Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

AN224619

How to Use Ethernet Controller in Traveo II Family

Author: Akshat Saraf

Associated Part Family: Traveo™ II Family CYT4B/CYT4D Series

Related Documents: see Related Documents

AN224619 describes how to setup the Ethernet MAC (EMAC) controller in Traveo II MCUs. This application note shows the basic features, register setting, configuration, and usage of Ethernet controller based on the use case.

Contents

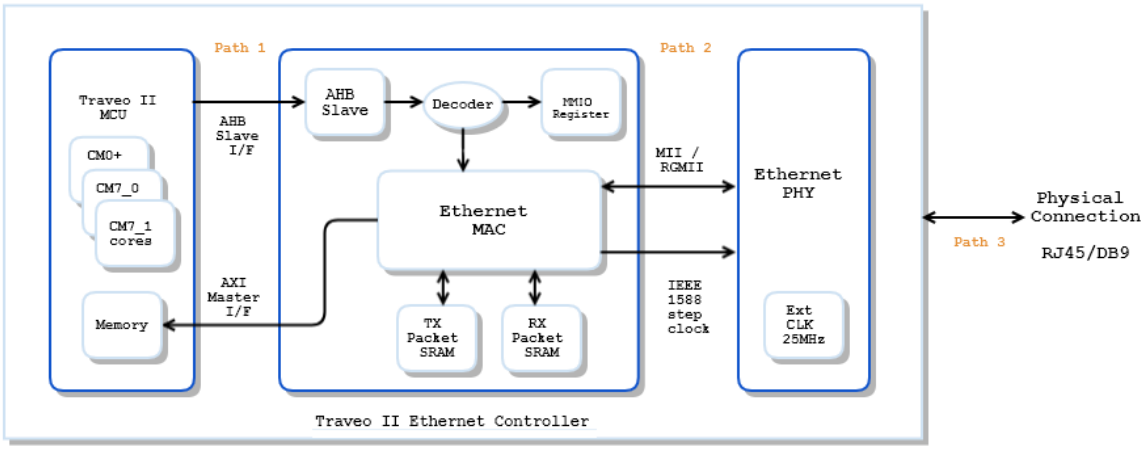
1 Introduction

This application note describes how to use and setup Ethernet controller in Cypress Traveo II family CYT4B Body High and CYT4DN Cluster device series MCUs.

The Ethernet MAC module in the device implements a 10/100/1000 Mbps Ethernet MAC compatible with the IEEE 802.3 standard, supporting MII, RMII, GMII, and RGMII PHY interfaces to support several automotive applications. See the Architecture Technical Reference Manual (TRM) for details of Ethernet.

To understand the functionality described and terminology used in this application note, see the Ethernet MAC chapter in the Architecture TRM. Figure 1 shows examples of typical signal paths.

Figure 1. Ethernet Controller Interface



- Path 1: ETH-MAC module communicates with the CPU core using AHB bus while AXI interface is used for DMA access to the memory.
- Path 2: ETH-MAC module communicates with PHY interface using MII, RMII, GMII and RGMII interface.
- Path 3: Ethernet PHY converts MII/RGMII signals to physical channel signals.

1.1 Ethernet Controller Features

The Ethernet MAC module in the device implements a 10/100/1000 Mbps EMAC compatible with the IEEE 802.3 standard, supporting MII, RMII, GMII, and RGMII PHY interfaces to support several automotive applications.

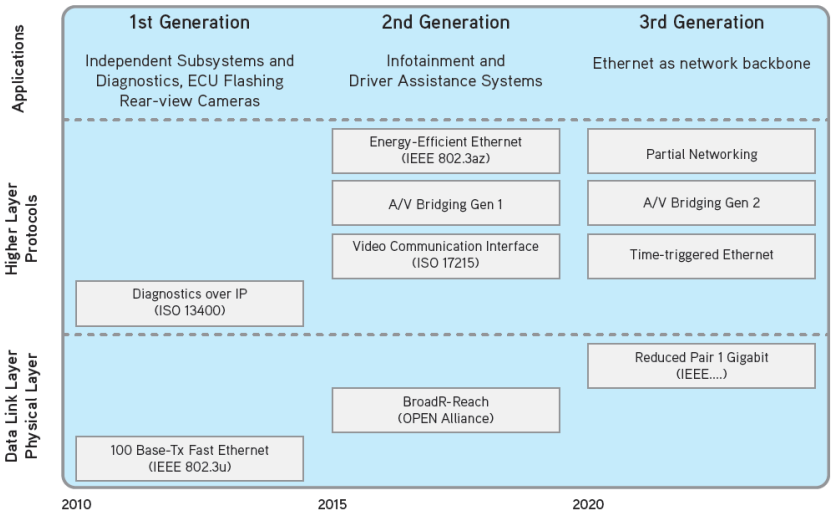
The main features of Ethernet Controller are:

