error, over flow error, control frame received, long frame error and good frame received.
- Else only the MAC TX control frame sent interrupt will be enabled.
- Program the MAC TX and RX control blocks and enables both receiver and transmitter.

3.3 ETH_initialise_phy

ETH_STATUS ETH_initialise_phy(ETH_DEVICE *ETH_device*, **ETH_PHY_CONF *** *ETH_phy_conf***)**

Ethernet PHY initialisation function

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

PHY initialisation is successful

Parameters

ETH_device

Ethernet controller hardware module identification value

ETH_phy_conf

Configuration parameters, to be programmed in the PHY

Implementation Details

- PHY initialisation routine should not be exposed to application and it will called by only ETH_initialise_mac routine.
- Check if a valid PHY is present
- Find the address of the PHY
- Reset the PHY
- Configure the initialisation parameters in the PHY (full duplex, speed 10/ 100), loopback.

3.4 ETH_check_link

ETH_STATUS ETH_check_link(ETH_DEVICE *ETH_device*)

Ethernet check link function, this function reads the PHY status register to detrmine if the link is up or not. The application can call this function before transmitting or receiving data to determine if the link is up or not.

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Link is up

ETH_ERR

Link is down

Parameters

ETH_device

Ethernet controller hardware module identification value

Implementation Details

- Read the PHY status register
- Check if link is up, if yes retun ETH_SUCCESS else return ETH_ERR

ETH_phy_autoneg

ETH_STATUS ETH_phy_autoneg(void)

Ethernet PHY autoneg

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

PHY detection is successful

Implementation Details

- PHY autoneg routine should not be exposed to application and it will called by only ETH_initialise_phy routine.
- Program the PHY for Autonegotiating with the link partner.
- Set the start autoneg

- First try to find whether the PHY is present in the default address
- If found return success
- If not found, loop through all the addresses & store the address of the PHY

3.5 ETH_read_phy

void ETH_read_phy(ETH_DEVICE *ETH_device*, **ETH_PHY_CTRL *** *ETH_phy_ctrl*, **IFX_UINT16 *** *ETH_mdio_data***)**

Ethernet read PHY function

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

PHY detection is successful

Parameters

ETH_device

Ethernet controller hardware module identification value

ETH_phy_ctrl

Pointer to structure containing the PHY address & register address

ETH_mdio_data

Pointer to data which was read from the PHY register

Implementation Details

- PHY initialisation routine should not be exposed to application and it will called by only ETH_detect_phy routine.
- Program the station management control register with the phy address, register address, command (read) & set the busy flag
- Read the values of the station management data register

3.6 ETH_camctrl_caseAodd

void ETH_camctrl_caseAodd(ETH_DEVICE *ETH_device*, **ETH_CAM_PROG *** *ETH_cam_prog***)**

This function is used for CAM run time address programming

Defined in: eth_api.h

Return Value

None

Parameters

ETH_device

Ethernet controller hardware module identification value

ETH_cam_prog

CAM programming parameters

Implementation Details

- This initialization function not exposed to application, it will be called by ETH_cam_ctrl() function at run time when the user wants to prog the CAM addresses
- Addresses are divided into 3 cases A, B & C for ease of programming. case A addresses are the 1st, 4th, 7th, 10th.. as programmed by the user they take the shape

- This function programs a single MAC address in the CAM.
- The user_array_position field of the input parameter defines the position of the start of MAC address in the user array.

3.7 ETH_camctrl_caseAeven

void ETH_camctrl_caseAeven(ETH_DEVICE *ETH_device*, **ETH_CAM_PROG *** *ETH_cam_prog***)**

This function is used for CAM run time address programming

Defined in: eth_api.h

Return Value

None

Parameters

ETH_device

Ethernet controller hardware module identification value

ETH_cam_prog

CAM programming parameters

Implementation Details

- This initialization function not exposed to application, it will be called by ETH_cam_ctrl() function at run time when the user wants to prog the CAM addresses
- This function programs sets of 2 MAC address in the CAM.
- The addresses programmed by this function take the form _______________
- The user_array_position field of the input parameter defines the position of the start of MAC address in the user array.

3.8 ETH_camctrl_caseB

void ETH_camctrl_caseB(ETH_DEVICE *ETH_device*, **ETH_CAM_PROG *** *ETH_cam_prog*)

This function is used for CAM run time address programming

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Receive Buffer initialization is success

Parameters

ETH_device

Ethernet controller hardware module identification value

ETH_cam_prog

CAM programming parameters

Implementation Details

- This initialization function not exposed to application, it will be called by ETH_cam_ctrl() function at run time when the user wants to prog the CAM addresses
- This is the case when the CAM addr have the position ________ ______
- The user_array_position field of the input parameter defines the position of the start of MAC address in the user array.

3.9 ETH_initialise_rb

ETH_STATUS ETH_initialise_rb(ETH_DEVICE *ETH_device*, **ETH_COM_PARAMS** * *ETH_setup*)

Receive Buffer Initialization function

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Receive Buffer initialization is success

Parameters

ETH_device

Ethernet controller hardware module identification value

ETH_setup

Configuration parameters, to be programmed to the RB kernel registers

Implementation Details

- This initialization function not exposed to application, it will be called by ETH_initialise_dev()function at the time of driver initialization.
- Program the threshold codes.
- Enable the Receive Buffer interrupts.
-
- Initialize the RB by issuing an init command.

3.10 ETH_initialise_tb

ETH_STATUS ETH_initialise_tb(ETH_DEVICE *ETH_device*, ETH_COM_PARAMS * *ETH_setup*)

Transmit Buffer initialization function

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Transmit Buffer initialization is success

Parameters

ETH_device

Ethernet controller hardware module identification value

ETH_setup

Configuration parameters, to be programmed to the TB kernel registers

Implementation Details

- This initialization function not exposed to application, it will be called by ETH_initialise_dev() function at init time
- Program the forward, refill threshold codes and individual transmit buffer size.
- Enable the Transmit Buffer interrupts.
- Initialize the TB by issuing an init command.

3.11 ETH_initialise_dmur

ETH_STATUS ETH_initialise_dmur(ETH_DEVICE *ETH_device*, **ETH_COM_PARAMS** * *ETH_setup*)

Data Management Unit Receive initialization function

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

DMUR initialization is success

Parameters

ETH_device

Ethernet controller hardware module identification value

ETH_setup

Configuration parameters, to be programmed to the DR kernel registers

Implementation Details

- This initialization function not exposed to application, it will be called by ETH_initialise_dev() function at init time
- Create and program the DMUR descriptors as a linked list and update all fields. Each DMUR descriptor will be associated with one data buffer.
- ETH_CFG_DMUR_DESC_NUM and ETH_CFG_DMUR_DBUFF_SIZE are defined in ETH_CFG.H file specifies the number of DMUR descriptors and each data buffer size associated with descriptor respectively.
- Program the configuration registers. Command complete, silent discard and frame end interrupts will be enabled in the interrupt mask register.

- Enable the DMUR interrupt.
- Initialize the DMUR by issuing an init command.

3.12 ETH_initialise_dmut

ETH_STATUS ETH_initialise_dmut(ETH_DEVICE *ETH_device*, **ETH_COM_PARAMS *** *ETH_setup*)

Data Management Unit Transmit Initialization function

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

DMUT initialization is success

Parameters

ETH_device

Ethernet controller hardware module identification value

ETH_setup

Configuration parameters, to be programmed to the kernel registers

Implementation Details

- This initialization function not exposed to application, it will be called by ETH_initialise_dev() function at the time of driver initialization.
- Create and program the DMUT descriptors as a linked list and update all fields.
- The DMUT descriptors are not associated with data buffers each descriptor will set the HI bit.

- The define ETH_CFG_DMUT_DESC_NUM defined in ETH_CFG.H file specifies the number of DMUT descriptors.
- Program the configuration registers. Command complete, transmission abort and hold transmission abort interrupts are enabled in the interrupt mask register. Host Indication bit also set in each descriptor.
- Enable the DMUT interrupt.
- Initialize the DMUT by issuing an init command.

3.13 ETH_terminate_dev

ETH_STATUS ETH_terminate_dev(ETH_DEVICE *ETH_device***)**

Ethernet Driver Terminate function, this function sets the peripheral, selected by the ETH_device parameter, into a disabled state and frees any system resources previously allocated in ETH_initialise_dev. After this function has been called ETH_initialise_dev must be called successfully before any of the other API functions are used.

