How to Use Ethernet Controller in Traveo II Family

About this document

Scope and purpose

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

Associated Part Family

Traveo II Family CYT3/CYT4 Series

Table of contents

1 Introduction

This application note describes how to use and set up the Ethernet controller in Cypress Traveo II family CYT3/4 series MCUs.

The Ethernet MAC (EMAC) module in the device implements a 10/100/1000 Mbps Ethernet MAC compatible with the IEEE 802.3 standard, supporting media-independent interface (MII), reduced media-independent interface (RMII), gigabit media-independent Interface (GMII), and reduced gigabit media-independent Interface (RGMII) PHY interfaces to support several automotive applications. The interfaces supported depend on the device. See the device-specific datasheet to determine the interfaces and number of channels supported in the device. See the **Architecture Technical Reference Manual (TRM)** for details of Ethernet interfaces.

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.

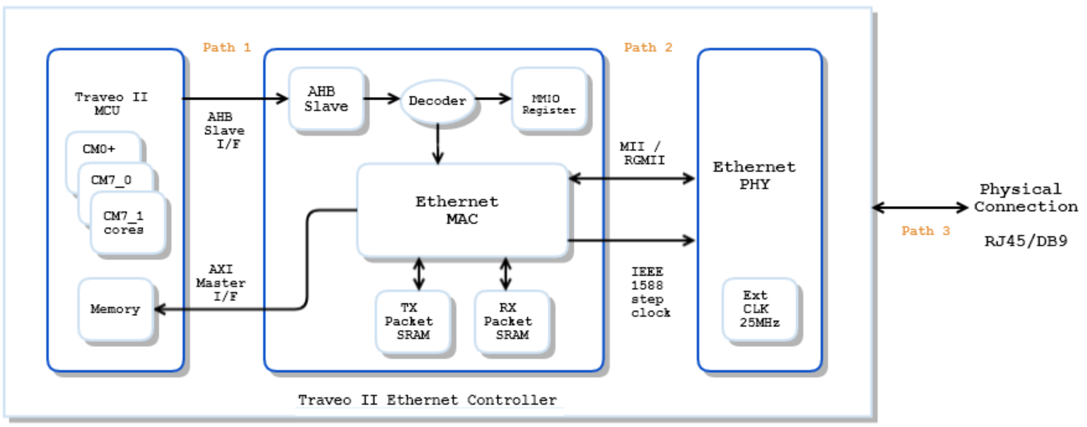

Figure 1 Ethernet controller interface

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

- Full Store and Forward mode and Partial Store and Forward mode for full duplex operation
- 10 Mbps, 100 Mbps, 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 Std 802.3 – Pause frame and MAC Priority-based Flow Control (PFC) priority-based pause frame support
- Receive and transmit IP, TCP, and UDP checksum offload
- Automatic pad and CRC generation on transmitted frames
- Management Data I/O (MDIO) interface for PHY management
- Strict priority, Deficit Weighted Round Robin (DWRR), or Enhanced Transmission Selection (ETS – 802.1Qaz) on transmit queues
- Support for 802.3az Energy Efficient Ethernet (EEE)

