

# QNX EMAC for Third-Party PHY/Switch

## Integration Guide

80-PG469-20 Rev. AA

March 8, 2023

**Confidential – Qualcomm Technologies, Inc. and/or its affiliated companies – May Contain Trade Secrets**

**NO PUBLIC DISCLOSURE PERMITTED:** Please report postings of this document on public servers or websites to:  
[DocCtrlAgent@qualcomm.com](mailto:DocCtrlAgent@qualcomm.com).

**Confidential Distribution:** Use or distribution of this item, in whole or in part, is prohibited except as expressly permitted by written agreement(s) and/or terms with Qualcomm Incorporated and/or its subsidiaries.

Not to be used, copied, reproduced, or modified in whole or in part, nor its contents revealed in any manner to others without the express written permission of Qualcomm Technologies, Inc.

All Qualcomm products mentioned herein are products of Qualcomm Technologies, Inc. and/or its subsidiaries.

Qualcomm is a trademark or registered trademark of Qualcomm Incorporated. Other product and brand names may be trademarks or registered trademarks of their respective owners.

This technical data may be subject to U.S. and international export, re-export, or transfer ("export") laws. Diversion contrary to U.S. and international law is strictly prohibited.

Qualcomm Technologies, Inc.  
5775 Morehouse Drive  
San Diego, CA 92121  
U.S.A.

## Revision history

Revision	Date	Description
AA	March 2023	Initial release

Qualcomm  
Confidential - May Contain Trade Secrets  
2024-01-10 11:34:07 GMT  
mintukumar.patel@harman.com

# Contents

---

<b>1 Introduction.....</b>	<b>5</b>
1.1 Purpose .....	5
1.2 Disclaimer .....	5
1.3 Conventions .....	5
1.4 Technical assistance.....	5
<b>2 EMAC Hardware.....</b>	<b>6</b>
<b>3 PHY Wrapper APIs.....</b>	<b>7</b>
3.1 Phy_Register_Extended() .....	7
3.2 Phy_Register_Extended_CL45().....	8
3.3 Phy_FindPhy() .....	9
3.4 Phy_InitPhy() .....	9
3.5 Phy_InitPhy_CL45().....	10
3.6 Phy_PowerupPhy() .....	10
3.7 Phy_PowerupPhy_CL45().....	11
3.8 Phy_AutoNegotiate().....	11
3.9 Phy_EnableMonitor() .....	12
3.10 Phy_DisableMonitor().....	12
3.11 Phy_GetActiveMedia() .....	13
3.12 Phy_MonitorPhy() .....	13
3.13 Phy_PowerdownPhy().....	14
3.14 Phy_DeRegister().....	14
<b>A References.....</b>	<b>15</b>
A.1 Acronyms and terms.....	15

## Figures

Figure 2-1 Qualcomm EMAC driver.....	6
--------------------------------------	---

Qualcomm  
Confidential - May Contain Trade Secrets  
2024-01-10 11:34:07 GMT  
mintukumar.patel@harman.com

# 1 Introduction

---

## 1.1 Purpose

This document helps Qualcomm Technologies, Inc. (QTI) chipset licensees to integrate QTI Ethernet media access controller (EMAC) solution on QNX with a third-party physical layer (PHY) or switch. It describes the factors to consider to integrate the QTI EMAC solution with a third-party physical layer (PHY) or switch.

## 1.2 Disclaimer

OEMs that use third-party PHYs and controllers must configure and validate those third-party PHYs and controllers..

## 1.3 Conventions

Function declarations, function names, type declarations, attributes, and code samples appear in a different font, for example, `cp armcc armcpp`.

Code variables appear in angle brackets, for example, `<number>`.

Commands to be entered appear in a different font, for example, `copy a:*. * b:`.

Button and key names appear in bold font, for example, click **Save** or press **Enter**.

Shading indicates content that has been added or changed in this revision of the document.

## 1.4 Technical assistance

For assistance or clarification on information in this document, open a technical support case at <https://support.qualcomm.com/>.

You will need to register for a Qualcomm ID account and your company must have support enabled to access our Case system.

Other systems and support resources are listed on <https://qualcomm.com/support>.

If you need further assistance, you can send an email to [qualcomm.support@qti.qualcomm.com](mailto:qualcomm.support@qti.qualcomm.com).

## 2 EMAC Hardware

EMAC is an integrated Ethernet controller from QTI that facilitates Qualcomm® modem chips to communicate with video, audio, precision time protocol (PTP) peripheral devices, and best effort (BE) traffic over RJ45 cables.

- Qualcomm custom PHY driver library
- Wrapper over QNX MII library.
- Default Qualcomm PHY library implementation of these APIs is one-to-one mapping to QNX MII APIs
- Vendors can change the default implementation of this library to support different PHY, such as clause 45 support for basic PHY registers, and support for EMAC connected to a switch.
- The custom PHY library should call into EMAC driver provided read and write callbacks to read and write into corresponding PHY registers using EMAC registers.

