The following document contains information on Cypress products. The document has the series name, product name, and ordering part numbering with the prefix “MB”. However, Cypress will offer these products to new and existing customers with the series name, product name, and ordering part number with the prefix “CY”.

How to Check the Ordering Part Number

1. Go to www.cypress.com/pcn.
2. Enter the keyword (for example, ordering part number) in the **SEARCH PCNS** field and click **Apply**.
3. Click the corresponding title from the search results.
4. Download the Affected Parts List file, which has details of all changes

For More Information

Please contact your local sales office for additional information about Cypress products and solutions.

About Cypress

Cypress is the leader in advanced embedded system solutions for the world's most innovative automotive, industrial, smart home appliances, consumer electronics and medical products. Cypress' microcontrollers, analog ICs, wireless and USB-based connectivity solutions and reliable, high-performance memories help engineers design differentiated products and get them to market first. Cypress is committed to providing customers with the best support and development resources on the planet enabling them to disrupt markets by creating new product categories in record time. To learn more, go to www.cypress.com.

AN204417

MB9BFxxx, S6E2CC, S6E2G Ethernet Driver Manual

The Ethernet Driver is used as a low level driver for FM Family microcontrollers, such as FM3 and FM4 devices. It can be used stand-alone or in conjunction with Cypress's PDL (Peripheral Drivers Library).

Contents

1 Introduction 1
2 Ethernet Driver Structure 1
2.1 Driver File System 1
2.2 Example Files (stand-alone version) 2
2.3 Driver System 2
2.4 Interrupt Flow 3
3 Using the Ethernet Driver 4
3.1 User Settings in emac_user.h 4
3.2 Extra Settings in *emac.c* 13
3.3 Usage Examples 14
4 Ethernet Driver Configuration and Data Types 17
4.1 Preface 17
4.2 Configuration Structure 17
4.3 Link Status enumerated Type 17
4.4 Link Mode enumerated Type 17
5 Ethernet Driver API 18
5.1 Ethernet MAC functions 18
5.2 Ethernet PHY functions 25
6 Additional Information 26
Document History 27
Worldwide Sales and Design Support 28
Products 28
PSoC® Solutions 28
Cypress Developer Community 28
Technical Support 28

1 Introduction

The Ethernet Driver is used as a low level driver for FM Family microcontrollers, such as FM3 and FM4 devices. It can be used stand-alone or in conjunction with Cypress's PDL (Peripheral Drivers Library).

2 Ethernet Driver Structure

This Chapter Explains the Structure of the Ethernet Driver

2.1 Driver File System

The Ethernet driver consists of the following files in its *emac* directory:

emac.c	Ethernet user API
emac.h	Ethernet API header file
emac_user.h	Ethernet driver user setting file
emac_user.c	User provided callback function for specific PHY
EmacDescription.h	More detailed changelog of Ethernet driver, documentation only

2.2 Example Files (stand-alone version)

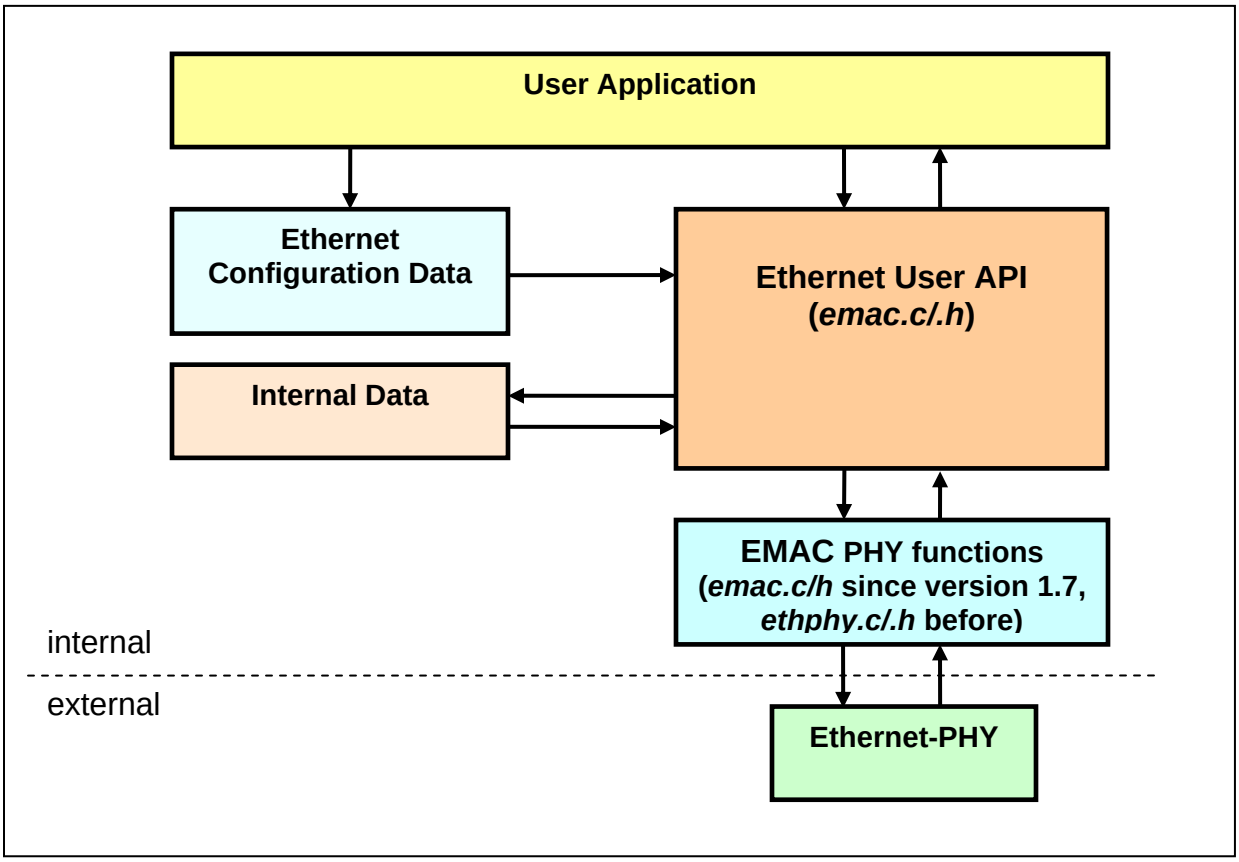
To give an example how to use the Ethernet driver, the software package also contains some files to operate the evaluation board besides the *main.c* module.