1.2 Applications 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.

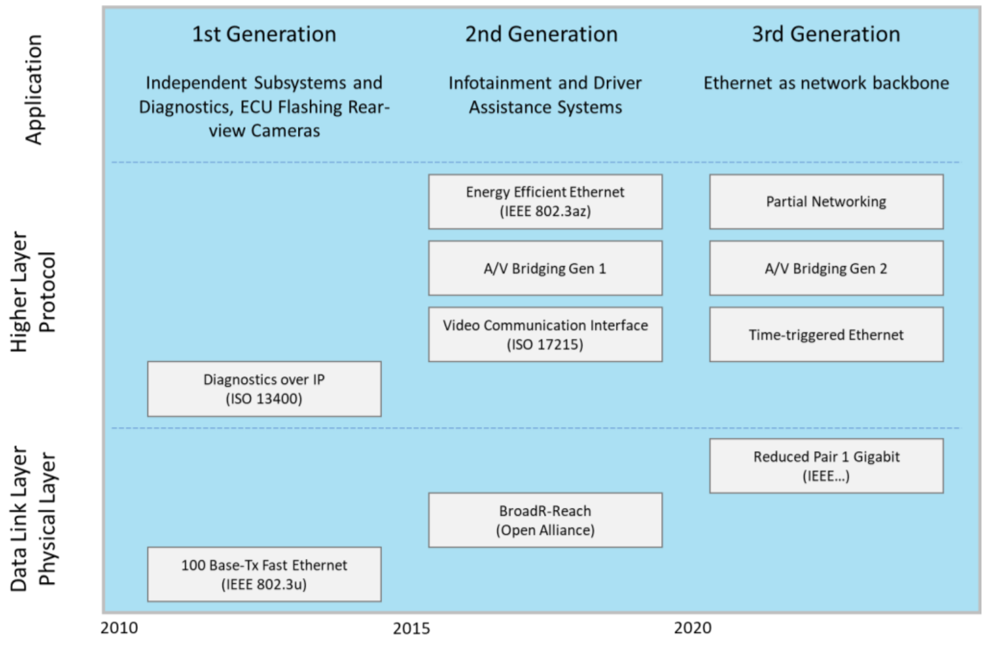
Application
Higher Layer Protocol
Data Link Layer Physical Layer
1st Generation
Independent Subsystems and Diagnostics, ECU Flashing Rear-view Cameras
2nd Generation
Infotainment and Driver Assistance Systems
3rd Generation
Ethernet as network backbone
Energy Efficient Ethernet (IEEE 802.3az)
Partial Networking
A/V Bridging Gen 1
A/V Bridging Gen 2
Video Communication Interface (ISO 17215)
Time-triggered Ethernet
Diagnostics over IP (ISO 13400)
Reduced Pair 1 Gigabit (IEEE...)
BroadR-Reach (Open Alliance)
100 Base-Tx Fast Ethernet (IEEE 802.3u)
2010
2015
2020

Figure 2 Car electronics anatomy using ethernet

2 Operational overview

Figure 3 shows the block diagram of the EMAC module. EMAC is positioned in the signal path between the MCU cores and the PHY transceiver.

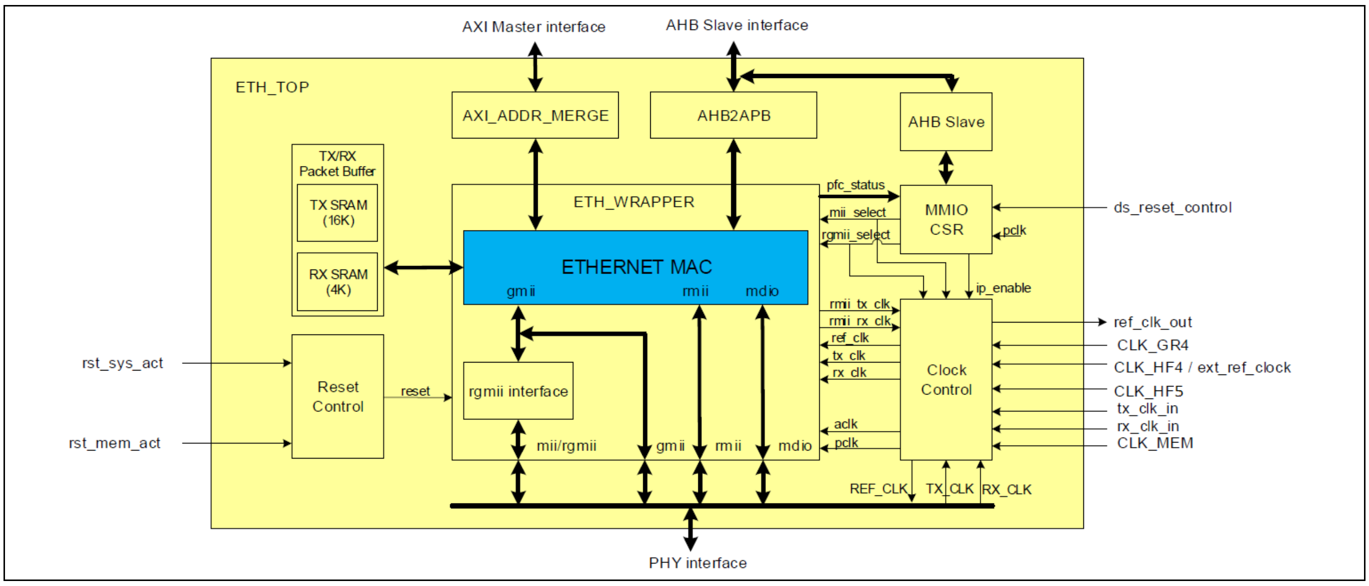

Figure 3 **Block diagram of ethernet MAC (EMAC)**

2.1 Controller interfaces

The Ethernet Controller in Traveo II has the following main interfaces:

- AXI Master interface
- AHB Slave interface
- Media-independent interfaces (MII, RMII, GMII, RGMII)
- MDIO interface

2.1.1 AXI master interface

The 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 the 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 four, 32-byte transfers can be made with a single request. The burst length is controlled via the ETHx_dma_config register. “x” indicates the channel number.