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Termination of driver is success

ETH_ERR

Termination of driver is failure.

Parameters

ETH_device

Ethernet controller hardware module identification value

Implementation Details

- API will be exposed to application.
- Release the interrupt priorities and return to System HAL.
- Free the ports and return to System HAL.
- Call the each functional block(DMUR, DMUT, RB, TB and MAC) terminate routines.

3.14 ETH_terminate_mac

ETH_STATUS ETH_terminate_mac(ETH_DEVICE *ETH_device*)

Ethernet MAC controller terminate function

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Termination of MAC is success

Parameters

ETH_device

Ethernet controller hardware module identification value

Implementation Details

- This function is not exposed to application and called by ETH_terminate_dev at the time of driver termination.
- Disable the MAC interrupts.
- Disable the MAC receiver and transmitter.
- Isolate PHY device.

3.15 ETH_terminate_rb

ETH_STATUS ETH_terminate_rb(ETH_DEVICE *ETH_device*)

Receive Buffer terminate function

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Termination of Receive Buffer is success

Parameters

ETH_device

Ethernet controller hardware module identification value

Implementation Details

- This function is not exposed to application and called by ETH_terminate_dev at the time of driver termination.
- Disables the free pool monitor and action queue count interrupts.

3.16 ETH_terminate_tb

ETH_STATUS ETH_terminate_tb(ETH_DEVICE *ETH_device*)

Transmit Buffer terminate function

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Termination of Transmit Buffer is success

Parameters

ETH_device

Ethernet controller hardware module identification value

Implementation Details

- This function is not exposed to application and called by ETH_terminate_dev at the time of driver termination.
- Disable the Transmit Buffer interrupt.
- Stop the TB by issuing an off command.

3.17 ETH_terminate_dmur

ETH_STATUS ETH_terminate_dmur(ETH_DEVICE *ETH_device*)

Data Management Unit Receive terminate function

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Termination of DMUR is success

Parameters

ETH_device

Ethernet controller hardware module identification value

Implementation Details

- This function is not exposed to application and called by ETH_terminate_dev at the time of driver termination.
- Disables the interrupts associated with DMUR.
- Stop the DMUR by issuing an off command.

3.18 ETH_terminate_dmut

ETH_STATUS ETH_terminate_dmut(ETH_DEVICE *ETH_device*)

Data Management Unit Transmit terminate function

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Termination of DMUT is success

Parameters

ETH_device

Ethernet controller hardware module identification value

Implementation Details

- This function is not exposed to application and called by ETH_terminate_dev at the time of driver termination.
- Disables the interrupts associated with DMUT.
- Free the data buffers associated with DMUT and waiting for transmission.
- Stop the DMUT by issuing an off command.

3.19 ETH_abort

ETH_STATUS ETH_abort(ETH_DEVICE *ETH_device*)

Ethernet driver abort function, this function cancels all currently queued data transfers and stops any transfers currently being processed on the peripheral module selected by ETH_device. ETH_initialise_dev need not be called after this function before the other API functions can be used, this function merely clears all current and pending transfers it does not terminate the HAL. New transfers may be requested using ETH_read and/or ETH_write immediately after this function returns. This function may be used to clear all requests before changing modes

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Termination of DMUT is success

Parameters

ETH_device

Ethernet controller hardware module identification number

Implementation Details

- Stop the MAC TX and RX control blocks.
- Set a flag to indicate the application that device is busy.
- Issue an off command to the DMUT, DMUR and TB.
- Free the pending TX request data buffers (memory) associated with DMUT.
- Rearrange all descriptors and data buffers associated with DMUR and DMUT.
- Enable all the functional blocks of Ethernet controller to make the driver ready to receive and transmit frames.

3.20 ETH_status_dev

ETH_STATUS ETH_status_dev(ETH_DEVICE *ETH_device*, **ETH_STAT_INF *** *ETH_stat_inf***)**

Ethernet driver status function read and returns the current driver configuration settings. Optionally returns the driver statistics counters provided ETH_CFG_STAT_LOG feature is enabled.

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Reading and returning the status of Ethernet driver is success.

Parameters

ETH_device

Ethernet controller hardware module identification number

ETH_stat_inf

User provided structure to read present configuration parameters of a particular module

Implementation Details

- Copy all the functional unit driver configuration parameters to the application provided data structure.
- Copy the statistics counters optionally to the user provided data structure.
- The statistics counters only copied when ETH_CFG_STAT_LOG feature is enabled in ETH_CFG.H file.

3.21 ETH_control_dev

ETH_STATUS ETH_control_dev(ETH_DEVICE *ETH_device*, **ETH_CTRL_CODE** *ETH_ctrl_code*, **void *** *ctrl_data***)**

Ethernet Controller runtime configuration control function, this function is used to change the existing configuration of driver during the run time. This function will call the particular control function depending upon the ETH_CTRL_CODE argument provided. During these changes the driver will not be aborted from transmission and reception of frames.

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Change of configuration settings is success

ETH_ERR

Change of configuration settings is failure

Parameters

ETH_device

Ethernet controller hardware module identification number.

ETH_ctrl_code

The functional block which configuration has to be changed.

ctrl_data

The new configuration parameters.

Implementation Details

- This function would be used to change the configuration settings of a particular block of Ethernet controller.
- Check for the ETH_ctrl_code argument to identify the application requested functional block to change the configuration parameters.
- Depending upon the ETH_ctrl_code value call the appropriate IOCTL function.
- To the called function pass the new configuration values (ctrl_data)

ETH_ctrl_mac

ETH_STATUS ETH_ctrl_mac(ETH_DEVICE *ETH_device*, **ETH_MAC_CONF *** *ctrl_data*)

MAC controller runtime configuration control function, this function is used to change the MAC configuration during the run time. The configuration includes loop back and duplex modes. This function will be invoked internally by ETH_control_dev() function.

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Change of configuration settings is success

Parameters

ETH_device

Ethernet controller hardware module identification number

ctrl_data

The new configuration data

Implementation Details

- Function will change the MAC controller configuration parameters.
- The parameters are duplex mode and loop back mode.

3.22 ETH_ctrl_cam

ETH_STATUS ETH_ctrl_cam(ETH_DEVICE *ETH_device*, **ETH_CAM_CTRL *** *ctrl_data***)**

CAM controller runtime configuration control function, this function will be called by ETH_control_dev routine for Adding the MAC addresses to CAM, enable only the specified MAC addresses to filter the receiving frames and change the configuration parameters.

The application cannot remove the MAC addresses which are programmed into CAM. Either it can disable the MAC address or over writing the particular address with all zeros.

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Change of configuration settings is success

Parameters

ETH_device

Ethernet controller hardware module identification number

ctrl_data

The new configuration data

Implementation Details

- Analyze the requested operation which is embedded in ETH_CAM_CTRL argument.
- To add/remove MAC addresses to/from CAM, writhe the application provided addresses into CAM at specified address.
- Enable only the specified MAC addresses.
- Change configuration parameters.

3.23 ETH_ctrl_mac_tx

ETH_STATUS ETH_ctrl_mac_tx(ETH_DEVICE *ETH_device*, **ETH_MAC_TX_CONF *** *ctrl_data*)

MAC TX controller runtime configuration control function, this function is used to change the attributes of the MAC Transmission like adding CRC or pad during run time. This function will be invoked internally by ETH_control_dev() function.

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Change of configuration settings is success

Parameters

ETH_device

Ethernet controller hardware module identification number

ctrl_data

The new configuration data

Implementation Details

- Change the MAC TX configuration values.
- Configuration values include adding pad to the out going frame, provided the frame size is less than 64 bytes, adding CRC, checking defer.

3.24 ETH_ctrl_mac_rx

ETH_STATUS ETH_ctrl_mac_rx(ETH_DEVICE *ETH_device*, **ETH_MAC_RX_CONF *** *ctrl_data***)**

MAC RX controller runtime configuration control function, this function is used to change the attributes of the MAC RX during run time, like accept long/short frames, ignore CRC. This function will be invoked internally by ETH_control_dev() function.

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Change of configuration settings is success

Parameters

ETH_device

Ethernet controller hardware module identification number

ctrl_data

The new configuration data

Implementation Details

- Change the MAC receiver control block configuration parameters.
- The configuration parameters include checking the CRC, stripping the CRC from received frames.
- Accept or reject short(less than 64 bytes) and long(longer than 1518 bytes) frames.

