Qualcomm Technologies, Inc.

QNX EMAC for Third-Party PHY/Switch

Integration Guide

80-PG469-20 Rev. AB

September 22, 2023

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

Revision history

Revision	Date	Description
AA	March 2023	Initial release
AB	September 2023	Add Chapter 4 *EMAC Switch Integration*

Contents

Figures

1 Introduction

1.1 Purpose

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

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.

2 EMAC hardware

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

The EMAC solution includes the following:

- Qualcomm custom PHY driver library
- Wrapper over QNX MII library.
- Default Qualcomm PHY library implementation of APIs is one-to-one mapping to QNX MII APIs
- Vendors can change the default library implementation 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.

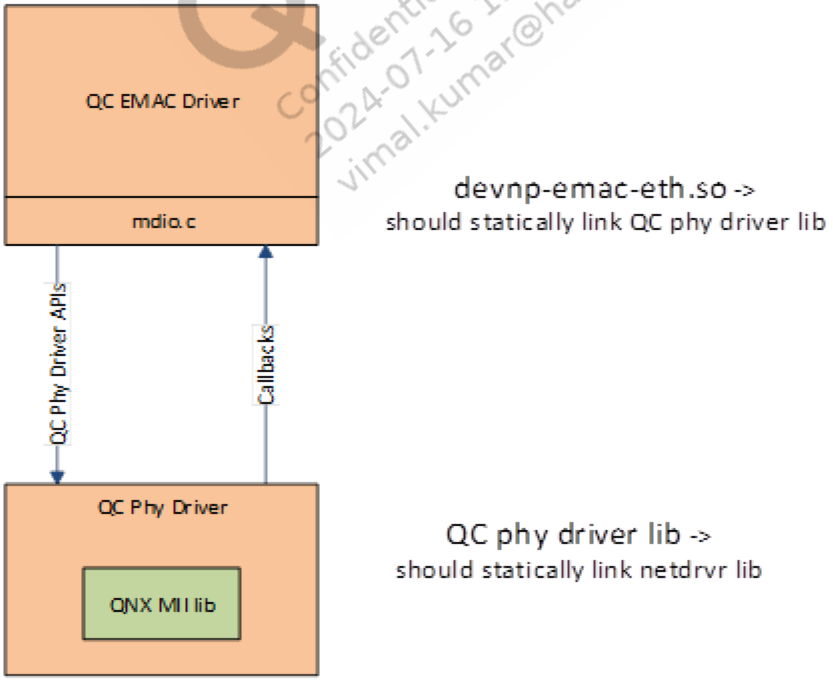

Figure 2-1 Qualcomm EMAC driver

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 with the PhyAddr address.

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 with the 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 auto negotiation 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.

Prevents MDI_MonitorPhy() 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

```
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.

4 EMAC switch integration

It is not necessary to implement the PHY wrapper calls for the switch attach use case; however, the following change is required.

1. In the emac_public.h header file, add a new SWITCH type, for example “RTL9068AB”, as shown in the following example:

   ```
   typedef enum {
      KSZ9131RNX,
      RTL9068AB,
      MARVELL_88EA1512,
      MARVELL_88Q5072,
      MARVELL_88Q2220,
      QC_AR8031,
      MARVELL_88EA1512_SGMII,
      MARVELL_AQR113_SGMII,
      SGMII_SWITCH_2500,
      <New switch type>
   } phy_switch_type;
   ```

   The enumeration “RTL9068AB” represents an example switch type used on a reference board, add the new switch type at <New switch type>.

2. This value is used in the emac_mdio.c and phy_wrapper.c main file, look for the following clause:

   ```
   if(pdata->phy_type == <New switch type>)
   ```

NOTE: OEMs might require additional changes that are specific to the switch used within the <New switch type> clause.

For numerous PHY switch-related changes, make a separate ENUM entry for the specific switch.

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.