Files in library directory	Hardware drivers
scheduler.c ,	Simple scheduler for cyclic functions
scheduler.h	Scheduler header file
stacklessudp.c	Function to create UDP packets without a full-featured TCP/IP stack to send data in a local network for demo purposes.
stacklessudp.h	UDP function header
tasks.c	Tasks to be run by scheduler
tasks.h	Task prototypes

2.3 Driver System

The following graphic illustrates the Ethernet driver for a single instance.

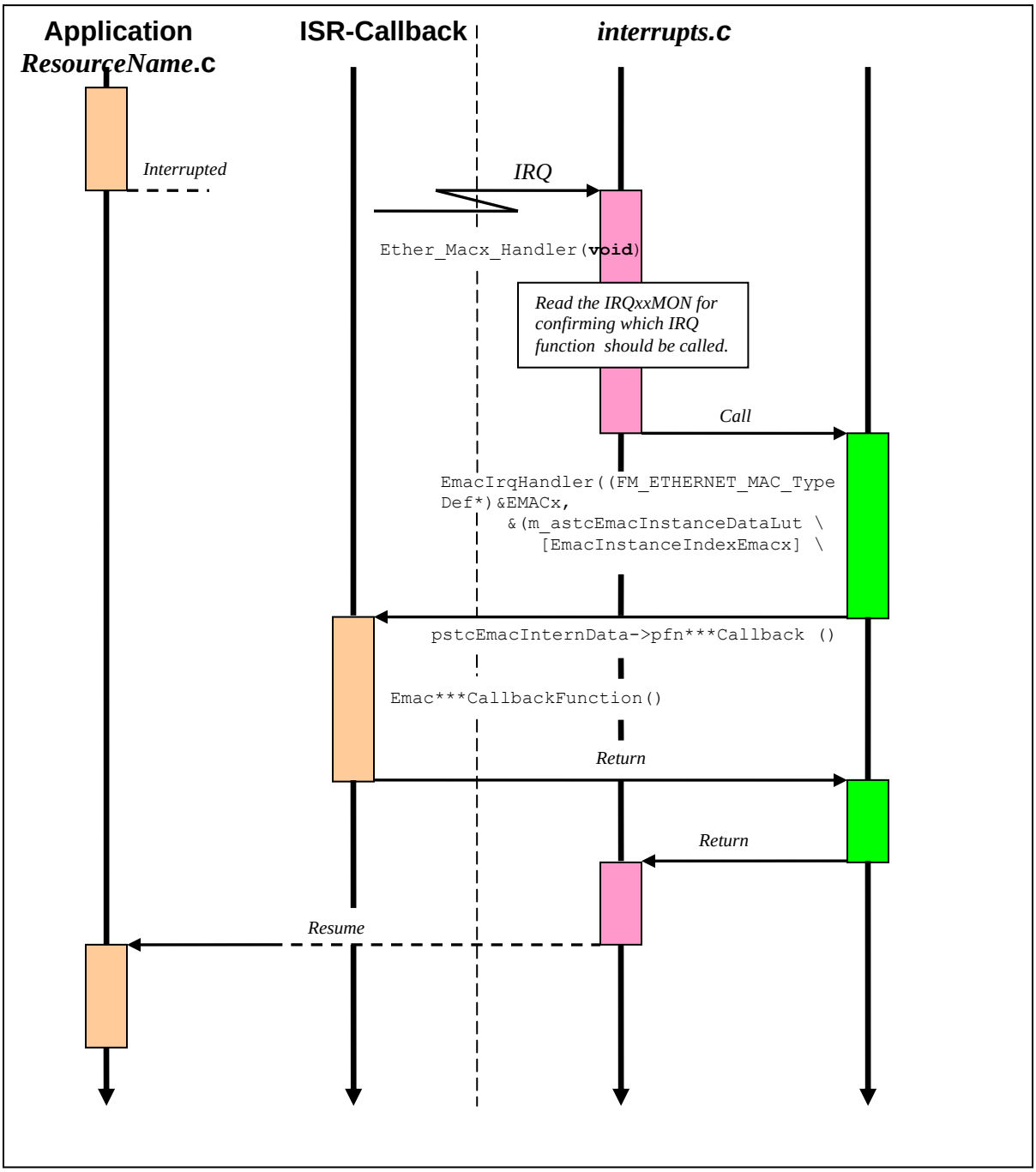
Figure 1. Resource Driver Structure Example



2.4 Interrupt Flow

The following flow shows how interrupts are handled in the L3:

Figure 2. Interrupt and Callback Flow Diagram



3 Using the Ethernet Driver

How to Set-Up the Ethernet Driver

3.1 User Settings in emac_user.h

The file *emac_user.h* allows the user to configure the Ethernet driver at compile time. Later configurations on runtime can be done by the API configuration, which is described in chapter 4.2.

3.1.1 PDL integration

The Ethernet driver may be used with or without PDL (Peripheral Drivers Library) provided by Cypress. Because the driver is configured using symbols such as PDL_ON and PDL_OFF, those symbols must be defined if the driver shall be used stand-alone. This is done with the first option:

```
/**
 ******************************************************************************
 ** \brief Should Cypress Peripheral Drivers Library be used or not
 **
 ** Possible definitions are 1 or 0
 **
 ** 0: Use driver stand-alone
 ** 1: Use driver together with PDL
 **
 ******************************************************************************/
#define EMAC_USE_PDL        0
```

3.1.2 Ethernet Instance Activation

If dual Ethernet is supported by the FM Family microcontroller, the instances can be activated individually by using the `PDL_ON` symbol.

```
/**
 ******************************************************************************
 ** \brief User Defines for EMAC resource enable
 **
 ** Possible definitions are PDL_ON and PDL_OFF.
 ******************************************************************************/
#define PDL_PERIPHERAL_ENABLE_EMAC0  PDL_ON
#define PDL_PERIPHERAL_ENABLE_EMAC1  PDL_ON
```