- Full Store-Forward mode and Partial Store-Forward mode for full-duplex operation
- 10 Mbits, 100 Mbits, or 1 Gbps operation
- MII, RMII, GMII and RGMII PHY interface modes
- 1536 bytes of maximum frame length
- Three transmit and receive priority queues
- IEEE Std 802.1BA – Audio Video Bridging Systems
- IEEE Std 802.1Qav – Forwarding and Queuing Enhancements for Time-Sensitive Streams
- IEEE Std 802.1AS – Timing and Synchronization for Time-Sensitive Application in Bridged LANs
- IEEE Std 1588 – Precision Time Protocol
- IEEE 802.3 Pause frame and MAC PFC priority based pause frame support
- Receive and transmit IP, TCP, and UDP checksum offload
- Automatic pad and CRC generation on transmitted frames
- MDIO interface for PHY management
- Strict priority, DWRR, or Enhanced Transmission Selection (ETS – 802.1Qaz) on transmit queues
- Support for 802.3az EEE

1.2 Application of Ethernet

Automotive Ethernet is a physical network used to connect components within a car using a wired network. It is designed to meet the needs of the automotive market, including meeting electrical requirements (EMI/RFI emissions and susceptibility), bandwidth requirements, latency requirements, synchronization, and network management requirements.

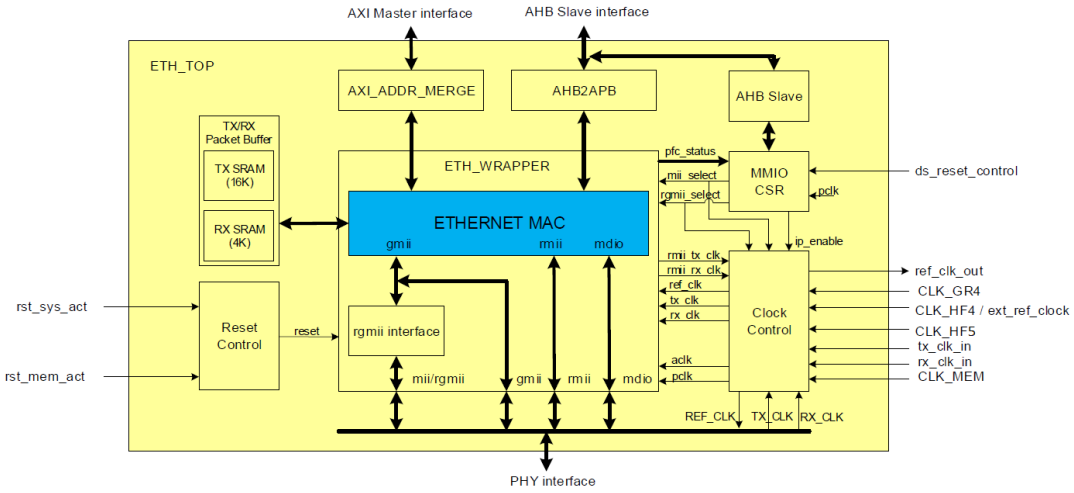
Figure 2. Car Electronics Anatomy using Ethernet



2 Operational Overview

Error! Reference source not found. Figure 3 shows the block diagram of EMAC module. EMAC is positioned in the signal path between the MCU cores and the PHY transceiver.

Figure 3. Block Diagram of Ethernet MAC (EMAC)



2.1 Controller Interfaces

The main interfaces of Ethernet Controller in Traveo II are:

- AXI Master interface
- AHB Slave interface
- Media Independent interfaces (MII/RGMII/RMII)
- MDIO interface

2.1.1 AXI Master Interface

AXI Master interface attached to the EMAC provides separate data channels and common address channels for read and write operations. With stated channels, the interface supports two outstanding transactions on both the Read and Write channels. EMAC is required to store configuration parameters for each transmit and receive frame through descriptors.

TX and RX descriptor reads are issued up-front and stored in a local buffer to feed the underlying DMA when required. This optimizes performance and avoids the need for the underlying DMA to pause while new descriptor fetches are sent to the system bus. TX and RX descriptor writes issued by the underlying DMA are buffered locally to avoid holding up the underlying DMA when the system delays the completion of descriptor writes. Note that a descriptor write transaction is not considered complete until the write response (BRESP) associated with that transaction has arrived.

The DMA can use programmable maximum burst lengths. Single accesses and bursts with up to four beats (pulses) can be selected. With 64-bit data path and a burst length setting of 4, 32 bytes transfers can be made with a single request. The burst length is controlled via the `dma_config` register.

2.1.2 AHB Slave Interface

The AHB Slave interface is used to program the EMAC related registers. The interface will internally go through a decoder to identify if the access is targeted for MMIO or MAC module. The registers implemented in MMIO will be used to control some configurable inputs of the EMAC interface.