3.25 ETH_ctrl_mac_phy

ETH_STATUS ETH_ctrl_mac_phy(ETH_DEVICE *ETH_device*, **ETH_PHY_CTRL *** *ctrl_data*)

PHY device runtime configuration control function, change the physical device configuration settings like speed, duplex mode during run time. The application can able to read/program any physical device and any of the register which is inside PHY. This function will be invoked internally by ETH_control_dev() function.

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Change of configuration settings is success

Parameters

ETH_device

Ethernet controller hardware module identification number

ctrl_data

The new configuration data

Implementation Details

- The application can be read/write the data from/to any register in any of the PHY device.
- This configuration is not updated in driver configuration parameters data structure.

3.26 ETH_ctrl_tb

ETH_STATUS ETH_ctrl_tb(ETH_DEVICE *ETH_device*, **ETH_TB_CONF *** *ctrl_data***)**

Transmit Buffer runtime configuration control function, change the transmit buffer configuration parameters like threshold values during run time. This function will be invoked internally by ETH_control_dev() function.

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Change of configuration settings is success

Parameters

ETH_device

Ethernet controller hardware module identification number

ctrl_data

The new configuration data

Implementation Details

Program the forward and refill threshold codes.

3.27 ETH_ctrl_dmur

ETH_STATUS ETH_ctrl_dmur(ETH_DEVICE *ETH_device*, **ETH_DMUR_CONF *** *ctrl_data***)**

Data Management Unit Receive runtime configuration control function, changes the DMUR configuration parameters during run time. This function will be invoked internally by ETH_control_dev() function.

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Change of configuration settings is success

Parameters

ETH_device

Ethernet controller hardware module identification number

ctrl_data

The new configuration data

Implementation Details

Program the DMUR endian configuration value.

3.28 ETH_ctrl_dmut

ETH_STATUS ETH_ctrl_dmut(ETH_DEVICE *ETH_device*, **ETH_DMUT_CONF *** *ctrl_data***)**

To change the DMUT configuration, change the DMUT configuration parameters during run time. This function will be invoked internally by ETH_control_dev() function.

Defined in: eth_api.h

Return Value

Returns Ethernet Status

ETH_SUCCESS

Change of configuration settings is success

Parameters

ETH_device

Ethernet controller hardware module identification number

ctrl_data

The new configuration data

Implementation Details

Program the DMUT endian configuration value.

3.29 ETH_read

ETH_STATUS ETH_read(ETH_DEVICE *ETH_device***)**

Read data/frame received from PHY medium, this function is used by the application to register call back routine. When a complete descriptor is received, the receive ISR will call the application receive callback function and pass the pointer to the data buffer. The application is required to copy the data in its own (application's buffer). The application must not make any assumptions about the data buffer once the callback function is exited.

Defined in: eth_api.h

Return Value

Returns Ethernet status

ETH_SUCCESS

Change of configuration settings is success

Parameters

ETH_device

Ethernet controller hardware module identification number

Implementation Details

- Register the application provided call back function to the HAL. This call back function will receive the frames coming from PHY medium.
- The call back function will copy the received frame into local data buffer and release the buffer associated with frame.
- The call back function should have a specific prototype

- E.g. void app_callback(ETH_TRANSFER *rcvd_frm)

3.30 ETH_reg_tx_cb

ETH_STATUS ETH_reg_tx_cb(ETH_DEVICE *ETH_device*, **ETH_APP_TX_CB** *APP_tx_cb***)**

This function is used by the application to register the transmit complete callback function with the Ethernet driver. The driver calls the callback function to free the data buffer, after the data has been transmitted by the controller. The callback function is passed as argument to this function.

Defined in: eth_api.h

Return Value

Returns Ethernet status

ETH_SUCCESS

Change of configuration settings is success

Parameters

ETH_device

Ethernet controller hardware module identification number

APP_tx_cb

Pointer to application call back routine

Implementation Details

- Register the application provided call back function with the HAL. This callback function will be called by the driver in the DMUT ISR after a frmae has been transmitted by the controller.

- The callback function will send the data buffer back to the appl after which the application is free to reuse the buffer.
- The call back function should have a specific prototype E.g. void app_callback(ETH_TRANSFER *sent_buff)

3.31 ETH_write

ETH_STATUS ETH_write(ETH_DEVICE *ETH_device*, **ETH_TRANSFER *** *ETH_transfer***)**

Application writes data/frame to the driver, this function will associate the data(frame) received from application to the DMUT free descriptor list. As soon as the data buffer is associated with DMUT, this function will return the ETH_SUCCESS to calling function. If there is any failure it will return the failure status. The DMUT will automate the process of transmission.

Defined in: eth_api.h

Return Value

Returns Ethernet status

ETH_SUCCESS

The frame is associated to DMUT

ETH_ERR

The frame is not associated to DMUT, because of no free descriptors are available

Parameters

ETH_device

Ethernet controller hardware module identification number

ETH_transfer

Pointer to frame data

Implementation Details

- The DMUT will maintain 2 descriptor lists, one is free and another one is fill descriptor list.
- The frame received from application will be associated to free d descriptor. This free descriptor will be disassociated from free descriptor list and attached to the tail end of DMUT fill descriptor list.
- Check for the DMUT hold bit status, if the DMUT sets the hold bit then it issues a hold reset command.
- Always at least one free descriptor is maintained to protect global variables, which in turn make the ISRs reentrant.

4 Configure/Optimize the Ethernet HAL

4.1 Configuring the Ethernet HAL

Although the Ethernet HAL can be used immediately without configuring it to suit a particular application it will often be the case that some features written into the HAL will either be unnecessary; degrade performance to an unacceptable level, take up too much memory or only be required for debugging purposes. For this reason the Ethernet HAL has been designed in such a way that it can be easily configured to remove unused features, a number of optional features may be enabled and/or disabled through the ETH_CFG.H file:

- –ETH device number checking
- –Initialisation check on API calls

Additionally further options which affect the Ethernet HAL may be present in the System HAL. Depending on the actual system in question these, or other, options may be available:

```
-On the fly peripheral clock changing
-Initial interrupts numbers/priorities settings
-ETH physical interface (GPIO) configuration
```

The System HAL User Guide should be available from the same source as this document, please refer to it for more details on the available settings and features.

4.2 Ethernet Driver HAL configuration parameters

4.2.1 ETH_CFG_DEV_CHK macro

#define ETH_CFG_DEV_CHK

Defined in: eth_cfg.h

Ethernet device check, this define selects whether device ID checking will be performed in the Ethernet HAL API functions. Disabling this feature will result in less code being generated. This checking is normally used when the h/w supports more than one similar module.

1	Enable the feature
0	Disable the feature

4.2.2 ETH_CFG_INIT_CHK macro

#define ETH_CFG_INIT_CHK

Defined in: eth_cfg.h

Ethernet initialization check, this define checks if the HAL has been initialized before executing. Disabling this feature will result in less code being generated. This also acts as a safe guard mechanism

1	Enable the feature
0	Disable the feature

4.2.3 ETH_CFG_STAT_LOG macro

#define ETH_CFG_STAT_LOG

Defined in: eth_cfg.h

Ethernet log statistics enable, this define selects whether the ETH HAL should maintain a log of successfully received/transmitted frames and frames with errors on each peripheral it controls. This allows application software to validate the reliability of the connection.

If this is disabled, then driver handles the interrupt which are critical to RX and TX.

Disabling this feature will result in smaller code and less data.

1	Enable the feature
0	Disable the feature

4.2.4 ETH_CFG_USE_CAM macro

#define ETH_CFG_USE_CAM

Defined in: eth_cfg.h

This define select whether the CAM feature should be included or excluded from the driver code. By using CAM the application would be accept or reject the frames from the particular PHY devices by programming the corresponding MAC address into CAM. CAM also provides the facilities like accepting/reject unicast, multicast or broadcast frames.

1	Enable the feature
0	Disable the feature

4.3 API Function Exclusion

If certain API functions are not required then they may be removed in order to reduce code size. In the HAL distribution all the API functions will be included by default, in order to remove an API function the relevant define should located and value is changed from 1 to 0. This sections detail defines which are available to exclude API functions from the HAL. Set Macro value as 1 to include corresponding function code. Setting the value zero not to include the function code

4.3.1 ETH_CFG_FUNC_TERMINATE macro

#define ETH_CFG_FUNC_TERMINATE

Defined in: eth_cfg.h

This controls whether or not the ETH_terminate_dev API function is included.

