Application Note

Ethernet Frame Transmission Using the eZ80F91 MCU

AN021204-0813

Abstract

This application demonstrates how to configure the eZ80F91 MCU to enable communication through an Ethernet cable. Data is transferred from one eZ80F91 device to another eZ80F91 device, and vice versa. The HyperTerminal emulation program (or similar software) aids the application by displaying data received via the Ethernet and acquiring data to send to the eZ80F91 device at the other end of the Ethernet cable. The application discussed in this document follows the IEEE 802.3 frame format.

Note: The source code file associated with this application note, AN0212-SC01.zip, is available for free download from the Zilog website. This source code has been tested with version 5.2.1 of ZDSII for eZ80Acclaim! MCUs. Subsequent releases of ZDSII may require you to modify the code supplied with this application note.

Throughout this document, the term *development board* is used to denote a board from one of the following three eZ80 development kits:

- eZ80F91 Development Kit (eZ80F910300ZCOG): eZ80 Development Platform (99C0858-001G) with an eZ80F91x150MODG Module (99C0879-001G)
- eZ80F91 Modular Development Kit (eZ80F910200KITG): eZ80 Modular Development Board (98C0945-001G) with an eZ80F915005MODG Module (99C0942-001G) or an eZ80F916005MODG Module (99C0942-002G)
- eZ80Acclaim*Plus!* Development Kit (eZ80F910300KITG): eZ80Acclaim*Plus!* Development Board (99C1322-001G)

Features

Zilog's eZ80F91 MCU features an on-chip Ethernet Media Access Controller (EMAC) unit with a fully-functional 10/100Mbps Media-Independent Interface (MII). This EMAC contains the following blocks:

- Central clock and reset module
- Host memory interface and transmit/receive arbiter
- FIFO buffer and DMA control blocks for transmit and receive
- 802.3x media access control block
- MII interface management

The eZ80F91 MCU offers the following main features:

- Single-cycle instruction fetch in a high-performance, pipelined eZ80 CPU core
- 10/100 BaseT ethernet media access controller with Media-Independent Interface (MII)
- 256KB Flash memory
- 16KB SRAM (8KB user and 8KB Ethernet)
- Low-power features including Sleep and Halt modes, plus selective peripheral power-down control
- Two Universal Asynchronous Receivers/Transmitters (UARTs) with independent Baud Rate Generators (BRGs)
- Serial Peripheral Interface (SPI) with independent clock rate generator
- I^2C with independent clock rate generator
- IrDA-compliant infrared encoder/decoder
- Glueless external peripheral interface with 4 chip selects, individual wait state generators, an external WAIT input pin; supports Z80-, Intel-, and Motorola-style buses
- Fixed-priority vectored interrupts (both internal and external) and interrupt controller
- Real-time clock with separate V_{DD} pin for battery backup and selectable on-chip 32kHz oscillator or external 50/60Hz input
- Four 16-bit counter/timers with prescalers and direct input/output drive
- Watchdog Timer with internal oscillator clocking option
- 32 bits of general-purpose input/output (GPIO)

Discussion

Computer networking is an essential part of today's technology. Such networks allow computers and other electronic devices to share information to one another. The Internet is the most popular example of computer networking; it connects millions of computers around the world. On a smaller scale, a Local Area Network (LAN) connects many devices that are in close proximity, such as in office buildings. The most dominant type of LAN architecture is the Ethernet, which is a networking standard that defines the physical and data link layers of the OSI Model discussed later in this document.

The OSI Model

The Open System Interconnection (OSI) model illustrates how network protocols and equipment should communicate with each other. It consists of seven layers arranged from the lowest layer to the highest layer.

The upper three layers describe application-specific functions such as data formatting and connection management. The Hypertext Transfer Protocol (HTTP) and the Simple Mail

Transfer Protocol (SMTP) are examples of data that reside in the upper layers of the OSI Model. The lower four layers describe network-specific functions such as addressing and routing. The Transmission Control Protocol and Internet Protocol (TCP/IP) and the Ethernet are examples of data that reside in the lower layers of the OSI Model.

The OSI Model is illustrated in Figure 1; each layer is defined in Table 1.