2.1.2 AHB slave interface

The AHB Slave interface is used to program EMAC-related registers. The interface will internally go through a decoder to identify whether the access is targeted for memory-mapped I/O (MMIO0 or the 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 (4 bits/cycle)
2’d2	1	0	RGMII – 100 Mbps (4 bits/cycle)
2’d2	0	1	RGMII – 1000 Mbps (4 bits/cycle)
2’d3	0	0	RMII – 10 Mbps
2’d3	1	0	RMII – 100 Mbps

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 the Ethernet Controller.

Note: *ETH_MODE must be configured before configuring the ETHx_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. The PHY maintenance register (ETHx_phy_management) 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 ETHx_network_status register (about 2000 PCLK cycles later when bits [18:16] are set to 010b in the ETHx_network_config register). An interrupt is generated when this bit is set.

During this time, the most significant bit (MSb) of the ETHx_phy_management register is output on the mdio_out pin and the LSb updated from the mdio_in pin with each Management Data Clock 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 the 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.

The 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 ETHx_network_config register determine by how much PCLK should be divided to produce the MDC.

2.2 DMA interface

Ethernet MAC accesses data from other available system memory such as SRAM through the DMA interface and stores the fetched data in a 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 the fetched data. This configuration allows the application to use either of the following operation modes to store and forward the 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 the AXI. In this way, this mechanism reduces 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 a receive operation, the receiver will begin to forward the packet to the external AXI slave only when enough frame data is stored in the local packet buffer.

This approach makes the following features available:

- Transmit TCP/IP checksum offload
- Priority queuing
- Rx packet flush when there is lack of resources
- 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 starts transmission only when the complete transmit frame is written into the local Tx buffer. The transmitted frame will be flushed from the local buffer only after the EMAC completes the transmission and the Tx buffer descriptor (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 without error. The received frame will be flushed from the local packet buffer only after the frame is copied and the Rx BD is updated with the status fields.

When the EMAC DMA is configured in full store and forward mode, a receive overrun condition occurs when the receive packet buffer memory is full, or because an AXI error has 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
- Allow 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 the system memory. This allows Ethernet packets to be broken up and scattered around the system memory.

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

1. Receive buffer manager write/read
2. Transmit buffer manager write/read
3. Receive data DMA write
4. 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.

The transfer size is set to 64-bit words by default in the ETHx_network_config register; burst length can be programmed in the range from single access up to 256 accesses per burst using the ETHx_dma_config register. It is recommended to set the 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 Single port SRAM (SPRAM) and Rx Packet SPRAM). The EMAC is configured to use SPRAM.

For the 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 dropped 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 frame check sequence (FCS), are written to receive buffers located in the system memory. The receive buffer depth is programmable in the range of 64 bytes to 16 KB 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 other queues, receive buffer depths are programmed through specific queue configuration registers (starting from offset 0x4a0). The default is 128 bytes.

The start address for each receive buffer is stored in the 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 to 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 the configured buffer descriptor (BD) mode, where ‘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; 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 the base address register of the receive buffer queue may require three AXI clock cycles to take effect. Therefore, reception cannot be enabled until three AXI clock cycles after the base address register of the receive buffer queue 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; the maximum number of buffers permitted for each transmit frame is 128.

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

Each buffer descriptor can be of either two or four words, depending on the configured BD mode, where ‘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, the AXI bandwidth has been used most efficiently in the device. **Figure 4** illustrates the structure of the EMAC data paths.