1	Include the function code in driver code
0	Exclude the function code from driver code

4.3.2 ETH_CFG_FUNC_ABORT macro

#define ETH_CFG_FUNC_ABORT

Defined in: eth_cfg.h

This controls whether or not the ETH_abort API function is included.

1	Include the function code in driver code
0	Exclude the function code from driver code

4.3.3 ETH_CFG_FUNC_STATUS macro

#define ETH_CFG_FUNC_STATUS

Defined in: eth_cfg.h

This controls whether or not the ETH_status_dev API function is included.

1	Include the function code in driver code
0	Exclude the function code from driver code

4.3.4 ETH_CFG_FUNC_CONTROL macro

#define ETH_CFG_FUNC_CONTROL

Defined in: eth_cfg.h

This controls whether or not the ETH_control_dev API function is included.

1	Include the function code in driver code
0	Exclude the function code from driver code

4.3.5 ETH_CFG_FUNC_READ macro

#define ETH_CFG_FUNC_READ

Defined in: eth_cfg.h

This controls whether or not the ETH_read API function is included. Application cannot read the data received from PHY medium if this feature is disabled.

1	Include the function code in driver code
0	Exclude the function code from driver code

4.3.6 ETH_CFG_FUNC_WRITE macro

#define ETH_CFG_FUNC_WRITE

Defined in: eth_cfg.h

This controls whether or not the ETH_write API function is included. Application cannot send data to PHY medium if this feature is disabled.

1	Include the function code in driver code
0	Exclude the function code from driver code

4.3.7 ETH_CFG_FUNC_CTRL_MAC macro

#define ETH_CFG_FUNC_CTRL_MAC

Defined in: eth_cfg.h

This controls whether or not the ETH_ctrl_mac API function is included. By disabling this feature the application is not able to change the MAC configuration at run time.

1	Include the function code in driver code
0	Exclude the function code from driver code

4.3.8 ETH_CFG_FUNC_CTRL_CAM macro

#define ETH_CFG_FUNC_CTRL_CAM

Defined in: eth_cfg.h

This controls whether or not the ETH_ctrl_mac_cam API function is included. The application cannot add the MAC addresses and change the configuration settings at run time if this feature is disabled.

1	Include the function code in driver code
0	Exclude the function code from driver code

4.3.9 ETH_CFG_FUNC_CTRL_MAC_TX macro

#define ETH_CFG_FUNC_CTRL_MAC_TX

Defined in: eth_cfg.h

This controls whether or not the ETH_ctrl_mac_tx API function is included. If this feature is disabled the application cannot change the MAC TX configuration settings during run time.

1	Include the function code in driver code
0	Exclude the function code from driver code

4.3.10 ETH_CFG_FUNC_CTRL_MAC_RX macro

#define ETH_CFG_FUNC_CTRL_MAC_RX

Defined in: eth_cfg.h

This controls whether or not the ETH_ctrl_mac_rx API function is included. If this feature is disabled the application cannot change the MAC receiver configuration settings during run time.

1	Include the function code in driver code
0	Exclude the function code from driver code

4.3.11 ETH_CFG_FUNC_CTRL_MAC_PHY macro

#define ETH_CFG_FUNC_CTRL_MAC_PHY

Defined in: eth_cfg.h

This controls whether or not the ETH_ctrl_mac_phy API function is included. Application may not able to program the PHY devices at run time if this feature is disabled.

1	Include the function code in driver code
0	Exclude the function code from driver code

4.3.12 ETH_CFG_FUNC_CTRL_TB macro

#define ETH_CFG_FUNC_CTRL_TB

Defined in: eth_cfg.h

This controls whether or not the ETH_ctrl_tb API function is included. To change the threshold codes at run time this feature has to be enabled.

1	Include the function code in driver code
0	Exclude the function code from driver code

4.3.13 ETH_CFG_FUNC_CTRL_DMUR macro

#define ETH_CFG_FUNC_CTRL_DMUR

Defined in: eth_cfg.h

This controls whether or not the ETH_ctrl_dmur API function is included. By disabling this feature the application may not be able to change the DMUR endian and alignment modes at run time.

1	Include the function code in driver code
0	Exclude the function code from driver code

4.3.14 ETH_CFG_FUNC_CTRL_DMUT macro

#define ETH_CFG_FUNC_CTRL_DMUT

Defined in: eth_cfg.h

This controls whether or not the ETH_ctrl_dmut API function is included. The application is not able to change the endian mode at run time if this feature is disabled.

1	Include the function code in driver code
0	Exclude the function code from driver code

Ethernet driver initialization parameters

The application or user may configure the driver initialization parameters. These parameters can be changed at run time provided the corresponding IOCTL function is included in driver. This can be done by enabling defines ETH_CFG_FUNC_CTRL_xx.

4.3.15 ETH_CFG_FULLDUPLEX macro

#define ETH_CFG_FULLDUPLEX

Defined in: eth_cfg.h

Full duplex - Receive and transmit data simultaneously

Half duplex - Either receive or transmit data at a specified time.

1	Enable the full duplex feature
0	Enable half duplex feature

4.3.16 ETH_CFG_MAC_LOOPBACK macro

#define ETH_CFG_MAC_LOOPBACK

Defined in: eth_cfg.h

In MAC loop back mode the frames transmitted at MAC TX will be received at MAC RX. The data won't be propagated through the PHY device. This feature mainly used for debugging the Ethernet controller. This feature would be enabled when Ethernet controller operates in full duplex mode.

1	Enable the feature
0	Disable the feature

4.3.17 ETH_CFG_PHYDEV_LOOPBACK macro

#define ETH_CFG_PHYDEV_LOOPBACK

Defined in: eth_cfg.h

In PHY device loop back mode the frames transmitted at MAC TX will be received at MAC RX. The data will be propagated through the PHY device. This feature mainly used for debugging the Ethernet controller including PHY device. This feature would be enabled when Ethernet controller operates in full duplex mode.

1	Enable the feature
0	Disable the feature

4.3.18 ETH_CFG_SPEED macro

#define ETH_CFG_SPEED

Defined in: eth_cfg.h

By default Ethernet controller support both 100Mbps and 10Mbps modes. This configuration value is used to configure the PHY device which is attached to Ethernet controller.

1	100Mbps mode
0	10Mbps mode

4.3.19 ETH_CFG_USE_PAUSE macro

#define ETH_CFG_USE_PAUSE

Defined in: eth_cfg.h

This feature is used to send control frame when the device operates in full duplex mode. If this feature is enabled the driver will send PAUSE when there are no resources(data buffers) to receive the incoming data. If this feature is enabled the address (01-80-C2-00-00-01) is programmed into CAM (ETH_CFG_MACADDRS0) to use as destination address for PAUSE. The ETH_CFG_MACADDRS2 will be a source address.

1	Feature is enabled
0	Feature is disabled

4.3.20 ETH_CFG_PAUSE_TIME macro

#define ETH_CFG_PAUSE_TIME

Defined in: eth_cfg.h

The first 2 most significant bytes indicate the length of pause control frame, ranges from 0x00 to 0xFFFF. This length is used in pause control frame.

The length of pause is measured in quanta, which is equal to 512 bit times.

4.3.21 ETH_CFG_USE_CAM macro

#define ETH_CFG_USE_CAM

Defined in: eth_cfg.h

This feature is used to filter the incoming frames on basis of destination address encoded in received frame header with the MAC address in CAM table.

The user can program the desired MAC addresses in CAM.

1	Feature is enabled
0	Feature is disabled

4.3.22 ETH_CFG_NEGATIVECAM macro

#define ETH_CFG_NEGATIVECAM

Defined in: eth_cfg.h

The MAC controller will reject the frame which CAM recognizes and accept the other frames. This feature has an effect when CAM is enabled.

1	Reject the frames which are CAM recognizes, accept others
0	Accept all frames, which is recognized by CAM

4.3.23 ETH_CFG_UNICAST macro

#define ETH_CFG_UNICAST

Defined in: eth_cfg.h

Accept all frames with the destination address of a single host. When this feature is enabled CAM does not have any effect on unicast frames. When this feature is disabled and CAM is enabled, controller will accept the frames which are recognized by CAM.

1	Accept all unicast frames.

0

Accept the frames which are recognized by CAM, provided CAM is enabled

4.3.24 ETH_CFG_MULTICAST macro

#define ETH_CFG_MULTICAST

Defined in: eth_cfg.h

Accept all the frames which have the multicast address as destination address irrespective of CAM table.