3.1.3 Ethernet Instance Interrupt Mode

If using Ethernet IRQ support, the instances should be activated by using the `PDL_ON` symbol. In that case, `ETHER_MACx_IRQHandler()` is called when interrupt causes occurred.

```
/**
 ******************************************************************************
 ** \brief Activate IRQ support for Ethernet driver
 **
 ** Possible definitions are PDL_ON and PDL_OFF.
 **
 ******************************************************************************/
#define EMAC_INTERRUPT_MODE     PDL_OFF
```

3.1.4 Ethernet Instance Interrupt Levels

The interrupt level for each instance can be set individually.

```
/**
 ******************************************************************************
 ** \brief User Emac Interrupt level settings
 **
 ** Possible values are 0 (high priority) to 15 (low priority)
 ******************************************************************************/
#define PDL_IRQ_LEVEL_EMAC0             2
#define PDL_IRQ_LEVEL_EMAC1             4
```

3.1.5 Buffer Location (EMAC0 as example)

The user can select whether to use driver internal buffers or provide pointers to own buffers. If set to `PDL_ON` buffers are created in the driver's memory space.

```
/**
 ******************************************************************************
 ** \brief EMAC0 User/Driver Buffer location
 **
 ** PDL_ON:  Driver buffer are used
 ** PDL_OFF: User has to provide buffer memory
 ******************************************************************************/
#define EMAC0_BUFFERS_IN_DRIVERSPACE PDL_ON
```

3.1.6 Transmission/Reception Ring Size (EMAC0 as example)

Inbound and outbound Ethernet frames are stored in ring buffers. This definition determines the number of elements the ring buffers encompass, for transmission and reception direction each.

```
/**

*******************************************************************************
*
 ** \brief EMAC0 Transmission/Reception Ring size

*******************************************************************************
*/
#define EMAC0_TX_RING_SIZE     2
#define EMAC0_RX_RING_SIZE     4
```

3.1.7 Transmission/Reception Buffer Size (EMAC0 as example)

The size of each transmission and reception buffer can be set here. This definition determines the maximum size of an Ethernet frame that can be handled. Note that it must be a multiple of 4 and the default value should be suitable for most applications.

```
/**
 ******************************************************************************
 ** \brief EMAC0 Transmission/Reception Buffer size
 **
 ** Only used if EMAC0_BUFFERS_IN_DRIVERSPACE == PDL_ON
 ** Must be multiple of 4, 13 Bit value, page 159
 ******************************************************************************/
#define EMAC0_TX_BUF_SIZE     1536
#define EMAC0_RX_BUF_SIZE     1536
```

3.1.8 Buffer Settings for EMAC1

The buffer location and Tx/Rx ring and buffer sizes can also be adjusted by defines for EMAC1.

3.1.9 Starter kit setting

For convenience when using a Fujitsu/Spansion/Cypress starter kit, it can be defined here. The correct, board dependent PHY address is selected automatically then.

```
/**
 ******************************************************************************
 ** \brief Support for Cypress starter kits
 **
 ** Activate board specific settings by defining exactly one of these symbols:
 **
 ** SK_FM3_176PMC_ETHERNET from Fujitsu Semiconductor Europe:
 ** STARTERKIT_SK_FM3_176PMC_ETHERNET
 **
 ** If version 1.0 is used, please use additionally:
 ** SK_FM3_176PMC_ETHERNET_BOARDVERSION10
 **
 ** FSSDC-9B618-EVB from Fujitsu Semiconductor Limited Asia:
 ** STARTERKIT_FSSDC9B618EVB
 **
 ******************************************************************************/
#define STARTERKIT_SK_FM3_176PMC_ETHERNET
//#define SK_FM3_176PMC_ETHERNET_BOARDVERSION10
//#define STARTERKIT_FSSDC9B618EVB
//#define STARTERKIT_SK_FM4_216_ETHERNET
```

3.1.10 Addresses of external PHYs

Every MII/RMII compliant Ethernet PHY has a hardware address that usually can be set using pull-up or pull-down resistors. This hard-wired address of the used external PHYs for each instance can be adjusted as follows. This setting varies according to the used board.

Please note that EMAC1_PHY_ADDRESS must be defined even though the microcontroller contains only a single Ethernet MAC. The value must exist but has no effect in this case.

```
/**
 ******************************************************************************
 ** \brief Addresses of the used external PHYs
 ******************************************************************************/
#define EMAC0_PHY_ADDRESS     0
#define EMAC1_PHY_ADDRESS     1
```

3.1.11 EMAC0 and EMAC1 MAC Address

The MAC address or hardware address can be set for each instance. Please note that since driver version 1.4 this definition is only for convenience in the demo and the actual address configuration is performed at runtime.

```
/**
 ******************************************************************************
 ** \brief EMAC0 MAC address
 ******************************************************************************/
#define EMAC0_MAC_ADDRESS0  0x00
#define EMAC0_MAC_ADDRESS1  0x01
#define EMAC0_MAC_ADDRESS2  0x01
#define EMAC0_MAC_ADDRESS3  0x66
#define EMAC0_MAC_ADDRESS4  0x73
#define EMAC0_MAC_ADDRESS5  0x42

/**
 ******************************************************************************
 ** \brief EMAC1 MAC address
 ******************************************************************************/
#define EMAC1_MAC_ADDRESS0  0x00
#define EMAC1_MAC_ADDRESS1  0x01
#define EMAC1_MAC_ADDRESS2  0x01
#define EMAC1_MAC_ADDRESS3  0x66
#define EMAC1_MAC_ADDRESS4  0x73
#define EMAC1_MAC_ADDRESS5  0x38
```

3.1.12 Multicast address

The so-called Multicast address can be set for instance.

```
/**
 ******************************************************************************
 ** \brief Multicast address
 ******************************************************************************/
#define MULTICAST_ADDRESS0  0x00
#define MULTICAST_ADDRESS1  0x00
#define MULTICAST_ADDRESS2  0x00
#define MULTICAST_ADDRESS3  0x00
#define MULTICAST_ADDRESS4  0x00
#define MULTICAST_ADDRESS5  0x00
```

3.1.13 External PHY reset pins

The external PHY must be reset by the microcontroller using GPIO pins. This setting defines which port pin controls the PHY's reset input line. Please provide an address pointer to the GPIO bit-band alias address, which is defined in the device's header file.