2.1.3 Media Independent Interface

EMAC supports four different PHY interfaces – MII, RMII, GMII, and RGMII. Table 1 shows the required settings to select any of the interfaces.

Table 1. PHY Interface Selection

CTL.ETH_MODE	Network_config[0] (Speed)	Network_config[10] (gigabit_mode_enable)	PHY Mode
2'd0	0	0	MII – 10 Mbps
2'd0	1	0	MII – 100 Mbps
2'd1	0	1	GMII – 1000 Mbps
2'd2	0	0	RGMII – 10 Mbps (4bits/Cycle)
2'd2	1	0	RGMII – 100 Mbps (4bits/Cycle)
2'd2	0	1	RGMII – 1000 Mbps (4bits/Cycle)
2'd3	0	0	RMII – 10 Mbps
2'd3	1	0	RMII – 100 Mbps

The EMAC interfaces with an external PHY (reference Texas Instruments DP83867) using MII, RMII or RGMII signals. RGMII and GMII/MII interfaces are design time configurable and mutually exclusive in Ethernet Controller.

Note: ETH_MODE must be configured before configuring network config register. Check the device specific datasheet for the PHY supported modes.

2.1.4 MDIO Interface

MDIO is a single bi-directional tristate signal between the EMAC and PHY. PHY maintenance register (`phy_management`) is implemented as a shift register. Writing to the register starts a shift operation, which is signaled as complete when bit two is set in the network status register (about 2000 pclk cycles later when bits [18:16] are set to 010 in the network configuration register). An interrupt is generated as this bit is set.

During this time, the MSb of the register is output on the mdio_out pin and the LSb updated from the mdio_in pin with each MDC cycle. This causes the transmission of a PHY management frame on MDIO. Reading during the shift operation will return the current contents of the shift register. At the end of the management operation the bits will have shifted back to their original locations. For a read operation, the data bits will be updated with data read from the PHY. It is important to write the correct values to the register to ensure that a valid PHY management frame is produced.

MDC should not toggle faster than 2.5 MHz (minimum period of 400 ns), as defined by the IEEE 802.3 standard. MDC is generated by dividing CLK_GR4. Three bits in the network configuration register determine by how much pclk should be divided to produce MDC.

2.2 DMA Interface

Ethernet MAC accesses data from other available system memory through DMA interface and stores fetched data in local dedicated TX/RX packet buffer. DMA is attached to the Ethernet MAC's external FIFO interface to provide a scatter gather type capability for packet data storage. Configured for packet buffering mode, DMA uses dual port memory to store fetched data. This configuration allows the application to use either of the following operation modes to store and forward data:

- Full Store and Forward Mode
- Partial Store and Forward Mode

In full store and forward mode, the packet will be replayed directly from the packet buffer memory instead of being fetched again from the system memory through AXI. In this way, this mechanism is reducing AXI bus transfers.

In partial store and forward mode, the transmitter will only forward the packet to the MAC when there is enough frame data stored in the packet buffer. Similarly, in case of receive operation, the receiver will only begin to forward the packet to the external AXI slave when enough frame data are stored in the local packet buffer.

The following features have also become available due to this approach:

- Transmit TCP/IP checksum offload
- Priority queuing
- RX packet flush when there is lack of resource
- Burst padding at end of packet and end of buffer to maximize AXI efficiency
- TX/RX timestamp capture to buffer descriptor entry

2.2.1 Full Store and Forward Mode using Packet Buffer DMA

In full store and forward mode, the EMAC only starts transmission when the complete transmit frame is written into the local TX buffer. Transmitted frame will be flushed from the local buffer only after the EMAC completes the transmission and TX BD is updated with the status fields.

In receive process, DMA starts forwarding data to the configured memory address only after the entire frame has been received and does not contain any error. Received frame will be flushed from local packet buffer only after the frame is copied and RX BD is updated with the status fields.

When the EMAC DMA is configured in the full store and forward mode, a receive over run condition occurs when the receive packet buffer memory is full, or because an AXI error occurred.

The benefits of full store and forward mode are:

- Discard packets that are received with errors before they are partially written out of DMA thus saving AXI bandwidth and driver processing overhead
- Retry failed transmit frames from the packet buffer itself, thus saving AXI bus bandwidth
- Implement transmit IP/TCP/UDP checksum offload
- Allows multi-buffer frames

2.2.2 DMA Transaction

The Ethernet controller DMA uses separate transmit and receive lists of buffer descriptors, with each descriptor describing a buffer area in system memory. This allows Ethernet packets to be broken up and scattered around the system memory.

The DMA controller performs six types of operation on the AMBA bus. In order of priority these are:

- Receive buffer manager write/read
- Transmit buffer manager write/read
- Receive data DMA write
- Transmit data DMA read

All read operations are routed to the AXI read channel and all write operations to the AXI write channel. Both read and write channels may operate simultaneously. Arbitration logic is used when multiple requests are active on the same channel.