1

Accept all frames with multicast address as destination address

0

Reject multicast frames

4.3.25 ETH_CFG_BROADCAST macro

#define ETH_CFG_BROADCAST

Defined in: eth_cfg.h

Accept all the frames which have the broadcast address as destination address irrespective of CAM table.

1

Accept all frames with broadcast address as destination address

0

Reject broadcast frames

4.3.26 ETH_CFG_MACADDR_STRT_LOC macro

#define ETH_CFG_MACADDR_STRT_LOC

Defined in: eth_cfg.h

Starting location of MAC addresses which are programmed into CAM. If the pause feature is enabled then this value should be zero. The sum of ETH_CFG_MACADDR_STRT_LOC ranges from 0x0 to 0x13.

4.3.27 ETH_CFG_NUM_OF_MACADDR macro

#define ETH_CFG_NUM_OF_MACADDR

Defined in: eth_cfg.h

The user can program up to 20 MAC addresses including the pause control address fields. If the pause feature is not used then the user can program up to 20 MAC addresses to filter the receiving frames from PHY medium. Ranges from 0x1 to 0x14 The sum of ETH_CFG_MACADDR_STRT_LOC and ETH_CFG_NUM_OF_MACADDR should be less than or equal to 0x14.

4.3.28 ETH_CFG_ENBL_MACADDRS macro

#define ETH_CFG_ENBL_MACADDRS

Defined in: eth_cfg.h

The following define is used to enable the MAC addresses programmed into CAM. Each bit value corresponds to one MAC address i.e.

The MAC address 0 corresponds to least significant bit of ETH_CFG_ENBL_MACADDRS.

The MAC address 1 corresponds to second least significant bit of ETH_CFG_ENBL_MACADDRS.

The MAC address 2 corresponds to third least significant bit of ETH_CFG_ENBL_MACADDRS.

4.3.29 ETH_CFG_ADDPAD macro

#define ETH_CFG_ADDPAD

Defined in: eth_cfg.h

If ETH_CFG_ADDPAD is enabled the MAC TX controller will add the pad automatically, when the transmitted frame size less is than 64 bytes.

1

Enable the feature

0

Disable the feature

4.3.30 ETH_CFG_ADDCRC macro

#define ETH_CFG_ADDCRC

Defined in: eth_cfg.h

MAC TX controller will calculate and append the CRC to the out going frame.

1

Enable the feature

0

Disable the feature

4.3.31 ETH_CFG_XDEFER macro

#define ETH_CFG_XDEFER

Defined in: eth_cfg.h

MAC TX controller will check for excessive delays for the out going frame.

1

Enable the feature

0

Disable the feature

4.3.32 ETH_CFG_SQECHECK macro

#define ETH_CFG_SQECHECK

Defined in: eth_cfg.h

MAC TX controller will check for Signal Quality Error in 10Mbps mode

1

Enable the feature

0

Disable the feature

4.3.33 ETH_CFG_RCVSHORT macro

#define ETH_CFG_RCVSHORT

Defined in: eth_cfg.h

MAC receive controller will accept the frames with frame length less than 64 bytes.

1

Enable the feature

0

Disable the feature

4.3.34 ETH_CFG_RCVLONG macro

#define ETH_CFG_RCVLONG

Defined in: eth_cfg.h

MAC RX controller will accept the frames with frame length more than 1518 bytes and long frame received interrupt won't be occurred upon receiving long frames.

1

Enable the feature

0

Disable the feature

4.3.35 ETH_CFG_STRIPCRC macro

#define ETH_CFG_STRIPCRC

Defined in: eth_cfg.h

MAC RX controller will strip the CRC from received frames before it storing on system memory.

1

Enable the feature

0

Disable the feature

4.3.36 ETH_CFG_PASSCTRLPKTS macro

#define ETH_CFG_PASSCTRLPKTS

Defined in: eth_cfg.h

MAC controller will pass the received control frames to the upper layers(Logical Link Control layer).

1

Enable the feature

0

Disable the feature

4.3.37 ETH_CFG_NOTCHECKCRC macro

#define ETH_CFG_NOTCHECKCRC

Defined in: eth_cfg.h

MAC RX controller will not check the CRC of received frames.

1

Enable the feature

0

Disable the feature

4.3.38 ETH_CFG_AUTONEGOTIATE macro

#define ETH_CFG_AUTONEGOTIATE

Defined in: eth_cfg.h

Auto-negotiation is a mechanism that enables devices to negotiate the SPEED and MODE (duplex or half-duplex) of an Ethernet Link. This bit can be set provided the PHY device also support this feature.

1

Enable the feature

0

Disable the feature

This feature is not supported in this version.

4.3.39 ETH_CFG_TBFTC macro

#define ETH_CFG_TBFTC

Defined in: eth_cfg.h

The Transmit Buffer forward threshold code determines number of buffer locations which must be filled before protocol machines (e.g. Ethernet MAC) start transmission/segmentation. Nevertheless, the Transmit Buffer forwards data packets to the protocol machine as soon as an entire packet or the end of a packet is stored in the Transmit Buffer.

0	Correspond to 1 DWORD
1	Correspond to 4 DWORDS
2	Correspond to 8 DWORDS
3	Correspond to 12 DWORDS
4	Correspond to 16 DWORDS
5	Correspond to 24 DWORDS
6	Correspond to 32 DWORDS
7	Correspond to 40 DWORDS

4.3.40 ETH_CFG_TBRTC macro

#define ETH_CFG_TBRTC

Defined in: eth_cfg.h

The internal Transmit Buffer has a programmable number of buffer locations per channel. When number of free locations reaches Transmit Buffer refill threshold, internal Transmit Buffer requests new data from the DMUT.

0	Correspond to 1 DWORD
1	

Correspond to 4 DWORDS

2

Correspond to 8 DWORDS

3

Correspond to 12 DWORDS

4

Correspond to 16 DWORDS

5

Correspond to 24 DWORDS

6

Correspond to 32 DWORDS

7

Correspond to 40 DWORDS

4.3.41 ETH_CFG_BUFFSIZE macro

#define ETH_CFG_BUFFSIZE

Defined in: eth_cfg.h

The Transmit Buffer size configures the number of internal Transmit Buffer locations for a particular channel. Buffer locations will be allocated on command transmit init and released after command transmit off.

The sum of transmit forward threshold and the transmit refill threshold must be smaller than the internal buffer size.

Ranges from 0x00 to 0xFF

4.3.42 ETH_CFG_DMUR_ENDIAN macro

#define ETH_CFG_DMUR_ENDIAN

Defined in: eth_cfg.h

Data Management Unit Receive will store the received frame in big/little endian mode.

0

Little endian

1

Big endian

4.3.43 ETH_CFG_DMUT_ENDIAN macro

#define ETH_CFG_DMUT_ENDIAN

Defined in: eth_cfg.h

Data Management Unit Transmit will transmit the frames in big/little endian mode.

0

Little endian

1

Big endian

4.3.44 ETH_CFG_DMUR_DESC_NUM macro

#define ETH_CFG_DMUR_DESC_NUM

Defined in: eth_cfg.h

The number of descriptors and data buffers is maintained by DMUR. Each descriptor will be associated with one data buffer.

4.3.45 ETH_CFG_DMUR_DBUFF_SIZE macro

#define ETH_CFG_DMUR_DBUFF_SIZE

Defined in: eth_cfg.h

Size of data buffers which are associated with DMUR descriptor. It is recommended that each data buffer size is good enough to receive complete frame, it will save the execution time. Note that the device handles the status (CRC, frame status) of frame based protocols (Ethernet) internally in the same way as payload data. Therefore, byte number should include four bytes more than the maximum length of incoming frames.

4.3.46 ETH_CFG_DMUT_DESC_NUM macro

#define ETH_CFG_DMUT_DESC_NUM

Defined in: eth_cfg.h

The number of descriptors is maintained by DMUT. One descriptor is not being used actively, to make the ISRs re-entrant.

5 Cache memory usage for Ethernet LLD

The usage of the Cache memory is not recommended for the device with BA-step, it is STILL possible to use CACHE memory on TC1130 BA-step, as long as DMI12 workaround is switched ON with GNU Compiler options.(But this is not recommended)

With Tasking compiler, it is not possible to use CACHE memory on TC1130 BA-step, because there is no workaround available.