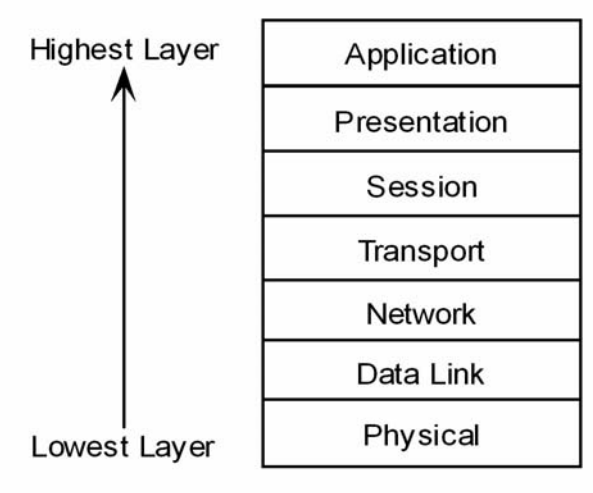

Figure 1. The OSI Model

Table 1. OSI Model: Layer Definitions

Application	Works along with the application software in implementing necessary actions requested by the data it received.
Presentation	Responsible for translating data to/from the application layer from/to a standard format understood by the other lower levels.
Session	Responsible for establishing, maintaining, and ending communication with a remote system.
Transport	Responsible for the dispatch and classification of data sent and received.
Network	Responsible in identifying if the received frame has reached its destination or if the frame requires to be forwarded.
Data Link	Responsible in checking for transmission errors and formats data bits into frames. The data link layer is composed of two sub-layers - the Media Access Control (MAC) and the Logical Link Control (LLC). The MAC is responsible for controlling how other devices on the network gain access to the data it carries. While the LLC is responsible for controlling frame synchronization and flow control.
Physical	Responsible in transmitting/receiving data over the network communications media. Cables, connectors, hubs and repeaters are examples of standard network devices that function at the physical layer.

Ethernet IEEE 802.3 Frame Format

The Ethernet IEEE 802.3 frame defines the frame formats that reside in the physical and data link layers of the OSI Model. The upper five layers of the OSI Model are compiled into the data field of the Ethernet frame format.

An IEEE 802.3 frame consists of a 14-byte header, 46–1500 bytes of data, and an optional 4-byte cyclic redundancy check. The 14-byte header is further subdivided into three parts: a source address, a destination address, and a length field. Figure 2 illustrates this IEEE 802.3 frame format.

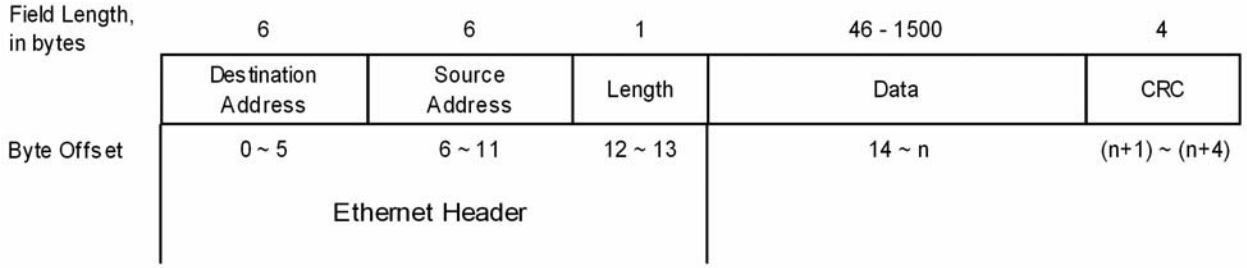

Figure 2. IEEE 802.3 Frame Format

A discussion of the implementation of the upper five layers of the OSI Model is beyond the scope of this application note. For the sake of simplicity, the characters entered by the user in the HyperTerminal window can be used as the contents of the data field portion of the Ethernet frame shown above.

eZ80F91 MCU EMAC

The eZ80F91 MCU EMAC can be viewed in functional blocks, as shown in Figure 3.