The following definition shows an example of how to do this. Here Port Pin P45 is used for PHY0 and P44 for PHY1.

```
/**
 ******************************************************************************
 ** \brief External PHY reset pins
 **
 ** Use GPIO addresses of the bit-band alias definitions of the device header
 ** file.
 ******************************************************************************/
#define EMAC0_PHY_RESET_PIN ((uint32_t*) &bFM3_GPIO_PDOR4_P5)
#define EMAC1_PHY_RESET_PIN ((uint32_t*) &bFM3_GPIO_PDOR4_P4)
```

3.1.14 PLL Clocking Mode

Ethernet capable microcontrollers have an output for 25 and 50 MHz that can be activated here. Please check with your PHY manufacturer's datasheet whether jitter and frequency tolerances allow this setting.

```
/**
 ******************************************************************************
 ** \brief PLL for External PHY
 **
 ** PDL_ON:  PLL is used for PHY clock
 ** PDL_OFF: PLL is not used for PHY clock
 ******************************************************************************/
#define EMAC_ECOUT  PDL_OFF
```

3.1.15 PHY Management Bus Settings

Every MII/RMII has a management bus. To determine, which EMAC instance uses which management bus to the external PHY connected, the following definitions are used. This can be useful if both PHYs shall be connected to the same management bus.

```
/**
 ******************************************************************************
 ** \brief Management Bus Definition
 ******************************************************************************/
#define EMAC0_MANAGEMENTBUS    EMAC0
#define EMAC1_MANAGEMENTBUS    EMAC1
```

3.1.16 RMII PHY Interface Usage

The PHY interface can be set to RMII by using `PDL_ON` for the following definition. Otherwise (`PDL_OFF`), MII is used. Please note that RMII is necessary in order to use both Ethernet interfaces simultaneously.

```
/**
 ******************************************************************************
 ** \brief RMII PHY interface usage
 ******************************************************************************/
#define EMAC_PHYINTERFACE_RMII    PDL_ON
```

3.1.17 Buffer Fragmentation

If `PDL_ON` is used for `EMAC_BUFFERS_NOT_FRAGMENTED` the driver assumes, that all buffers are not fragmented. This option must be set to `PDL_ON`.

```
/**
 ******************************************************************************
 ** \brief EMAC Buffer fragmentation
 **
 ** All buffers are large enough to contain a whole Ethernet frame (MTU size)
 ******************************************************************************/
#define EMAC_BUFFERS_NOT_FRAGMENTED    PDL_ON
```

3.1.18 Checksum Insertion Control

The following controls can be used:

0: Checksum insertion disabled

1: Only IP header checksum calculation and insertion are enabled

2: IP header checksum and payload checksum and insertion are enabled, but pseudo-header checksum is not calculated in hardware

3: IP header checksum and payload checksum and insertion are enabled, and pseudo-header checksum is not calculated in hardware

```
/**
 ******************************************************************************
 ** \brief CIC (Checksum Insertion Control)
 **
 ** These bits control the checksum calculation and insertion. Bit encodings
 ** are as shown below.
 ** - 0  Checksum Insertion Disabled.
 ** - 1  Only IP header checksum calculation and insertion are enabled.
 ** - 2  IP header checksum and payload checksum calculation and insertion are
 **      enabled, but pseudo-header checksum is not calculated in hardware.
 ** - 3  IP Header checksum and payload checksum calculation and insertion are
 **      enabled, and pseudo-header checksum is calculated in hardware.
 ******************************************************************************/
#define EMAC_COE_MODE   3
```

3.1.19 ICMP Checksum Bug Work-Around

If PDL_ON is used for EMAC_ENABLE_ICMP_CHECKSUM_BUG_WORKAROUND, the driver will overwrite the checksum field of a ping reply (ICMP echo reply) header with zeros. This option can be useful if EMAC_COE_MODE is set to 3 and you observe checksum errors while using the ping command toward the embedded system.

```
/**
 ******************************************************************************
 ** \brief Activate work-around for bug in uIP and LwIP TCP/IP stacks
 **
 ** For the Checksum Offload Engine to work correctly, the checksum fields in
 ** the IP header, as well as in the TCP, UDP or ICMP header must be 0.
 **
 ** LwIP up to the recent stable 1.4.0 version, and current uIP have an option
 ** to disable software checksum calculation for IP, UDP and TCP.
 ** A similar option ICMP was (apparently by accident) omitted.
 **
 ** If EMAC_ENABLE_ICMP_CHECKSUM_BUG_WORKAROUND is set to PDL_ON,
 ** every frame to be sent will be checked if it is an ICMP echo reply
 ** and if so, its checksum field cleared.
 **
 ** This slows down every packet transmission and should be fixed in the
 ** original code. This is a known bug and is expected to be fixed
 ** in future releases of LwIP.
 **
 ** For other TCP/IP stacks, this work-around is usually not necessary
 ** and should be disabled.
 **
 ** Cypress tries to retain third-party software in software examples in its
 ** original state as far as possible in order to retain compatibility to the
 ** maximum possible extend.
 **
 ******************************************************************************/
      #define EMAC_ENABLE_ICMP_CHECKSUM_BUG_WORKAROUND   PDL_ON
```

3.1.20 Multicast Address Filter

If PDL_ON is used for EMAC_MULTICAST_FILTER the driver assumes, the user can filter receiving frame in multicast address.

```
/**
 ******************************************************************************
 ** \brief Activate multicast filtering
 **
 ** PDL_ON:  Multicast filtering is used
 ** PDL_OFF: Multicast filtering is not used
 **
 ******************************************************************************/
#define EMAC_MULTICAST_FILTER   PDL_OFF
```