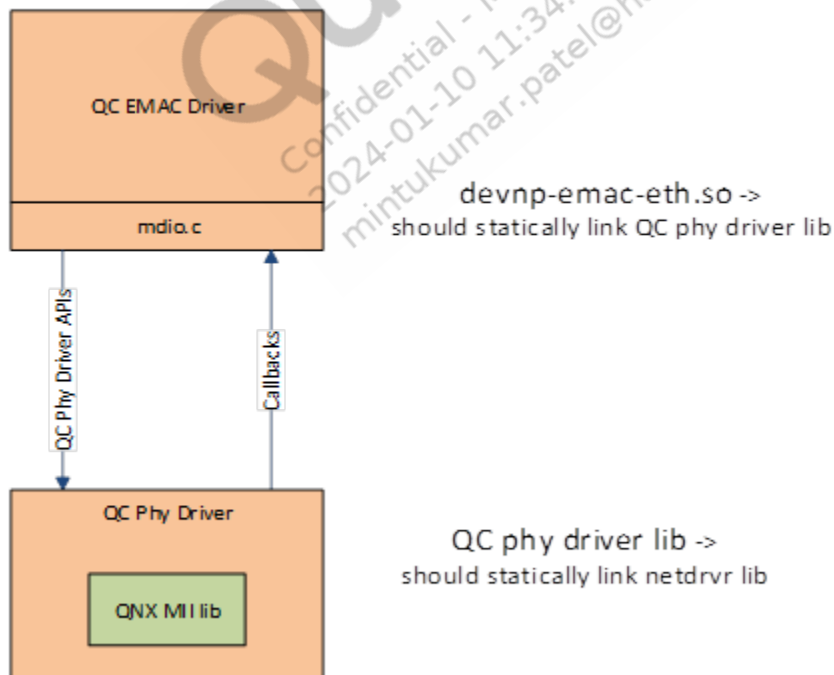


Figure 2-1 Qualcomm EMAC driver

## 3 PHY Wrapper APIs

---

### 3.1 Phy\_Register\_Extended()

Registers with the MII management library.

Determines if a PHY with an address of PhyAddr exists.

#### Prototype

```
int Phy_Register_Extended(void *handle,
                          MDIWriteFunc write,
                          MDIReadFunc read,
                          MDICallBack callback,
                          mdi_t **mdi,
                          struct sigevent *event,
                          int priority,
                          int callback_interval,
                          int phy_type);
```

#### Parameters

in	handle	Handle that the library passes to each driver callback.
in	write	Pointer to a function that writes to a PHY register through the MAC device..
in	read	Pointer to a function that reads a PHY register through the MAC device.
in	callback	Pointer to a function that the library calls when the link state changes.
in	mdi	Void pointer to an mdi_t structure that the library initializes.
in	event	sigevent pointer for when the driver receives link monitor pulses.
in	priority	Priority of the link monitor pulses.
in	callback_interval	Frequency, in seconds, of link monitor pulses.

#### Returns

1 -- Success.

## 3.2 Phy\_Register\_Extended\_CL45()

For CL\_45 supported PHY, registers with the MII management library  
Determines if a PHY with an address of PhyAddr exists.

### Prototype

```
int Phy_Register_Extended_CL45(void *handle,
                               MDI_CL45_WriteFunc write,
                               MDI_CL45_ReadFunc read,
                               MDIWriteFuncCl45 mdi_writecl45,
                               MDIReadFuncCl45 mdi_readcl45,
                               MDICallback callback,
                               mdi_t **mdi,
                               struct sigevent *event,
                               int priority,
                               int callback_interval,
                               int phy_type)
```

### Parameters

in	handle	Handle that the library passes to each driver callback.
in	write	Pointer to a function that writes to a PHY register through the MAC device.
in	read	Pointer to a function that reads a PHY register through the MAC device.
in	mdi_writecl45	Pointer to a function that writes to a CL_45 via clause 22 supported PHY register through the MAC device.
in	mdi_readcl45	Pointer to a function that reads to a CL_45 via clause 22 supported PHY register through the MAC device.
in	callback	Pointer to a function that the library calls when the link state changes.
in	mdi	Void pointer to an mdi_t structure that the library initializes.
in	event	sigevent pointer for when the driver receives link monitor pulses.
in	priority	Priority of the link monitor pulses.
in	callback_interval	Frequency, in seconds, of link monitor pulses.

### Returns

1 -- Success.



### 3.3 Phy\_FindPhy()

Determines if a PHY with an address of PhyAddr exists.

#### Prototype

```
int Phy_FindPhy(void *mdi,
               int PhyAddr,
               int phy_type);
```

#### Parameters

in	mdi	Pointer to the mdi_t structure.
in	PhyAddr	Physical address of the physical layer device.
in	Phy_type	Type of the PHY.

#### Returns

1 -- Success.

### 3.4 Phy\_InitPhy()

Initializes the PHY whose address is PhyAddr.