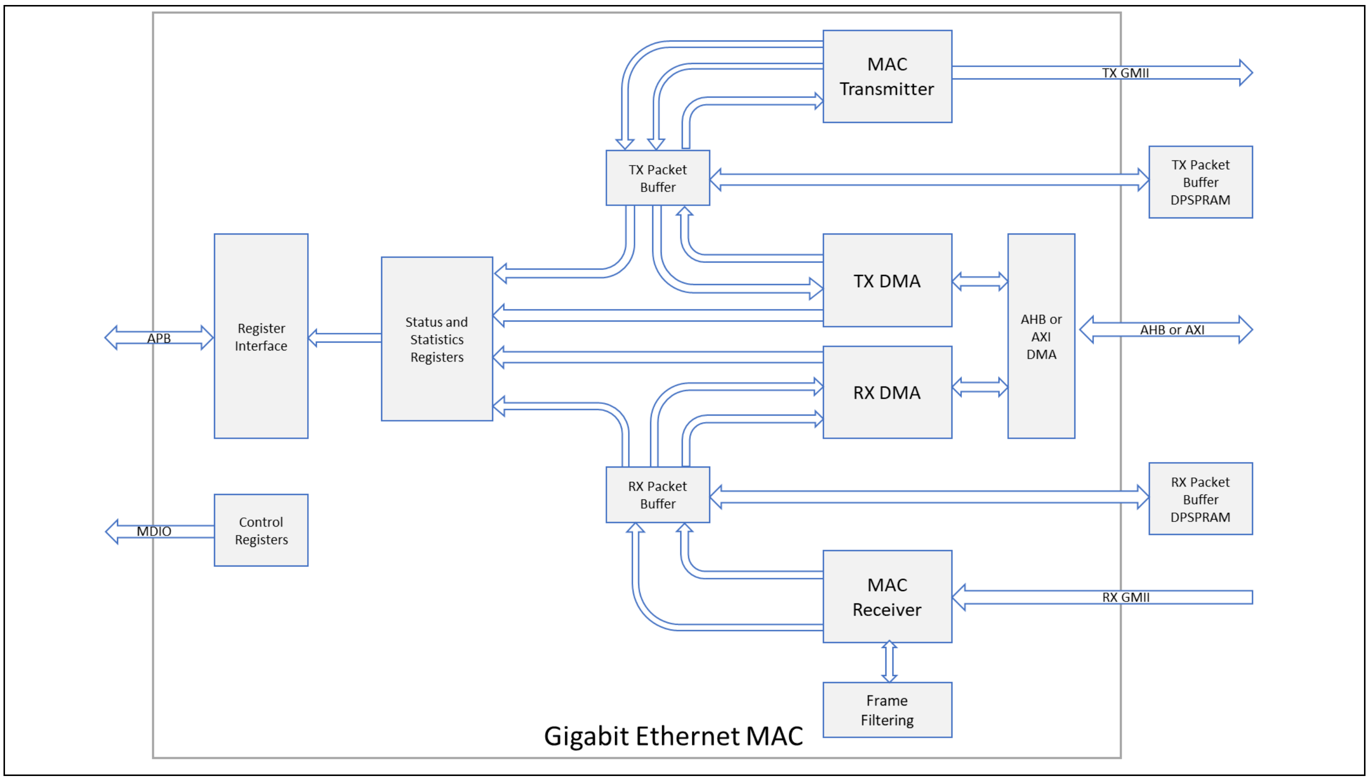

Figure 4 Ethernet MAC data path structure

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, 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. The following clocks and source of clocks are required:

- MII
  - Both Tx and Rx clocks are supplied from the external PHY.
- RMII
  - Both Tx and Rx clocks can be supplied from either an internal reference clock or an external clock source.
  - The Control register (ETHx_CTL.REFCLK_SRC_SEL) must be used to select the reference clock source from on-chip system resources or HSIO.
    TX_CLK_OUT will be enabled to supply reference clock to PHY when an internal clock source is selected.
  - ETHx_CTL.REFCLK_DIV must be used to divide the reference clock and generate the required frequency of 50 MHz.

- GMII
  - Rx clock is supplied from the external PHY.
  - Tx clock source can be selected either from an internal clock source or HSIO.
  - ETHx_CTL.REFCLK_SRC_SEL must be used to select the clock source for transmission functionality.
  - ETHx_CTL.REFCLK_DIV is used to divide the reference clock and 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 the external PHY.
  - Tx clock source can be selected either from an internal clock source or from HSIO.
  - ETHx_CTL.REFCLK_SRC_SEL must be used to select the clock source for transmission functionality.
  - ETHx_CTL.REFCLK_DIV is used to divide the reference clock and 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.

Note: For RGMII and GMII transmit operations, use a precise external clock source instead of the internal PLL. Ethernet MAC requires clocks to perform internal operation such as buffer data transfers or timestamp unit (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 EMAC and to generate the MDC clock
CLK_HF5	Time Stamp Unit (TSU)
CLK_MEM	AXI operation
CLK_HF4	Internal reference clock for MII

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

2.4.2 Reset

The Ethernet controller can be operated only 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

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

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 individual interrupt outputs because 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 which 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 the interrupt enable register with the relevant interrupt bit set to ‘1’. To disable an interrupt, write to the interrupt disable register with the relevant interrupt bit set to ‘1’. To check whether an interrupt is enabled or disabled, read the 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 the Ethernet PHY module. The Ethernet PHY has all signals used by MII, RMII, GMII, and RGMII. **Figure 5** also shows the physical signals used by individual MII interfaces.