3.1.21 PHY auto-negotiation callback function name

In IEEE 802.3 standard, there is defined a way to start the auto-negotiation process but not how to handle the result. Therefore a specific callback function like this must be provided for every specific used PHY type. The name of this PHY specific callback function must be given here.

```
/**
 ******************************************************************************
 ** \brief Select callback function to read out autonegotiation result from PHY
 **
 ** Provided as a reference are callback functions for PHYs that are assembled
 ** on eval boards SK-FM3-176PMC-ETHERNET and SK-FM4-216-ETHERNET
 *****************************************************************************/
#define EMAC_AUTONEG_FUNCTION  EmacUser_AutoNegotiatePhy_SMSC_LAN8710A_CB
```

3.1.22 PHY auto-negotiation callback function definition

The callback definition must be implemented in file *emac_user.c*. There are two functions provided as a reference, matching the PHYs that are used on Cypress's eval boards:

EmacUser_AutoNegotiatePhy_SMSC_LAN8710A_CB() for SK-FM3-176PMC-ETHERNET and

EmacUser_AutoNegotiatePhy_Micrel_KSZ8091_CB() for SK-FM4-216-ETHERNET

```
#if (EMAC_AUTONEG_FUNCTION == EmacUser_AutoNegotiatePhy_Micrel_KSZ8091_CB)
en_emac_link_mode_t EmacUser_AutoNegotiatePhy_Micrel_KSZ8091_CB(volatile
FM_ETHERNET_MAC_TypeDef* pstcEmac)
{
    en_emac_link_mode_t enLinkMode = EMAC_LinkModeAutonegotiation;
    uint16_t u16PhyRegVal = 0;

// According to data sheet, three LSB bits of register 0x1E indicate
// auto-negotiation result
    // Read out auto-negotiation result from PHY
    u16PhyRegVal = Ethphy_Read(pstcEmac, 0x1E);
    u16PhyRegVal &= 0x7;

    switch (u16PhyRegVal)
    {
        case(1):
            enLinkMode = EMAC_LinkModeHalfDuplex10M;
            break;
        case(2):
            enLinkMode = EMAC_LinkModeHalfDuplex100M;
            break;
        case(5):
            enLinkMode = EMAC_LinkModeFullDuplex10M;
            break;
         case(6):
            enLinkMode = EMAC_LinkModeFullDuplex100M;
            break;
         default:
            //printf("Autonegotiation error!");
            // Setting 100M full duplex, as it is a common setting
            enLinkMode = EMAC_LinkModeFullDuplex100M;
            break;
    }
    return enLinkMode;
}
#endif // (EMAC_AUTONEG_FUNCTION == EmacUser_AutoNegotiatePhy_Micrel_KSZ8091_CB)
```

3.2 Extra Settings in *emac.c*

The user can use various functions by adding following sample code.

3.2.1 Filter setting for Multicast MAC address

If adding following sample code in `Emac_Autonegotiate()`, the user can receive a frame which is filtered in multicast address. In this case, the MAC filters the value that masked bit0-bit7 of multicast address.

```
    pstcEmac->MAR1H = (uint32_t)(((au8GenericMulticast[5]) <<  8)
                                |((au8GenericMulticast[4]) <<  0)
                      );
    pstcEmac->MAR1L = (uint32_t)(((au8GenericMulticast[3]) << 24)
                                |((au8GenericMulticast[2]) << 16)
                                |((au8GenericMulticast[1]) <<  8)
                                |((au8GenericMulticast[0]) <<  0));

    pstcEmac->MAR1H |= (uint32_t)0x20000000;
    pstcEmac->MAR1H |= (uint32 t)0x80000000;
```

3.3 Usage Examples

This section explains the usage of the FM Family Ethernet driver by concrete example.

3.3.1 How to set up an Ethernet interface

If the user wants to use both EMAC0 and EMAC1, initialize them as follows. This setting must be done after the initialization of board pins. If necessary, set callback functions beforehand.

```
// Define configuration structure
stc_emac_config_t stcEmacConfig;
// Initialize configuration structure with zeros
PDL_ZERO_STRUCT(stcEmacConfig);

// Board pins setting for using Ethernet
ConfigureEthernetPins();

// Callback functions setting for EMAC0
stcEmacConfig.pfnRxCallback = EMAC0RxCallbackFunc;
stcEmacConfig.pfnTxCallback = EMAC0TxCallbackFunc;

// Set MAC address for EMAC0 (You can provide your own method here)
stcEmacConfig.au8MacAddress[0] = EMAC0_MAC_ADDRESS0;
stcEmacConfig.au8MacAddress[1] = EMAC0_MAC_ADDRESS1;
stcEmacConfig.au8MacAddress[2] = EMAC0_MAC_ADDRESS2;
stcEmacConfig.au8MacAddress[3] = EMAC0_MAC_ADDRESS3;
stcEmacConfig.au8MacAddress[4] = EMAC0_MAC_ADDRESS4;
stcEmacConfig.au8MacAddress[5] = EMAC0_MAC_ADDRESS5;

// Ethernet MAC 0 initialization
Emac_Init(&EMAC0, &stcEmacConfig);
Emac_Autonegotiate(&EMAC0);

// Define Ethernet MAC 1 Configuration
PDL_ZERO_STRUCT(stcEmacConfig); // Reuse config structure but reinitialize it
stcEmacConfig.au8MacAddress[0] = EMAC1_MAC_ADDRESS0;
stcEmacConfig.au8MacAddress[1] = EMAC1_MAC_ADDRESS1;
stcEmacConfig.au8MacAddress[2] = EMAC1_MAC_ADDRESS2;
stcEmacConfig.au8MacAddress[3] = EMAC1_MAC_ADDRESS3;
stcEmacConfig.au8MacAddress[4] = EMAC1_MAC_ADDRESS4;
stcEmacConfig.au8MacAddress[5] = EMAC1_MAC_ADDRESS5;

// callback functions setting for EMAC1
stcEmacConfig.pfnRxCallback = EMAC1RxCallbackFunc;
stcEmacConfig.pfnTxCallback = EMAC1TxCallbackFunc;

// Ethernet MAC 1 initialization
Emac_Init(&EMAC1, &stcEmacConfig);
Emac_Autonegotiate(&EMAC1);
```

It is recommended to call `Emac_Autonegotiate()` regularly to establish the Ethernet link and react to changed connection states. If the link already exists, calling this function does no harm.