With TC1130 BB-step, ALL compilers can be used. Especially for GNU compiler, workaround DMI12 should be disabled, so that the performance is not reduced by the workaround.

To use the cache memory along with Ethernet LLD following changes are required as stated below

1. **ETH_CFG.h** file

Define ETH_CFG_USE_CACHE to use the cache memory, this definition is used to place the DMUR and DMUT API definitions in cache memory location. Failing to do this may lead to unexpected results. This definition would not show any effect when Tasking compiler is used.

#define ETH_CFG_USE_CACHE

2. Loader script

Define the cache segment in MEMORY block as shown below

cache_cram (awx!p):org = 0x80000000, len = 512K

cache_cram --> the memory block name (user defined)

(awx!p) --> loader script memory control flags

0x80000000 --> cache memory segment address

512k --> length of cache memory segment

SECTIONS area need to be updated as shown below

. = 0x80000000 ;

.eth_cache ALIGN(32) :

```
{
  *(.bss.32align.*)
} > cache_cram
```

.= 0x80000000

need to be defined to place the definitions of DMUR and DMUT APIs in cache memory segment.

The italic and highlighted portion is the code need to be updated in the corresponding files as explained above.

6 Application Examples

This section presents a number of simple example applications using the Ethernet HAL which cover the most commonly be used Ethernet HAL API functions.

6.1 Example illustrating the initialisation and basic operation of the driver

This example explains how to write applications to

- initialize the LLD by calling ETH_initialise_dev()

- enable global Interrupts by calling ENABLE_GLOBAL_INTERRUPT()

- register the transmit and receive callback functions

  - Receive Callback Function. This will be called when a packet is received by the driver. The application is required to copy the data from the driver's buffer to its own buffer, in the callback function.

  - Transmit Callback Function. This will be called when the hardware has transmitted a packet. The application Can now re-use / free the buffer in the callback function.

- transmit packets by calling ETH_write() API. This application first creates an ARP packet to send out & then calls ETH_write() to send the packet on the wire. An ARP packet was selected so as to give the user a 'feel' of an ethernet packet. Normally the TCP/IP stack takes care of this.

- receive packets. The receive callback function is called when the driver receives a packet.This application copies the data in its own buffer.

```
/* Include files */
#include "ETH_API.H"
#include "SYS_API.H"
#ifndef  IFX_COMPILER_GNU
#include <string.h>
#endif

#define ARP_PKT_LEN  64
#define ETH_MTU_SIZE 1500
/*
The ETHERNET_DEVICE is the device number used in calling the API functions.
*/
#define ETHERNET_DEVICE 0
```

```
/* Number of frames which could not be transmitted by the DMUT*/
IFX_UINT32  tx_frames_dropped = 0;

/* Number of frames which were transmitted successfully */
IFX_UINT32  tx_frames = 0;

/* Number of packets that were received successfully */
IFX_UINT32  rx_packets= 0;

/* Data buffer where the rxed data will be copied into */
IFX_UINT8   rx_data[1500];

/* The "data" portion of the pkt to be transmitted. This is just an example provided here
to illustrate the formation of an ethernet packet. This data may be changed to suit the
user's requirements */
IFX_UINT8   tx_pktdata[] = {0x00, 0x01, 0x08, 0x00, 0x06, 0x04, 0x00,
                           0x01, 0x00, 0x08, 0xA1, 0x3E, 0x9E, 0xD5,
                           0xC1, 0xAC, 0x44, 0x02, 0x00, 0x00, 0x00,
                           0x00, 0x00, 0x00, 0xC1, 0xAC, 0x44, 0x01,
                           0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                           0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                           0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                           0x00};

/* The data buffer containing the tx pkt to be transmitted */
IFX_UINT8   tx_buf[ARP_PKT_LEN];

/* The transmit callback function prototype */
void App_tx_cb(ETH_TRANSFER *sent_buff);

/*
Prototype of the receive callback function which is registered by the application. This
function is called by the driver when packets are received.
```

The function takes a pointer to the ETH_TRANSFER structure. The driver passes the pointer to the data received in the ETH_buffer field of the structure. The application should copy the data pointed to by ETH_buffer in a locally allocated buffer.

The application SHOULD NOT release the memory pointed to by the ETH_buffer pointer.

```
*/
void App_rcv_cb(ETH_TRANSFER *rcv_frm);

/* Declare an array to receive the data */
IFX_UINT8 ETH_rx_data[RX_DATA_BUF_SIZE];
/* This function performs a byte wise copy from source to dest */
void *copy(void *dest, const void *src, IFX_UINT32 len);



/*
Main() function which initializes the driver.
*/
int main()
{
  ETH_COM_PARAMS   ETH_params;
  IFX_UINT8   ETH_count  = 0;
  ETH_TRANSFER  ETH_transdata;

/* Initialise the Ethernet controller */
  ETH_initialise_dev(ETHERNET_DEVICE, &ETH_params);

/*Enable Global Interrupts */
ENABLE_GLOBAL_INTERRUPT();

/* Register the application call back function by a call to the read API. */
  ETH_read(ETHERNET_DEVICE, App_rcv_cb);

/* Make a ethernet frame to transmit */
/* Store the destination address. Let the destination address be 00:08:A1:3E:9E:D5  */
```

```
  tx_buf[0] = 0x00;
  tx_buf[1] = 0x08;
  tx_buf[2] = 0xA1;
  tx_buf[3] = 0x3E;
  tx_buf[4] = 0x9E;
  tx_buf[5] = 0xD5;
/* Store the source address. Let the source address be 00:08:00:0B:19:34. This may be
changed to suit the user's requirements */
 */
  tx_buf[6] = 0x00;
  tx_buf[7] = 0x08;
  tx_buf[8] = 0x00;
  tx_buf[9] = 0x0B;
  tx_buf[10] = 0x19;
  tx_buf[11] = 0x34;

/* Store the protocol type. ARP in this case */
 tx_buf[12] = 0x08;
 tx_buf[13] = 0x06;

  /* User data in the IP packet */
  copy(&tx_buf[14],tx_pktdata,50);

  /* Give a delay for the autonegotiation to complete */
  delay();

  /* Transmit the packet, say 100 times */
  for(i = 0; i < 100; i++)
  {
    ETH_transdata.ETH_buffer = tx_buf;
    ETH_transdata.ETH_buffer_size = ARP_PKT_LEN;
    if((ETH_write(0, &ETH_transdata)) != ETH_SUCCESS)
    {
      tx_frames_dropped++;
    }
   }
    /* Loop infinitely so that we keep on receiving packets */
   while(1);
}

/* The Application receive call back function */
void App_rcv_cb(ETH_TRANSFER *rcv_frm)
{
   IFX_UINT32 len;
   len = rcv_frm->ETH_buffer_size;
  /* Check if we have received a complete frame.
  If so increment counter to update statistics.
  */
  if(rcv_frm->ETH_return_num.ETH_fe == 1)
  {
    rx_packets++;
    copy(rx_data,rcv_frm->ETH_buffer,len);
  }
}
void App_tx_cb(ETH_TRANSFER *sent_buff)
{
  tx_frames++;
}

void *copy(void *dest, const void *src, IFX_UINT32 len)
{
  IFX_UINT8 *dstp = (IFX_UINT8*)dest;
  IFX_UINT8 *srcp = (IFX_UINT8*) src;
  IFX_UINT32 i;

  for (i = 0; i < len; ++i)
  {
    dstp[i] = srcp[i];
  }
  return dest;
}
```

6.2 Example illustrating the programming of the CAM at run time.

Suppose the following MAC addresses have to be programmed
0x123456789ABC and
0x010203040506
at location 4 in the CAM.

```
# define ETHERNET_DEVICE 0

main()
{
   ETH_COM_PARAMS   ETH_params;
   ETH_TRANSFER  ETH_transdata;

  /* Program the MAC addresses in an array */
  IFX_UINT32 ETH_macaddr[] = {0x12345678,0x9abc0102, 0x03040506};
  ETH_CAM_CTRL  ETH_cam_prog;

   /* Initialise the driver */
   ETH_initialise_dev(ETHERNET_DEVICE, &ETH_params);

  /*Enable Global Interrupts */
  ENABLE_GLOBAL_INTERRUPT();

   /* Update the fields of the ETH_COM_PARAMS structure */
   ETH_cam_prog.command = ETH_ADD_MAC_ADDR;
   ETH_cam_prog.data.mac_addr_data.mac_addrs          = ETH_macaddr;
   ETH_cam_prog.data.mac_addr_data.no_of_mac_addrs = 2;
   ETH_cam_prog.data.mac_addr_data.start_mac_addr     = 4;

   /* Call the Control CAM API to program the CAM */
  if(ETH_ctrl_cam(ETHERNET_DEVICE, &ETH_cam_prog) == ETH_ERR)
   {
       /*  Handle error */
    }
   /* Do other processing here….*/
}
```