Transfer size is set to 64-bit words by default in the `network_config` register and burst length can be programmed in the range from single access up to 256 accesses per burst using the `dma_config` register. It is recommended to set burst length maximum to 4 to have a quicker arbitration for all masters accessing the bus.

2.3 Ethernet Packet Buffer

2.3.1 Packet Buffer Memory

The Ethernet controller is configured in a packet buffering mode using dedicated SRAM memories (TX Packet SPRAM and RX Packet SPRAM). The IP is configured to use single port SRAM.

For MAC receiver, the total buffer size will be 4 KB. The reason to allocate 4 KB for Rx SRAM is to be able to store at least two maximum length packets (1.5 KB) to avoid drop packets when operating in full store and forward mode.

For MAC transmitter, the buffer size will be configured to be 4 KB for Priority Queue 0 and 2 KB for Priority Queue 1 and Priority Queue 2, respectively.

2.3.2 Receive Buffers

Received frames, optionally including FCS, are written to receive buffers located in system memory. The receive buffer depth is programmable in the range of 64 bytes to 16 Kbytes in the DMA configuration register, with the default being 1536 bytes. If received frames are being routed to different priority queues via screening registers, it is possible to program different receive buffer depths for each queue. For queue 0, the receive buffer depth is programmed through the DMA configuration register (offset 0x10). For the other queues, the receive buffer depths are programmed through specific queue configuration registers (starting from offset 0x4a0). Default is 128 bytes.

Start address for each receive buffer is stored in system memory in a list of receive buffer descriptors at an address location described by the receive buffer queue pointer. The base address of the receive buffer queue pointer (also referred as the list of buffer descriptors) must be configured by software using the receive buffer queue base address registers.

Each buffer descriptor can be of either two or four words, depending on configured buffer descriptor (BD) mode, whereas word is defined as 32 bits. The first two words (Word 0 and Word 1) are used in both BD modes.

In Extended Buffer Descriptor mode, two BD words (Word 2 and Word 3) are added for timestamp capture if Timestamp Capture mode is enabled. Therefore, BDs will be of either two or four words size and each BD will have the same size.

To summarize, each BD must be of:

- 64 bits when Descriptor Time Capture mode is disabled
- 128 bits when Descriptor Time Capture mode is enabled

Note: Writing receive buffer queue base address register may require three AXI clock cycles to take effect. Therefore, reception cannot be enabled until three AXI clock cycles after receiving buffer queue base address register is updated. Firmware needs to take care of this restriction.

2.3.3 Transmit Buffers

Frames to be transmitted can be stored in one or more transmit buffers. Transmit frames can be between 1 and 1536 bytes long. Note that zero length buffers are allowed and the maximum number of buffers permitted for each transmit frame is 128.

The start addresses of each transmit buffer is stored in system memory in a list of transmit buffer descriptors located at the transmit buffer queue pointer. The base addresses of the transmit BD list must be configured by software using the transmit buffer queue base address registers.

Each buffer descriptor can be of either two or four words, depending on the configured BD mode, whereas word is defined as 32 bits. The first two words (Word 0 and Word 1) are used in both BD modes.

In Extended Buffer Descriptor mode, two BD words are added for timestamp capture if Timestamp Capture mode is enabled. Therefore, Transmit BDs will be of either two or four words size and each BD will have the same size.

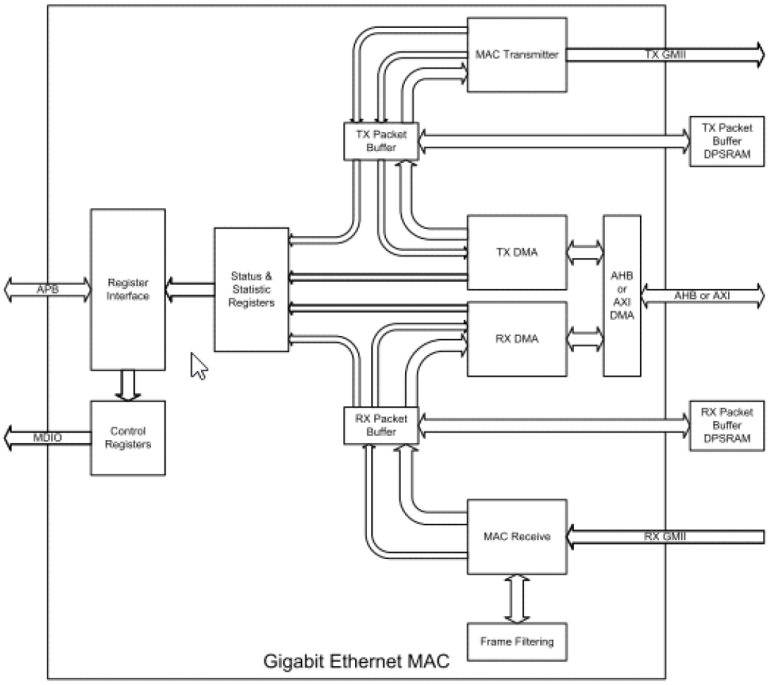
To summarize, each transmit BD must be of:

- 64 bits when Descriptor Time Capture mode is disabled
- 128 bits when Descriptor Time Capture mode is enabled

2.3.4 DMA Packet Buffer

The packet buffer DMA mode allows multiple packets to be buffered in both transmit and receive directions and allows the DMA to withstand variable levels of access latencies on the AXI fabric. Using packet buffers, AXI bandwidth has been used most efficiently in the device. Figure 4 illustrates the structure of the EMAC data paths.