3.3.2 How to Send a Frame

The user can transmit a frame by calling `Emac_TxFrame()` after initialization. This function has three parameters;

1st a pointer, which selects the EMAC (0 or 1)

2nd the address of the buffer to be sent, and

3rd the number of bytes to be sent by this function.

The buffer must contain the protocol headers and payload beforehand. The following example sets up (TxBufferUDPFill()) and transmits (Emac_TxFrame()) a UDP frame.

```
uint8_t ua8EthBuf[1500];
uint8_t ua8Payload[] = "Hello World!”;

// creating an Ethernet frame with IP and UDP header
TxBufferUDPFill(ua8EthBuf, ua8Payload, sizeof(ua8Payload));
ua8EthBuf[42 + sizeof(ua8Payload) - 3] = (u8NumberOfCalls + '0');

// transmit the frame
Emac_TxFrame(&EMAC0, ua8EthBuf, 42 + sizeof(ua8Payload));
```

3.3.3 How to Receive a Frame

If interrupt mode is activated, the user can receive a frame by using `memcpy()` after calling `Emac_RxFrame_GetBufPtr()`. Afterwards, It is necessary to set next descriptor by calling `Emac_RxFrame_ReleaseBuf()`.
If interrupt mode is not activated, the user can receive a frame by calling `Emac_Rx()`.

```
#if (EMAC_INTERRUPT_MODE == PDL_ON)
        u8pbuffer =(uint8_t *) Emac_RxFrame_GetBufPtr(pstcEmac);
        if (NULL != u8pbuffer)
        {
          memcpy((uint8_t*)p->payload, buffer, p->len);
        }
        Emac_RxFrame_ReleaseBuf(pstcEmac);
#else
        p->len =  Emac_RxFrame(pstcEmac,p->payload);
#endif
```

3.3.4 Interrupt Operation

For using interrupt mode, the user must set EMAC_INTERRUPT_MODE to `PDL_ON`.

```
#define EMAC_INTERRUPT_MODE     PDL_ON
```

Please add a necessary interrupt in `Emac_InitIrq()`. By the default setting the abnormal interrupt is not admitted.

```
pstcEmac->stcIER.NIE = 1;
pstcEmac->stcIER.RIE = 1;
pstcEmac->stcIER.TIE = 1;
```

When an interrupt occurs, `EmacIrqHandler()` is called. Please change the following code in `Emac_IrqHandler()` as needed. By the default code, in the case of transmission or reception completion, a callback function is called.

```
if(pstcEmac->stcSR.RI != 0)
{
    pstcEmac->stcSR.RI = 1;
    if(pstcEmacInternData->pfnRxCallback != NULL)
    {
        pstcEmacInternData->pfnRxCallback();
    }
}
if(pstcEmac->stcSR.TI != 0)
{
    pstcEmac->stcSR.TI = 1;
    if(pstcEmacInternData->pfnTxCallback != NULL)
    {
        pstcEmacInternData->pfnTxCallback();
    }
}
```

4 Ethernet Driver Configuration and Data Types

The Data Structure of the Ethernet Instance Configuration

4.1 Preface

Besides the static compiler definitions the user can configure dynamically an Ethernet instance by using an individual configuration. The data contents of this configuration are used by the initialization function to set the hardware and internal data. Afterwards this configuration is no longer needed.

4.2 Configuration Structure

The following data is used for the Ethernet driver's configuration structure `stc_emac_config_t`.

Type	Field	Possible Values	Description
boolean_t	bPromiscuousMode	PDL_ON, PDL_OFF	If enabled, interface will receive all frames, including those not addressed to it
func_ptr_t	pfnRxCallback	-	Function pointer to user's reception callback function
func_ptr_t	pfnTxCallback	-	Function pointer to user's transmission callback function
func_ptr_t	pfnErrorCallback	-	Function pointer to user's error callback function
uint8_t array	au8MacAddress[6]	-	Ethernet Hardware Address, also known as MAC Address

4.3 Link Status enumerated Type

The Ethernet driver uses an own enumerated type for the link status. It is defined in `en_emac_link_t` as follows:

Enumerator	Description
EMAC_LinkStatusLinkDown	No connection established
EMAC_LinkStatusLinkUp	Connection established, ready for data exchange, MAC has been configured
EMAC_LinkStatusAutonegotiationInProgress	Auto-negotiation started and in progress
EMAC_LinkStatusAutonegotiationSuccessful	Auto-negotiation agreement with remote nodes completed
EMAC_LinkStatusAutonegotiationNotSupported	Remote node does not support auto-negotiation
EMAC_LinkStatusInvalidParameter	One or more user configuration illegal
EMAC_LinkStatusUnknownError	In indeterminable error occurred

4.4 Link Mode enumerated Type

Also the link mode is defined as an enumerated type with the name `en_emac_link_mode_t`. Its content is:

Enumerator	Description
EMAC_LinkModeHalfDuplex10M	Half-Duplex Mode 10 MBit/s
EMAC_LinkModeFullDuplex10M	Full-Duplex Mode 10 MBit/s
EMAC_LinkModeHalfDuplex100M	Half-Duplex Mode 100 MBit/s
EMAC_LinkModeFullDuplex100M	Full-Duplex Mode 100 MBit/s

5 Ethernet Driver API

Detailed Description of the Ethernet driver API and example code

5.1 Ethernet MAC functions

This chapter explains the user API functions of the Ethernet driver in detail.

5.1.1 Emac_Init()

This function initializes and Ethernet instance. `pstcEmac` must be either `(FM_ETHERNET_MAC_TypeDef*)&EMAC0` or `(FM_ETHERNET_MAC_TypeDef*)&EMAC1`. `pstcConfig` is just the address to the user configuration, e.g. `&stcEmacConfig`.

Prototype	
`en_result_t Emac_Init( volatile FM_ETHERNET_MAC_TypeDef* pstcEmac, stc_emac_config_t* pstcConfig )`	
Parameter Name	**Description**
`[in] pstcEmac`	Ethernet macro instance pointer
`[in] pstcConfig`	Ethernet configuration parameters
Return Values	**Description**
Ok	Channel initialized

ErrorInvalidParameter	If one of the following conditions is met:
	`pstcEmac == NULL`
	`pstcEmacInternData == NULL`
	`(invalid or disabled Ethernet unit)`

5.1.2 Emac_DeInit()

This function de-initializes and Ethernet instance.