6.3 Example illustrating the use of the 'fe' bit in the receive path

When a complete Ethernet frame cannot fit into a receive buffer, the driver sends the received data to the application in more than one buffers. The driver indicates the end of a frame to the application by setting the 'fe' bit in the ETH_return_num field of the ETH_transfer structure.

```
# define ETHERNET_DEVICE 0

/* Receive call back function */
void App_rcv_cb(ETH_TRANSFER *rcv_frm);

/* The RX_DATA_BUF_SIZE is for defining the size of the buffer that is allocated to copy
the data to which is received.
*/
#define RX_DATA_BUF_SIZE  1536

/* Declare an array to receive the data */
IFX_UINT8 ETH_rx_data[RX_DATA_BUF_SIZE];

main()
{
  ETH_COM_PARAMS   ETH_params;
  ETH_TRANSFER  ETH_transdata;
  /* Initialise a pointer to the beginning of the receive data buffer */
  IFX_UINT8 *ETH_rxdata_ptr = ETH_rx_data;

  /* Initialise the driver */
  ETH_initialise_dev(ETHERNET_DEVICE, &ETH_params);
 /*Enable Global Interrupts */
 ENABLE_GLOBAL_INTERRUPT();


  /* Register the application call back function by a call to the read API. */
  ETH_read(ETHERNET_DEVICE, App_rcv_cb);
  /* Do other processing here….*/

  /* Loop infinitely so that we keep on receiving packets */
  while(1);
}

/* The Application receive call back function */
void App_rcv_cb(ETH_TRANSFER *rcv_frm)
{
  IFX_UINT8 count;

   /* Copy the data from the driver buffer to the applications buffer
      Advance the ETH_rxdata_ptr as we receive data and when the FE bit is set,
      a complete ethernet frame has been copied by the application */
   for(count = 0; count < rcv_frm->ETH_buffer_size; count++)
  {
     ETH_rxdata_ptr++ = rcv_frm->ETH_buffer[count];
   }
  /* Check for the FE bit */
  if(rcv_frm->ETH_return_num.ETH_fe == 1)
  {
     /* We have received a complete ethernet packet */
     /* Now we have to use another buffer to store the data received or -
        as in this case we are re- cycling the buffer. So we set ETH_rxdata_ptr
       to the beginning of the same buffer. This might not make too much
       sense, all we are illustrating is the use of the 'fe' bit.
    */
     ETH_rxdata_ptr = ETH_rx_data;
  }
```

}

6.4 Changing communication parameters + GPIO cfg example

The Interrupt priorities and the GPIO Port allocations are configured through the SYS_CFG.H file.

To configure the ports the following parameters have to be programmed :

Peripheral Module - Name of the macro which includes the name of the peripheral and the port line (Transmit/Receive).

Port - Port Number.

Pin - Bit Number in the Port.

Dir - Value of the bit in the Dir register.

Alt0 - Value of the bit in the Altsel0 register.

Alt1 - Value of the bit in the Altsel1 register.

Od - Value of the bit in the Open Drain register.

Pullsel - Value of the bit in the Pull up/Pull down selection register.

Pullen - Value of the bit in the Pull up/Pull down enable register.

The user may use -1, to indicate an unused (or dont care) value.

The user may not change the name of the macros in the SYS_CFG.H.

Example:

```
                Peripheral Module  Port  Pin  Dir   Alt0   Alt1  Od   Pullsel   Pullen
#define SYS_GPIO_ETH_TX0    1,    0,     1,     1,     0,    -1,    -1,        -1
```

In the exaple given above the macro SYS_GPIO_ETH_TX0 defines the following configuration :

Peripheral Module - Ethernet

Port - 1

Pin - 0

Dir - 1

Alt0 - 1

Alt1 - 0

Od - Not Used

Pullsel - Not Used

Pullen - Not Used

6.5 Application note on the disabling of interrupts in the Interrupt Service Routine

From the hardware perspective, an Interrupt Service Routine(ISR) is entered with the interrupt system globally disabled. The Low Level Driver(LLD) does not enable global interupt in the ISR as the LLD ISRs are kept short. Most LLD ISRs invoke a callback function that was registered by the application. If required, the application may enable global interrupts (by calling ENABLE_GLOBAL_INTERRUPT()) at the beginning of the ISR callback function.

Application Note for the MAC Controller

According to the IEEE Reference Model, the data link layer is split up into two sublayers:

Logical Link Control (LLC) and Media Access Control (MAC).

The LLC layer functions are:

On transmission, assemble data into frames with address and CRC fields.

On reception, disassemble frame, perform address recognition and CRC validation.

As shown in the figure above the Low level driver and the MAC Controller are in the MAC layer, hence the encapsulation of the Ethernet frame is the responsibility of the “layer“ above the driver. Normally this is taken care of by the Operating System (eg. eLinux). The driver is designed like this as this is the interface expected by the OS.

The MAC Address is a 48 bit address which is unique to a particular NIC (Network Interface Card), and is obtained by the NIC manufacturer from the IEEE. The user of the TC1130 chip will normally design a board with an appropraite memory layout, so that the MAC address for that board can be stored in a PROM or ROM along with other configuration information. The MAC address would be accessible to the application code and/or to the higher layer protocol at a defined memory location. The user may also choose to implement an access / wrapper function to return the MAC address.

In case of TC1130 Triboard as this is only a reference design, there is no special arrangement provided for storing/ accessing teh MAC address. The user’s application software is responsible for the MAC Address - either as a #define (sufficient for prototypes or demo programs), or by allocating a special memory location in the FLASH or EEPROM.

In the receive path, a complete ethernet frame is delivered to the “layer“ above the driver. The user can configure the driver to either strip off the CRC, at the hardware or leave it intact.

The user should program the MAC Address of the devices, from which he wishes to receive data from, in the CAM. The driver can be configured to receive broadcast, multicast and unicast data by setting the appropriate CFG macros as described in the section 3 of this document.

7 Related Documentation

Infineon Technologies HAL/Device Driver Software Suite Overview

Ethernet LLD design document version 2.0

TC1130 System Manual

TC1130 Peripheral Manual

This documents should be available from the same source as this User Manual.

8 Appendix A - Infineon IFX types

To overcome the problem of the size of data types changing between different compilers the HAL software modules use IFX types. These are defined in a file called COMPILER.H which is generated for each compiler that is supported. **Table** presents these IFX types.

Table 1 **Table of IFX Data Types**

IFX_UINT8	Unsigned 8 bit integer
IFX_UINT16	Unsigned 16 bit integer
IFX_UINT32	Unsigned 32 bit integer
IFX_SINT8	Signed 8 bit integer
IFX_SINT16	Signed 16 bit integer
IFX_SINT32	Signed 32 bit integer
IFX_VUINT8	Unsigned 8 bit volatile integer
IFX_VUINT16	Unsigned 16 bit volatile integer
IFX_VUINT32	Unsigned 32 bit volatile integer
IFX_VSINT8	Signed 8 bit volatile integer
IFX_VSINT16	Signed 16 bit volatile integer
IFX_VSINT32	Signed 32 bit volatile integer
IFX_SFLOAT	Signed flaot
IFX_STINT8	Signed static 8 bit integer
IFX_STINT16	Signed static 16 bit integer
IFX_STINT32	Signed static 32 bit integer
IFX_STUINT8	Unsigned static 8 bit integer
IFX_STUINT16	Unsigned static 16 bit integer
IFX_STUINT32	Unsigned static 32 bit integer

9 Appendix B - The System HAL

This appendix presents a brief description of the ASC related settings and options available in the System HAL.

9.1 SYS HAL Configurable Parameters

This section defines the configurable parameters of the SYS HAL - interrupts, GPIO ports, and the clock. The user may change only the value associated with the macros to suit application requirements. However, the user may NOT change the name of the macro.

9.1.1 System Clock Frequency

The clock must be operational before the controller can function. This clock is connected to the peripheral clock control registers, so changing the value of this clock frequency will affect all peripherals. The individual peripherals can scale down this frequency according to their requirements, for more details please refer to the corresponding user guide documents.