Figure 4. Ethernet MAC Data Path Structure



In the transmit direction, DMA will continue to fetch packet data up to a limit of 256 packets, or until the TX Packet Buffer Memory is full. In the receive direction, if the RX Packet Buffer Memory becomes full, then an overflow will occur. An overflow will also occur if the limit of 256 packets is breached.

2.4 Clock, Reset and Power modes

2.4.1 Clock

Clock requirements and configurations are different for each interface. Following are the required clocks and source of clocks:

- MII
  - Both TX and RX clocks are supplied from external PHY.
- RMII
  - Both TX and RX clocks can be supplied from either internal reference clock or from external clock source.
  - CTRL.REFCLK_SRC_SEL must be used to select the reference clock source from on-chip system resource or from HSIO.
    - TX_CLK_OUT will be enabled to supply reference clock to PHY when internal clock source is selected.
  - CTRL.REFCLK_DIV must be used to divide reference clock and generate the required frequency of 50 MHz.
- GMII
  - RX clock is supplied from PHY
  - TX clock source can be selected either from an internal clock source or from HSIO
  - CTRL.REFCLK_SRC_SEL must be used to select the clock source for TX functionality
  - CTRL.REFCLK_DIV is used to divide the reference clock and to generate the required clock of 125 MHz
  - TX_CLK_OUT will be enabled to supply the TX reference clock to PHY when an internal clock source is selected.
- RGMII
  - RX clock is supplied from PHY.
  - TX clock source can be selected either from an internal clock source or from HSIO

- CTRL.REFCLK_SRC_SEL must be used to select the clock source for TX functionality
- CTRL.REFCLK_DIV is used to divide the reference clock and to generate the required clock of 125 MHz
- TX_CLK_OUT will be enabled to supply the TX reference clock to PHY when internal clock source is selected

Note: For RGMII and GMII transmit operations, please use precise external clock source instead of the internal PLL. Ethernet MAC requires clocks to perform internal operation such as buffer data transfers or TSU operations. To perform those operations the following clocks are used. For more information about configuring mentioned clocks, see the Clocking System chapter in the TRM.

Table 2. Clocks to Ethernet MAC

Clock	Description
CLK_GR4	AHB operations in the IP and to generate MDC clock
CLK_HF5	Time Stamp Unit (TSU)
CLK_MEM	AXI operation
CLK_HF4	Internal reference clock for media independent interface

Check the device datasheet for appropriate clocks to different Ethernet instances.

2.4.2 Reset

The Ethernet controller can only be operated in the system Active power mode. In DeepSleep power mode, all logic including dedicated SRAMs are not retained except for the retention MMIO registers. As such, the retention MMIO register is reset by a deepsleep system reset.

2.4.3 Power modes

Ethernet controller is an Active Peripheral. In DeepSleep power mode, only the retention MMIO registers are retained. Table 3 shows Ethernet MAC availability in different device power modes.

Table 3. Ethernet MAC Status in Different Device Power Modes

Device Power Modes	IP Status
Active	EMAC is fully operational in Active power mode with power on and clocks running.
LPActive	EMAC is fully operational in LPActive power mode with power on and clocks running; clocks can be limited to save some power during LPActive mode.
Sleep	EMAC is fully operational in Sleep power mode.
LPSleep	EMAC is fully operational in LPSleep power mode with power on and clocks running; clocks can be limited to save some power during LPSleep mode.
DeepSleep	No clock is provided during DeepSleep power mode; hence the logic is not functional. All retention registers will hold the values in DeepSleep mode.
Hibernate	Entire EMAC including retention register are not functional during Hibernate power mode.