Prototype	
`en_result_t Emac_DeInit( volatile FM_ETHERNET_MAC_TypeDef* pstcEmac )`	
Parameter Name	**Description**
`[in] pstcEmac`	Ethernet macro instance pointer
Return Values	**Description**
`Ok`	Channel disabled
`ErrorInvalidParameter`	If the following condition is met: `pstcEmac == NULL`

This function deinitializes Ethernet IRQs, puts EMAC into reset condition, and erases internal data.

Please note that MAC1 can only work if MAC0 is sourced with a clock signal. Therefore, Emac_DeInit(EMAC0) will put EMAC0 into reset state in any case, but turn off clock supply only if EMAC1 has been deactive. This can be done either by calling Emac_DeInit(EMAC1) before or by never calling Emac_Init(EMAC1) before. Therefore, if you want to deinitialize both Ethernet channels, call Emac_DeInit(EMAC1) first and Emac_DeInit(EMAC0) secondly.

5.1.3 Emac_TxFrame()

This function is used to send an Ethernet frame.

Prototype	
`en_result_t Emac_TxFrame( volatile FM_ETHERNET_MAC_TypeDef* pstcEmac, uint8_t* pu8Buf, uint16_t u16Len )`	
Parameter Name	**Description**
`[in] pstcEmac`	Ethernet macro instance pointer
`[in] pu8Buf`	Pointer to a byte array buffer to be sent
`[in] u16Len`	Length of the byte array buffer, number of bytes to be sent
Return Values	**Description**
`Ok`	Transmission done
`ErrorInvalidParameter`	If the following condition is met: `pstcEmac == NULL`

5.1.4 Emac_GetRxFrameLength()

This function returns the length of a received frame.

Prototype	
`uint16_t Emac_GetRxFrameLength( volatile FM_ETHERNET_MAC_TypeDef* pstcEmac )`	
Parameter Name	**Description**
`[in] pstcEmac`	Ethernet macro instance pointer
Return Values	**Description**
`uint16_t`	Length of the received frame or `0` if nothing was received or `pstcEmac == NULL`

5.1.5 Emac_RxFrame()

Copies a received Ethernet frame into a buffer provided by the user application.

Prototype	
`uint16_t Emac_RxFrame( volatile FM_ETHERNET_MAC_TypeDef* pstcEmac, uint8_t* pu8Buf )`	
Parameter Name	**Description**
`[in] pstcEmac`	Ethernet macro instance pointer
`[in] pu8Buf`	Pointer to a byte array buffer that will be overwritten with contents of received Ethernet frame
Return Values	**Description**
`uint16_t`	Length of the received frame or `0` if nothing was received or `pstcEmac == NULL`

5.1.6 Emac_GetLinkStatus()

This function returns the recent link status. It uses the type `en_emac_link_status_t` described in chapter 4.3.

Prototype	
`en_emac_link_status_t Emac_GetLinkStatus( volatile FM_ETHERNET_MAC_TypeDef* pstcEmac )`	
Parameter Name	**Description**
`[in] pstcEmac`	Ethernet macro instance pointer
Return Values	**Description**

`en_emac_link_status_t`	Returns:
	`EMAC_LinkStatusLinkDown`
	`EMAC_LinkStatusLinkUp`
	`EMAC_LinkStatusAutonegotiationInProgress`
	`EMAC_LinkStatusAutonegotiationSuccessful`
	`EMAC_LinkStatusAutonegotiationNotSupported`
	`EMAC_LinkStatusInvalidParameter`
	`EMAC_LinkStatusUnknownError`
	
	See also description in chapter 4.3.

5.1.7 Emac_GetLinkMode()

This function returns the recent link mode. It uses the type `en_emac_link_mode_t` described in chapter 4.4.

Prototype	
`en_emac_link_mode_t Emac_GetLinkMode( volatile FM_ETHERNET_MAC_TypeDef* pstcEmac )`	
Parameter Name	**Description**
`[in] pstcEmac`	Ethernet macro instance pointer
Return Values	**Description**

`en_emac_link_mode_t`	Returns:
	`EMAC_LinkModeAutonegotiation`
	`EMAC_LinkModeHalfDuplex10M`
	`EMAC_LinkModeFullDuplex10M`
	`EMAC_LinkModeHalfDuplex100M`
	`EMAC_LinkModeFullDuplex100M`
	
	See also description in chapter 4.4.

5.1.8 Emac_SetLinkMode()

This function sets the link mode. It uses the type `en_emac_link_mode_t` described in chapter 4.4 as an argument and return value

Prototype	
`en_emac_link_mode_t Emac_SetLinkMode( volatile FM_ETHERNET_MAC_TypeDef* pstcEmac, en_emac_link_mode_t enLinkMode )`	
Parameter Name	**Description**
`[in] pstcEmac`	Ethernet macro instance pointer
`[in] enLinkMode`	Desired link mode. See below at Return Values description for enumerated value list.
Return Values	**Description**

`en_emac_link_mode_t`	Returns successful mode:
	`EMAC_LinkModeAutonegotiation`
	`EMAC_LinkModeHalfDuplex10M`
	`EMAC_LinkModeFullDuplex10M`
	`EMAC_LinkModeHalfDuplex100M`
	`EMAC_LinkModeFullDuplex100M`
	
	See also description in chapter 4.4.

5.1.9 Emac_Autonegotiate()

This function performs the auto-negotiation of the Ethernet macro. It should be called periodically.

Prototype	
`en_emac_link_status_t Emac_Autonegotiate( volatile FM_ETHERNET_MAC_TypeDef* pstcEmac )`	
Parameter Name	**Description**
`[in] pstcEmac`	Ethernet macro instance pointer
Return Values	**Description**

`en_emac_link_status_t`	Returns:
	`EMAC_LinkStatusLinkDown`
	`EMAC_LinkStatusLinkUp`
	`EMAC_LinkStatusAutonegotiationInProgress`
	`EMAC_LinkStatusAutonegotiationSuccessful`
	`EMAC_LinkStatusAutonegotiationNotSupported`
	`EMAC_LinkStatusInvalidParameter`
	`EMAC_LinkStatusUnknownError`
	
	See also description in chapter 4.3.