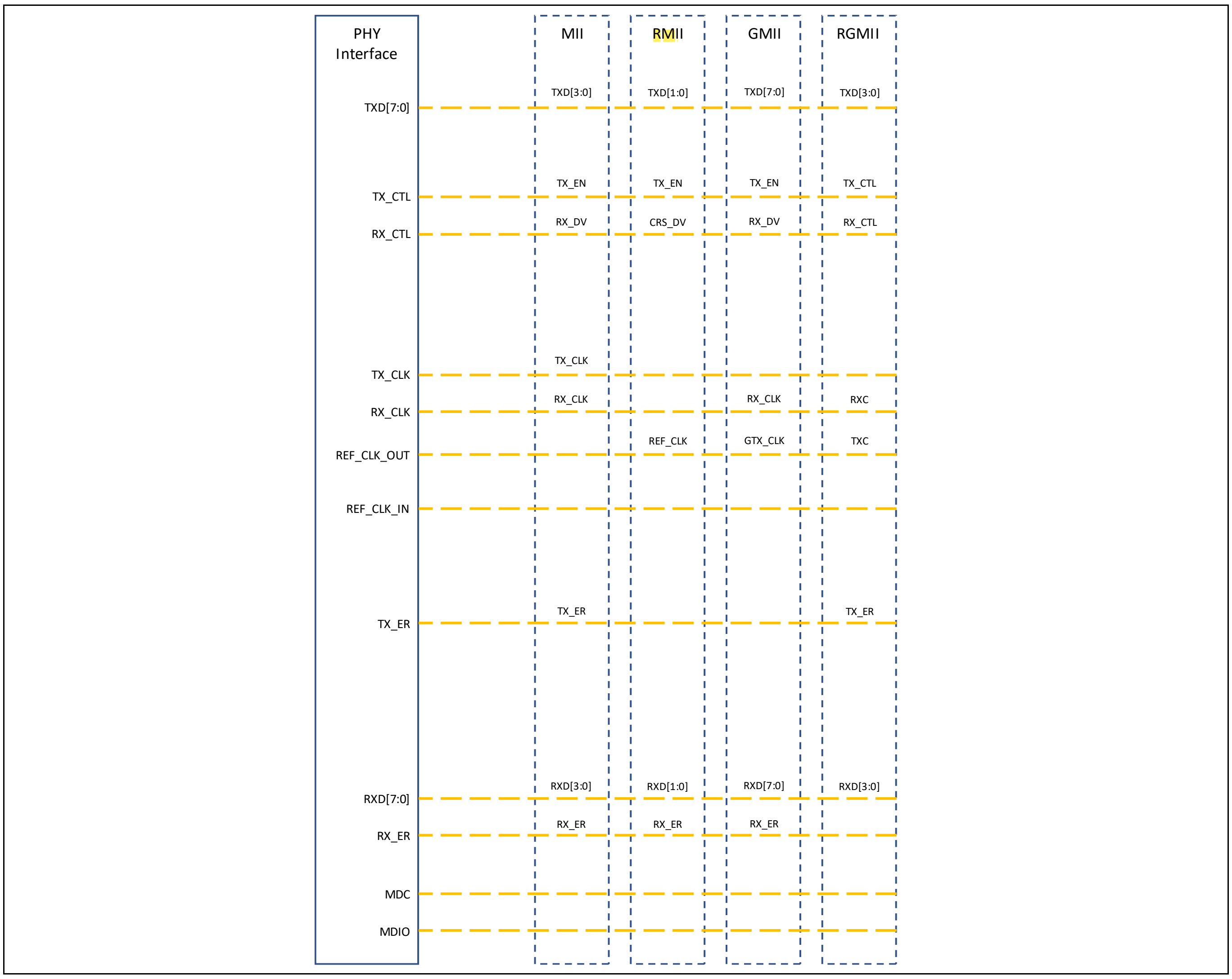

Figure 5 Block diagram of ethernet MAC

3 Ethernet configuration

3.1 Register sets

Table 4 Ethernet controller register set

Register	Name	Description
CTL	Control register	Selects the PHY mode and reference clock, and enables EMAC.
NETWORK_CONTROL	Network control register	Contains general MAC control functions for both receiver and transmitter.
NETWORK_CONFIG	Network config 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 operations.
TRANSMIT_Q_PTR	Transmit queue 0 pointer	Holds the start address of the transmit buffer queue (transmit buffers descriptor list).
PHY_MANAGEMENT	PHY management register	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 Ethernet configuration for MII mode with 100 Mbps**. 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.