2.5 Interrupts

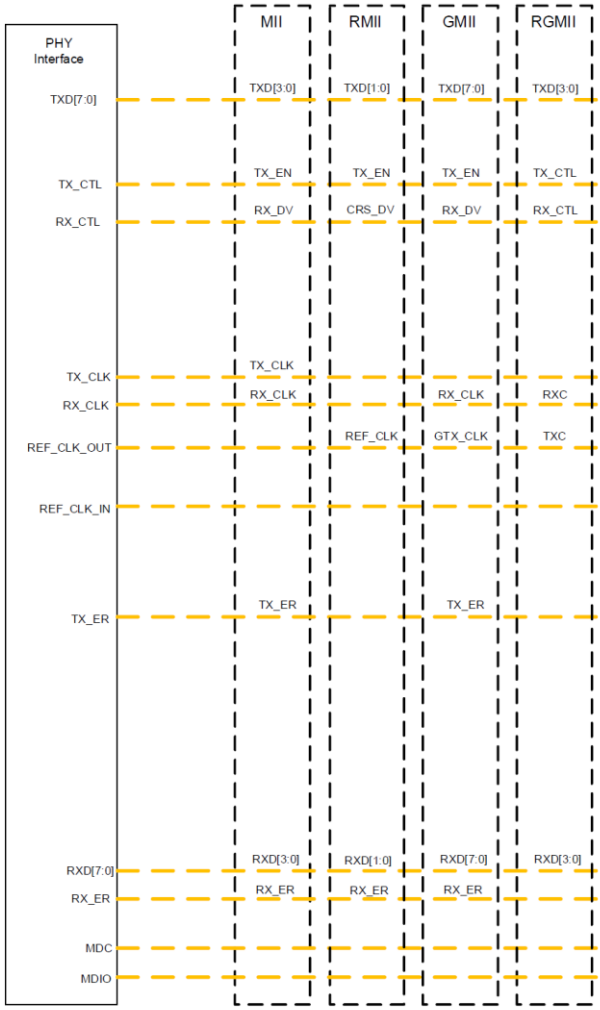
Several interrupt conditions can be enabled in EMAC. The interrupt outputs from the Ethernet MAC match the number of supported priority queues. Only EMAC DMA related events are reported using the individual interrupt outputs, as the Ethernet MAC can relate these events to specific queues. All other events generated within the Ethernet MAC are reported in the interrupt associated with the lowest priority queue (Queue 0). For the lowest priority queue, the interrupt status register is located at offset address 0x024. For all other priority queues, this register is located at sequential offset addresses starting at 0x400.

At reset, all interrupts are disabled. To enable an interrupt, write to interrupt enable register with the pertinent interrupt bit set to 1. To disable an interrupt, write to interrupt disable register with the pertinent interrupt bit set to 1. To check whether an interrupt is enabled or disabled, read interrupt mask register: if the bit is set to 1, the interrupt is disabled.

2.6 PHY Interface

Figure 5 shows the block diagram of Ethernet PHY module. Ethernet PHY has all signals used by all the Media Independent interfaces MII, RMII, GMII, and RGMII. Figure 5 also shows the physical signals used by individual MII interfaces.

Figure 5. Block Diagram of Ethernet MAC



3 Ethernet Configuration

3.1 Register Sets

Table 4. Ethernet Controller Register Set

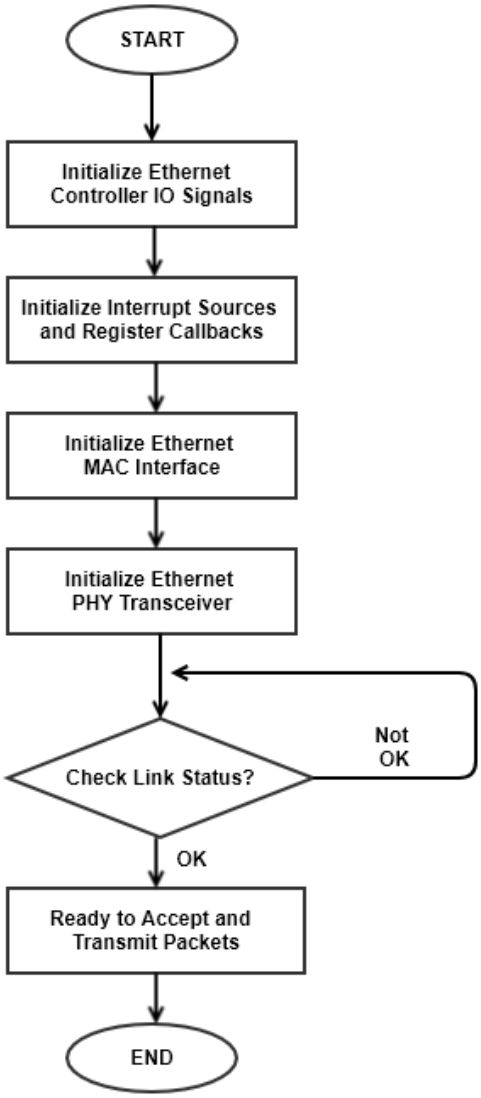
Register	Name	Description
CTL	Control Register	Control register to select type PHY mode and reference clock, and to enable the IP.
NETWORK_CONTROL	Network Control Register	The network control register contains general MAC control functions for both receiver and transmitter.
NETWORK_CONFIG	Network Config Register	The network configuration register contains functions to set the mode of operation for the Gigabit Ethernet MAC.
DMA_CONFIG	DMA Config Register	Register to configure DMA transfers for transmit and receive operation.
TRANSMIT_Q_PTR	Transmit Queue 0 Pointer	This register holds the start address of the transmit buffer queue (transmit buffers descriptor list).
PHY_MANAGEMENT	PHY Management Register	This register is implemented as a shift register. Writing to the register starts a shift operation which is signaled as complete when bit-2 is set in the network status register.