5.1.10 Emac_SetDescToTxBuf()

This function is only available if the buffer locations are defined in Ethernet driver space by user via EMAC0_BUFFERS_IN_DRIVERSPACE or EMAC1_BUFFERS_IN_DRIVERSPACE.

Prototype
en_result_t Emac_SetDescToTxBuf(volatile FM_ETHERNET_MAC_TypeDef* pstcEmac, uint16_t u16DescNumber, uint8_t * pu8UserBuf, uint16_t u16BufLength)

Parameter Name	Description
[in] pstcEmac	Ethernet macro instance pointer
[in] u16DescNumber	DMA descriptor number
[in] pu8UserBuf	Pointer to a byte array buffer
[in] u16BufLength	Length of the byte array buffer
Return Values	**Description**
Ok	Descriptor set
ErrorInvalidParameter	If the following condition is met: pstcEmac == NULL

5.1.11 Emac_SetDescToRxBuf()

This function is only available if the buffer locations are defined in Ethernet driver space by user via EMAC0_BUFFERS_IN_DRIVERSPACE or EMAC1_BUFFERS_IN_DRIVERSPACE.

Prototype
en_result_t Emac_SetDescToRxBuf(volatile FM_ETHERNET_MAC_TypeDef* pstcEmac, uint16_t u16DescNumber, uint8_t * pu8UserBuf, uint16_t u16BufLength)

Parameter Name	Description
[in] pstcEmac	Ethernet macro instance pointer
[in] u16DescNumber	DMA descriptor number
[in] pu8UserBuf	Pointer to a byte array buffer
[in] u16BufLength	Length of the byte array buffer
Return Values	**Description**
Ok	Descriptor set
ErrorInvalidParameter	If the following condition is met: pstcEmac == NULL

5.1.12 Emac_SetEcout()

This function allows the PLL clock to be used. If `EMAC_ECOUT` is set to `PDL_ON`, the E_COUT pin is configured as output and USB/Ethernet Clock activated.

Prototype	
`en_result_t Emac_SetEcout( void )`	
Parameter Name	**Description**
`void`	-
Return Values	**Description**
`Ok`	Completed

5.1.13 Emac_RxFrame_GetBufPtr()

This function returns a pointer to the buffer containing the received frame. It allows the usage of an optimized rx routine to save copy operations. This function is only available if interrupt mode is enabled. (Please refer to chapter 3.3.3)

Prototype	
`void* Emac_RxFrame_GetBufPtr( volatile FM_ETHERNET_MAC_TypeDef* pstcEmac )`	
Parameter Name	**Description**
`[in] pstcEmac`	Ethernet macro instance pointer
Return Values	**Description**
`void *`	Pointer of the received frame buffer or `0` if nothing was received or `pstcEmac == NULL`

5.1.14 Emac_RxFrame_ReleaseBuf()

This function sets next descriptor. It can be used to release a buffer after its contents have been processed by an optimized rx routine. This function is only available if interrupt mode is enabled. (Please refer capter3.3.3)

Prototype	
`void Emac_RxFrame_ReleaseBuf( volatile FM_ETHERNET_MAC_TypeDef* pstcEmac )`	
Parameter Name	**Description**
`[in] pstcEmac`	Ethernet macro instance pointer
Return Values	**Description**
`void`	-

5.2 Ethernet PHY functions

The Ethernet driver provides functions to reset a connected Ethernet PHY, to write data to it via MII/RMII's station management interface, and to read from it. Those functions can be regarded as internal and safely ignored however because the functions beginning with `Emac_` described in section 5.1 take care of them.

5.2.1 Ethphy_Read()

This function reads out a value from the PHY.

Prototype	
`uint16_t Ethphy_Read( volatile FM_ETHERNET_MAC_TypeDef* pstcEmac, uint8_t u8PhyReg );`	
Parameter Name	**Description**
`[in] pstcEmac`	Ethernet macro instance pointer
`[in] u8PhyReg`	Register address to be read out
Return Values	**Description**
`uint16_t`	Value of designated register

5.2.2 Ethphy_Write()

This function writes a value into the PHY.

Prototype	
`en_result_t Ethphy_Write( volatile FM_ETHERNET_MAC_TypeDef* pstcEmac, uint8_t u8PhyReg, uint16_t u16Val )`	
Parameter Name	**Description**
`[in] pstcEmac`	Ethernet macro instance pointer
`[in] u8PhyReg`	Register address to be overwritten
`[in] u16Val`	Value to be written into the PHY
Return Values	**Description**
`OK`	Value has been written into the PHY
`ErrorInvalidParameter`	pstcEmac invalid

5.2.3 Ethphy_Reset()

This function resets the PHY via the GPIO pin defined by EMAC0_PHY_RESET_PIN or EMAC1_PHY_RESET_PIN in *emac_user.h*.

Prototype	
`uint16_t Ethphy_Read( volatile FM_ETHERNET_MAC_TypeDef* pstcEmac );`	
Parameter Name	**Description**
`[in] pstcEmac`	Ethernet macro instance pointer
Return Values	**Description**
`OK`	PHY reset sequence has been conducted
`ErrorInvalidParameter`	pstcEmac invalid

6 Additional Information

More information about Cypress's Microcontrollers can be found on the Internet at

http://www.cypress.com/cypress-microcontrollers

The software examples related to this application note are:

mb9bfxxx_ethernet_driver

s6e2cc_ethernet_driver

Document History

Document Title: AN204417 - MB9BFxxx, S6E2CC, S6E2G Ethernet Driver Manual

Document Number: 002-04417

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	CHNO	01/08/2013	V1.0; CNo, YMo; 1st public version
			01/31/2014	V2.0; CNo Reflecting changes of driver API
*A	5040200	CHNO	01/07/2016	Converted Spansion Application Note “AN706-00059-2v0-E” to Cypress format
*B	5875962	AESATMP8	09/07/2017	Updated logo and Copyright.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2013-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.