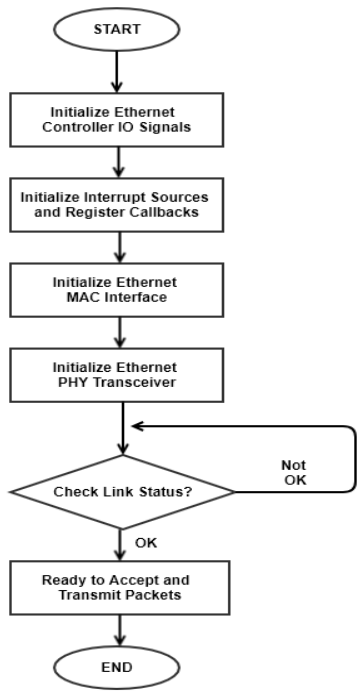
START
Initialize Ethernet Controller IO Signals
Initialize Interrupt Sources and Register Callbacks
Initialize Ethernet MAC Interface
Initialize Ethernet PHY Transceiver
Check Link Status?
Not OK
OK
Ready to Accept and Transmit Packets
END

Figure 6 Ethernet controller configuration flow

4 Example configuration

Ethernet can be useful for an application that involves communication between two ECUs or nodes at higher bandwidth and speed. The 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 the Ethernet controller as per the use case.

4.1 Ethernet configuration for MII mode with 100 Mbps

This use case demonstrates how to configure the 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 bytes
• CPU Core	: CM7_0

Note that this configuration is generated for the CYT4B devices; the same techniques apply to other devices (check I/O port and Ethernet Interface macro ETHx for different devices and families).

The following sections explain the configuration procedure of this use case.

4.1.1 Ethernet I/O port setting

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

Table 5 Ethernet I/O 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

Example configuration

Signal	Drive mode	Direction
ETHx_MDIO	STRONG	Bidirectional

4.1.2 Clock setting

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

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

4.1.3 IRQ and handler assignment

To enable an interrupt for a peripheral, the peripheral interrupt source should be mapped to one of the CPU interrupt sources (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 the system IRQ with `CPU_Int_Idx3` in the `CM7_0` control register. Here, the 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 the handler eth_intr_src_handler for ETH_INTR_SRC in VTOR:

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

Assign the 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 the 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 the interrupt priority for `CPU_Int_Idx3` source in NVIC:

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

Enable the interrupt for `CPU_Int_Idx3` in NVIC:

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

4.1.4 Ethernet controller initialization

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

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

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

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

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

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

Enable the transmit queue and configure the start address of the 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: *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 underrun, retry limit collision, transmit frame corruption, transmit complete, and receive overrun:

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

Enable MAC receiver and transmitter, and MDIO in the ETHx_network_control register:

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

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

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

4.1.5 Initialization for the PHY transceiver

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

4.1.6 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 packets.

For configuring different MII settings, see the **Architecture TRM** and **Register TRM**.

5 Glossary

Table 6 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.
DMA	Direct Memory Access. See the DMA 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.
FCS	Frame Check Sequence
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
SPRAM	Single Port SRAM
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
  - CYT3BB/4BB Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller Traveo™ II Family
- Body Controller High Family
  - Traveo™ II Automotive Body-High Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Body-High Registers Technical Reference Manual (TRM) for CYT4BF
  - Traveo™ II Automotive Body-High Registers Technical Reference Manual (TRM) for CYT3BB/4BB
- Cluster 2D Family
  - Traveo™ II Automotive Cluster 2D Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Cluster 2D Registers Technical Reference Manual (TRM)

Revision history

Document version	Date of release	Description of changes
**	2019-09-18	New application note.
*A	2020-07-22	Updated Associated Part Family as “Traveo™ II Family CYT3/CYT4 Series”. Changed target part numbers from “CYT4B/CYT4D Series” to “CYT4 Series” in all instances across the document. Added target part numbers “CYT3 Series” in all instances across the document.
*B	2021-05-26	Updated to Infineon template.

Edition 2021-05-26

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Go to **www.cypress.com/support**

Document reference

002-24619 Rev. *B