Table 4 shows the register sets used in example provided in section 4.1. For more details on Ethernet controller registers, see the Register TRM.

3.2 Configuration Flow

Figure 6 shows an example of the configuration flow of Ethernet controller in Traveo II.

Figure 6. Ethernet Controller Configuration Flow



4 Example Configuration

Ethernet can be useful for an application that involves communication between two ECU or nodes at higher bandwidth and speed. Ethernet controller uses MAC-based source and destination address communication so that theoretically millions of nodes can be present in the network. This section explains how to use Ethernet controller as per the use case.

4.1 Ethernet Configuration for MII Mode with 100 Mbps

This use case demonstrates how to configure Ethernet controller for MII mode with link speed 100 Mbps. The following are the basic parameters:

- Ethernet Interface : ETH0 or ETH1
- Ethernet Mode : MII
- Link Speed : 100Mbps
- Clock Source : CLK_HF5, CLK_HF4 and TX_CLK, RX_CLK from transceiver
- Communication Mode : Full Duplex
- Ethernet Packet Size : 1536 byte
- CPU Core : CM7_0

Note that this configuration is generated for the CYT4B devices; but the same techniques apply to CYT4D devices (check IO port and Ethernet Interface macro ETHx for different devices and family).

The following sections explain the configuration procedure of this use case. Ethernet IO Port Setting.

See the I/O System chapter of the Architecture TRM for details of IO initialization. The following is the Ethernet IO drive mode configuration for MII mode:

Signal	Drive Mode	Direction
ETHx_TD0	STRONG_IN_OFF	MAC to PHY
ETHx_TD1	STRONG_IN_OFF	MAC to PHY
ETHx_TD2	STRONG_IN_OFF	MAC to PHY
ETHx_TD3	STRONG_IN_OFF	MAC to PHY
ETHx_TXER	STRONG_IN_OFF	MAC to PHY
ETHx_TX_CTL	STRONG_IN_OFF	MAC to PHY
ETHx_RD0	HIGHZ	PHY to MAC
ETHx_RD1	HIGHZ	PHY to MAC
ETHx_RD2	HIGHZ	PHY to MAC
ETHx_RD3	HIGHZ	PHY to MAC
ETHx_RX_CTL	HIGHZ	PHY to MAC
ETHx_REF_CLK	HIGHZ	MAC to PHY
ETHx_TX_CLK	HIGHZ	PHY to MAC
ETHx_RX_CLK	HIGHZ	PHY to MAC
ETHx_MDC	STRONG_IN_OFF	MAC to PHY
ETHx_MDIO	STRONG	Bidirectional

4.1.1 Clock Setting

Ethernet controller gets the clock from two sources CLK_HF4 and CLK_HF5. By default, CLK_HF0 is enabled and to enable the CLK_HF4 and CLK_HF5, see the **Clocking System** chapter of the Architecture TRM.

Note: For transmit and receive operation in MII mode, Ethernet MAC gets the clock from PHY.

4.1.2 IRQ and Handler Assignment

To enable an interrupt for a peripheral, peripheral interrupt source should be mapped to one of the CPU interrupt source (available from 0 to 7). Any number of peripheral interrupt sources can be mapped to a CPU interrupt source. For more details, see the Interrupts chapter in the Architecture TRM.

Enable system IRQ with `CPU_Int_Idx3` in `CM7_0` control register. Here, CPU interrupt id 3 is used to map the Ethernet interrupt sources to the CPU core:

```
CPUSS_CM7_0_SYSTEM_INT_CTL |= 0x10000003;                              // Use CPU_Int_Idx3
```

Assign handler `eth_intr_src_handler` for `ETH_INTR_SRC` in VTOR:

```
SystemIrqUserTableRamPointer[ETH_INTR_SRC] = eth_intr_src_handler; // ETH_INTR_SRC
```

Assign handler `eth_intr_src_q1_handler` for `ETH_INTR_SRC_Q1` in VTOR:

```
SystemIrqUserTableRamPointer[ETH_INTR_SRC_Q1] = eth_intr_src_q1_handler;  //
ETH_INTR_SRC_Q1
```

Assign handler `eth_intr_src_q2_handler` for `ETH_INTR_SRC_Q2` in VTOR:

```
SystemIrqUserTableRamPointer[ETH_INTR_SRC_Q2] = eth_intr_src_q2_handler;  //
ETH_INTR_SRC_Q2
```