9.1.1.1 SYS_CFG_USB_DEVICE_ENABLE macro

#define SYS_CFG_USB_DEVICE_ENABLE

Defined in: SYS_CFG.H

User needs to configure whether the USB device has been used.

1

Equate this macro to 1 if onchip USB device is used.

0

Equate this macro to 0 if usb device is not used (default).

9.1.1.2 SYS_CFG_USB_ONCHIP_CLK macro

#define SYS_CFG_USB_ONCHIP_CLK

Defined in: SYS_CFG.H

User can configure the USB clock generation logic whether it internal or external. If clock is external, it will be derived from pin P4.0.

1

Equate this macro to 1 for internal clock generation

0

Equate this macro to 0 for external clock generation

9.1.1.3 SYS_CFG_USB_CLK_DIVISOR macro

#define SYS_CFG_USB_CLK_DIVISOR

Defined in: SYS_CFG.H

User needs to configure the USB clock ratio based upon the USB clock frequency. Since clock frequency can be either 48 MHZ, 96 or 144 MHZ, the ratio can 1, 2 or 3 respectively.

9.1.1.4 SYS_CFG_OSC_FREQ macro

#define SYS_CFG_OSC_FREQ

Defined in: SYS_CFG.H

User has to configure this with external applied frequency.

9.1.1.5 SYS_CFG_CLK_MODE macro

#define SYS_CFG_CLK_MODE

Defined in: SYS_CFG.H

User needs to configure this macro to any one of the following clock operation mode.

0

Direct drive (CPU clock directly derived from external applied frequency, N, P, and K values are not considered).

1

PLL mode (N, P, K values will be considered to derive CPU clock frequency from external frequency)

2

VCO bypass/pre-scalar mode (N value not considered to derive CPU clock from external frequency).

9.1.1.6 SYS_CFG_FREQ_SEL macro

#define SYS_CFG_FREQ_SEL

Defined in: SYS_CFG.H

This define decide the frequency ration between CPU and system, this is independent from the clock mode selection(SYS_CFG_CLK_MODE).

0

Ratio of fcpu/fsys is 2.

1

Ratio of fcpu/fsys is 1 i.e. fcpu = fsys.

9.1.1.7 SYS_CFG_KDIV macro

#define SYS_CFG_KDIV

Defined in: SYS_CFG.H

User has to configure this with a value ranges from 1 to 16, used for both PLL and VCO bypass modes.

9.1.1.8 SYS_CFG_PDIV macro

#define SYS_CFG_PDIV

Defined in: SYS_CFG.H

User has to configure this with a value ranges from 1 to 8, used for both PLL and VCO bypass modes.

9.1.1.9 SYS_CFG_NDIV macro

#define SYS_CFG_NDIV

Defined in: SYS_CFG.H

User has to configure this with a value ranges from 1 to 128, used only for PLL mode.

Comments

Advisable value range is 20 to 100.

9.1.1.10 SYS_CFG_FIX_TC1130A_BUG macro

#define SYS_CFG_FIX_TC1130A_BUG

Defined in: SYS_CFG.H

User can use this definition for software workaround done for TC1130A at system driver and not at module level.

1

Enbale software work-around for hardware bug fixes.

0

Disbale software work-around for hardware bug fixes.

9.1.2 Interrupt Configuration

The Interupt priorities are assigned statically by the System HAL. The priorities defined in the SYS_CFG.H file are recommeded when all the peripherals are enabled. The user can edit the priorities according to application requirements.

Priorities range from 1 to 255. Each interrupt should have a unique priority. The lowest priority is 1and the highest priority is 255.

As mentioned above, the user may change only the values associated with the macros and NOT the macro names.

9.1.2.1 SYS_ETH_MACRX0 macro

#define SYS_ETH_MACRX0 40

Ethernet MAC `RX0 interrupt priority`

9.1.2.2 SYS_ETH_MACRX1 macro

#define SYS_ETH_MACRX1 41

Ethernet MAC `RX1 interrupt priority`

9.1.2.3 SYS_ETH_MACTX0 macro

#define SYS_ETH_MACTX0 42

Ethernet MAC `TX0 interrupt priority`

9.1.2.4 SYS_ETH_MACTX1 macro

#define SYS_ETH_MACTX1 43

Ethernet MAC `TX1 interrupt priority`

9.1.2.5 SYS_ETH_MACRB0 macro

#define SYS_ETH_MACRB0 44

Ethernet MAC `RB0 interrupt priority`

9.1.2.6 SYS_ETH_MACRB1 macro

#define SYS_ETH_MACRB1 45

Ethernet MAC `RB1 interrupt priority`

9.1.2.7 SYS_ETH_MACTB0 macro

#define SYS_ETH_MACTB 46

Ethernet MAC `TB interrupt priority`

9.1.2.8 SYS_ETH_DMUR macro

#define SYS_ETH_DMUR 47

Ethernet MAC `DMUR interrupt priority`

9.1.2.9 SYS_ETH_MACDMUT macro

#define SYS_ETH_MACDMUT 48

Ethernet MAC `DMUT interrupt priority`

9.2 General Purpose I/O Configuration

This section defines the configurable port settings of the peripherals. The GPIO setting macros define the following parameters:

Peripheral Module

- Name of the macro which includes the name of the peripheral and the port line (Transmit/Receive).

Port

- Port Number.

Pin

- Bit Number in the Port.

Dir

- Value of the bit in the Dir register.

Alt0

- Value of the bit in the Altsel0 register.

Alt1

- Value of the bit in the Altsel1 register.

Od

- Value of the bit in the Open Drain register.

Pullsel

- Value of the bit in the Pull up/Pull down selection register.

Pullen

- Value of the bit in the Pull up/Pull down enable register.

The user may use -1, to indicate an unused (or don't care) value.

The user may not change the name of the macros in the SYS_CFG.H.

9.2.1 SYS_GPIO_ETH_TX0 macro

`#define` SYS_GPIO_ETH_TX0

Port configuration for Ethernet TX0 line.

9.2.2 SYS_GPIO_ETH_TX1 macro

`#define` SYS_GPIO_ETH_TX1

Port configuration for Ethernet TX1 line

9.2.3 SYS_GPIO_ETH_TX2 macro

`#define` SYS_GPIO_ETH_TX2

Port configuration for Ethernet TX2 line

9.2.4 SYS_GPIO_ETH_TX3 macro

`#define` SYS_GPIO_ETH_TX3

Port configuration for Ethernet TX3 line

9.2.5 SYS_GPIO_ETH_TXER macro

`#define` SYS_GPIO_ETH_TXER

Port configuration for Ethernet TXER line

9.2.6 SYS_GPIO_ETH_TXEN macro

`#define` SYS_GPIO_ETH_TXEN

Port configuration for Ethernet TXEN line

9.2.7 SYS_GPIO_ETH_MDC macro

`#define` SYS_GPIO_ETH_MDC

Port configuration for Ethernet MDC line

9.2.8 SYS_GPIO_ETH_RXDV macro

`#define` SYS_GPIO_ETH_RXDV

Port configuration for Ethernet RXDV line

9.2.9 SYS_GPIO_ETH_CRS macro

`#define` SYS_GPIO_ETH_CRS

Port configuration for Ethernet CRS line

9.2.10 SYS_GPIO_ETH_COL macro

`#define` SYS_GPIO_ETH_COL

Port configuration for Ethernet COL line

9.2.11 SYS_GPIO_ETH_RX0 macro

`#define` SYS_GPIO_ETH_RX0

Port configuration for Ethernet RX0 line

9.2.12 SYS_GPIO_ETH_RX1 macro

`#define` SYS_GPIO_ETH_RX1

Port configuration for Ethernet RX1 line

9.2.13 SYS_GPIO_ETH_RX2 macro

`#define` SYS_GPIO_ETH_RX2

Port configuration for Ethernet RX2 line

9.2.14 SYS_GPIO_ETH_RX3 macro

`#define` SYS_GPIO_ETH_RX3

Port configuration for Ethernet RX3 line

10 Limitations

This section defines the Limitations of the Ethernet LLD HAL.release SW_M9

1. **Data Cache Limitation on TC1130 - BA step:** The transmit and receive data buffers should not be allocated in the cache-able segments(Segment - 0xC0000000 and 0x80000000). For workaround on TC1130 BB Step board refer to section 5.0