#### Prototype

```
int Phy_InitPhy(void *mdi,
               int PhyAddr,
               int phy_type);
```

#### Parameters

in	mdi	Pointer to the mdi_t structure.
in	PhyAddr	Physical address of the physical layer device.
in	Phy_type	Type of the PHY.

#### Returns

1 -- Success.

### 3.5 Phy\_InitPhy\_CL45()

For CL\_45 supported PHY, initializes the PHY whose address is PhyAddr.

#### Prototype

```
int Phy_InitPhy_CL45(uint8_t PhyAddr);
```

#### Parameters

in	PhyAddr	Physical address of the physical layer device.
----	---------	--

#### Returns

1 -- Success.

### 3.6 Phy\_PowerupPhy()

Powers down the PHY whose address is PhyAddr.

#### Prototype

```
int Phy_PowerupPhy(void *mdi,
                   int PhyAddr,
                   int phy_type);
```

#### Parameters

in	mdi	Pointer to the mdi_t structure.
in	PhyAddr	Physical address of the physical layer device.
in	phy_type	Type of the PHY.

#### Returns

1 -- Success.

### 3.7 Phy\_PowerupPhy\_CL45()

Powers down the PHY whose address is PhyAddr.

#### Prototype

```
int Phy_PowerupPhy_CL45(int PhyAddr,
int phy_type);
```

#### Parameters

in	PhyAddr	Physical address of the physical layer device.
in	phy_type	Type of the PHY.

#### Returns

1 -- Success.

### 3.8 Phy\_AutoNegotiate()

Initiates the autonegotiation process between the PHY and its link partner.

#### Prototype

```
int Phy_AutoNegotiate(void *mdi,
int PhyAddr,
int phy_type,
int Timeout);
```

#### Parameters

in	mdi	Pointer to the mdi_t structure.
in	PhyAddr	Physical address of the physical layer device.
in	phy_type	Type of the PHY.

#### Returns

1 -- Success.

### 3.9 Phy\_EnableMonitor()

Allows the link monitor to communicate with the PHY and call the link state change of the driver when appropriate.

#### Prototype

```
int Phy_EnableMonitor(void *mdi,
                     int phy_type,
                     int LDownTest);
```

#### Parameters

in	mdi	Pointer to the mdi_t structure.
in	phy_type	Type of the PHY.
in	LDownTest	Tests for the link down state.

#### Returns

1 -- Success.

### 3.10 Phy\_DisableMonitor()

Prevents a change callback or a new link.

The Phy\_MDI\_DisableMonitor() function prevents MDI\_DisableMonitorPhy() from calling the callback for the link-down status change of the driver, or from attempting to establish a new link when no link is detected.

#### Prototype

```
void Phy_DisableMonitor(void *mdi,
                       int phy_type);
```

#### Parameters

in	mdi	Pointer to the mdi_t structure.
in	phy_type	Type of the PHY.

### 3.11 Phy\_GetActiveMedia()

Stores the active media type for PhyAddr.

Function prevents MDI\_MonitorPhy() from calling the callback for the driver's link-down status change, or from attempting to establish a new link when no link is detected.

#### Prototype

```
int Phy_GetActiveMedia(void *mdi,
                      int PhyAddr,
                      int phy_type,
                      int *Media);
```

#### Parameters

in	mdi	Void pointer to the mdi_t structure.
in	PhyAddr	Physical address of the physical layer device.
in	phy_type	Type of the PHY.
in	Media	Pointer to the media-type.

#### Returns

1 -- Success.

### 3.12 Phy\_MonitorPhy()

Check the status of all PHYs

The driver can call this function when it receives a link monitor pulse or a link event interrupt. The MDI\_MonitorPhy() function checks the status of all PHYs that were initialized with MDI\_InitPhy().

#### Prototype

```
void Phy_MonitorPhy(void *mdi,
                   int phy_type,
                   int phy_addr,
                   int current_link_state);
```

#### Parameters

in	mdi	Void pointer to the mdi_t structure.
in	phy_type	Type of the PHY.
in	phy_addr	Physical address of the physical layer device.

### 3.13 Phy\_PowerdownPhy()

Powers down the PHY whose address is PhyAddr.

#### Prototype

```
void Phy_PowerdownPhy(void *mdi,
                      int PhyAddr,
                      int phy_type);
```

#### Parameters

in	mdi	Void pointer to the mdi_t structure.
in	PhyAddr	Physical address of the physical layer device.
in	phy_type	Type of the PHY.

### 3.14 Phy\_DeRegister()

Deregisters from the MII management, invalidates the mdi\_t pointer, and frees any resources.

#### Prototype

```
void Phy_DeRegister(mdi_t **mdi);
```

#### Parameters

in	mdi	Void pointer to the mdi_t structure.
----	-----	--------------------------------------

# A References

---

## A.1 Acronyms and terms

Acronym or term	Definition
BE	Best effort
EMAC	Ethernet media access controller
PHY	Physical layer
PTP	Precision time protocol
QTI	Qualcomm Technologies, Inc.