Set interrupt priority for `CPU_Int_Idx3` source in NVIC:

```
NVIC->IP [CPU_Int_Idx3] = (uint8_t)((priority << (8 - __NVIC_PRIO_BITS))
```

Enable interrupt for `CPU_Int_Idx3` in NVIC:

```
NVIC->ISER[(CPU_Int_Idx3 >> 5UL)] = (uint32_t)(1UL << (((uint32_t) CPU_Int_Idx3) &
0x1FUL));
```

4.1.3 Ethernet Controller Initialization

Set MII mode (by default) and enable Ethernet controller:

```
ETHx->unCTL = 0x80000000;
```

Set communication mode to Full duplex, set no broadcast frames, frame size to 1536 bytes, set divide PCLK by 16, set 64-bit AMBA bus width, and set receive bad preamble frame:

```
ETHx->unNETWORK_CONFIG = 0x24240122;           // by default 10/100 operations using MII
```

Set DMA burst up to 4, use 8 KB of addressable space for receive, use 4 KB of addressable space for transmit, set DMA receive buffer size to 1536 bytes, enable transmitter and receiver extended BD mode, and set DMA address bus width to 32bits:

```
ETHx->unDMA_CONFIG = 0x34180704;
```

Enable transmit queue and configure start address of transmit buffer descriptor list:

```
ETHx->unTRANSMIT_Q2_PTR = (uint32_t) &g_txBuffer2;

ETHx->unTRANSMIT_Q1_PTR = (uint32_t) &g_txBuffer1;
```

```
ETHx->unTRANSMIT_Q_PTR = (uint32_t) &g_txBuffer0;
```

Note: The base addresses of transmit and receive descriptor lists must be aligned to the data word boundary, i.e. 32-bit boundary for 32-bit DMA addressing.

Enable interrupts for receive complete, transmit buffer under-run, retry limit collision, transmit frame corruption, transmit complete, and receive over-run:

```
ETHx->unINT_ENABLE = 0x000004FE
```

Enable MAC receiver and transmitter and enable MDIO in network control register:

```
ETHx->unNETWORK_CONTROL = 0x0000001C;
```

By writing `tx_start_pck` bit to 1, transmission will start:

```
ETHx->unNETWORK_CONTROL |= 0x00000200;
```

4.1.4 Initialization for PHY Transceiver

Check for appropriate PHY and see the device datasheet for more details on register, commands, and data read/write.

4.1.5 Read Link Status

Check for the link status register of the application board PHY in the datasheet. If the link status is ok, configuration is proper and setup is ready to send/receive Ethernet packet.

For configuring different media independent interface settings, see the Architecture TRM and Register TRM.

5 Glossary

Terms	Description
AHB	Advanced High-Performance Bus connects the components that need higher bandwidth.
AXI	Advanced eXtensible Interface connects the on-chip components for high performance, high speed parallel communication. It is a part of ARM specification.
BD	Buffer Descriptor.
CLK_HF	This is derived from the system clock using a peripheral clock divider. See the Clocking System chapter of the Architecture TRM for details.
DeepSleep	Power mode that only low-frequency peripherals are available. See the DeepSleep Mode section in the Device Power Modes chapter of the Architecture TRM for details.
DMA	Direct Memory Access. See the DMA chapter of the Architecture TRM for details.
FCS	Frame Check Sequences
GPIO	General-purpose input/output
HSIOM	High Speed I/O Matrix. See the High-Speed I/O Matrix section in the I/O System chapter of the Architecture TRM for details.
I/O Port	I/O Port provides the interface between the CPU core and peripheral components to the outside world. See the I/O System chapter of the Architecture TRM for details.
MAC	Media Access Controller.
MDC	Management Data Clock.
MDIO	Management Data Input Output
PCLK	PCLK is source clock for different peripheral functions, for Ethernet MAC it is used to generate the MDIO bus clock i.e. Management Data Clock (MDC).
PHY	Physical interface to the Ethernet Controller.
RGMII	Reduced Gigabit Media Independent Interface
RMII	Reduced Media Independent Interface
TSU	Timestamp Unit.

6 Related Documents

The following are the Traveo II family series datasheets and Technical Reference Manuals. Contact Technical Support to obtain these documents.

- Device datasheet
  - CYT4BF Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller Traveo™ II Family
  - CYT4DN Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller Traveo™ II Family
- Architecture TRM
  - Traveo™ II Automotive Body-High Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Cluster 2D Family Architecture Technical Reference Manual (TRM)
- Register TRM
  - Traveo™ II Automotive Body-High Registers Technical Reference Manual (TRM)
  - Traveo™ II Automotive Cluster 2D Registers Technical Reference Manual (TRM)

Document History

Document Title: AN224619 – How to Use Ethernet Controller in Traveo II Family

Document Number: 002-24619

Revision	ECN	Submission Date	Description of Change
**	6676566	09/18/2019	New Application Note.

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 MCU

Cypress Developer Community

Community | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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