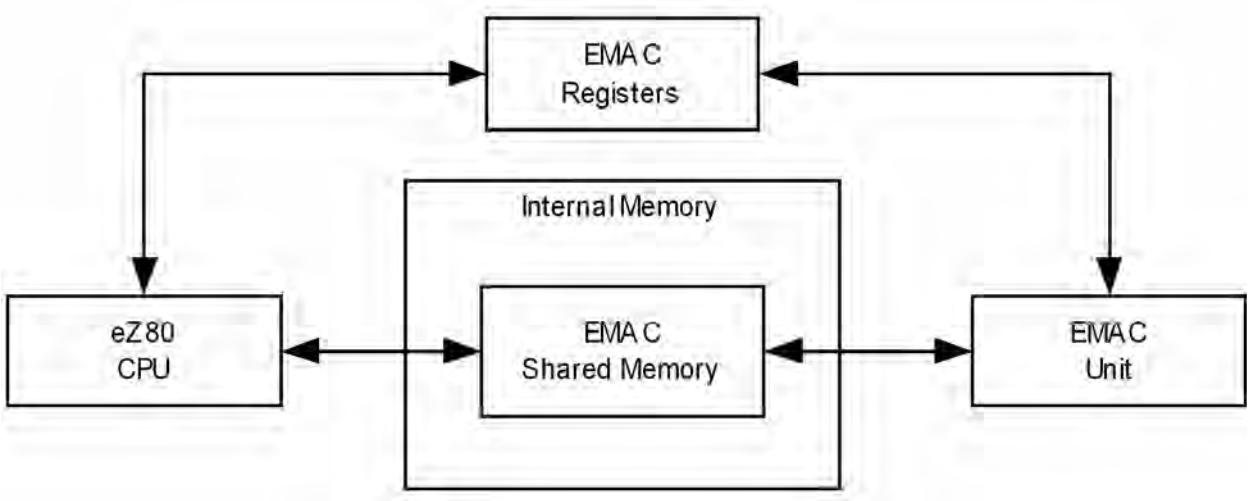

Figure 3. Ethernet Connectivity Blocks on the eZ80F91 MCU

EMAC Registers

Table 2 provides descriptions of the EMAC registers available with the eZ80F91 MCU. To learn more, refer to the EMAC section of the eZ80F91 Product Specification (PS0192).

Table 2. eZ80F91 MCU Registers for EMAC

Register	Description
EMAC Test Register EMAC_TEST	Enables/disables several modes for testing the functionality of the EMAC block.
EMAC Configuration Register 1 EMAC_CGF1	Controls padding, autodetection, cyclic redundancy checking, full-duplex or half-duplex modes, field length checking, maximum packet ignores, and proprietary header options.
EMAC Configuration Register 2 EMAC_CFG2	Controls the functionality of the back pressure and late collision data from the descriptor table.
EMAC Configuration Register 3 EMAC_CFG3	Controls preamble length and value, excessive deferment, and the number of retransmission tries before a packet is aborted.
EMAC Configuration Register 4 EMAC_CFG4	Controls pause control frame functionality, back pressure, and receive frame acceptance.
EMAC Station Address Register EMAC_STAD0 EMAC_STAD1 EMAC_STAD2 EMAC_STAD3 EMAC_STAD4 EMAC_STAD5	The 48-bit station address is represented by {EMAC_STAD_5, EMAC_STAD_4, EMAC_STAD_3, EMAC_STAD_2, EMAC_STAD_1, EMAC_STAD_0}. It is used as the source address when transmitting frames, and compared to destination address when receiving frames.
EMAC Tx Pause Timer Value Register EMAC_TPTV_L EMAC_TPTV_H	Inserted into an outgoing pause control frame as the pause timer value upon asserting a TPCF; e.g., a TPCF is bit 2 of EMAC_CFG4.
EMAC Interpacket Gap Register EMAC_IPGT	Represents the interpacket gap between back-to-back packets used in full-duplex and half-duplex modes.
EMAC Non-Back-to-Back IPG Register EMAC_IPGR1 EMAC_IPGR2	Represents the optional carrier sense window referenced in the IEEE 802.3/4.2.3.2.1 carrier reference.
EMAC Max Frame Length Register EMAC_MAXF_L EMAC_MAXF_H	Determines the maximum length of a frame that can be received.
EMAC Address Filter Register EMAC_AFR	Works as a filter to control Promiscuous Mode and to control multicast and broadcast messaging.

Table 2. eZ80F91 MCU Registers for EMAC (Continued)

Register	Description
EMAC Hash Table Register EMAC_HTBL_0 EMAC_HTBL_1 EMAC_HTBL_2 EMAC_HTBL_3 EMAC_HTBL_4 EMAC_HTBL_5 EMAC_HTBL_6 EMAC_HTBL_7	Represents the hash table matrix that is used as an option to select between different multicast addresses. If a multicast address is received, the first six bits of the CRC are decoded and added to a table that points to a single bit within the hash table matrix. If the selected bit = 1, the multicast packet is accepted; otherwise, the multicast packet is rejected.
EMAC MII Management Register EMAC_MIIMGT	Used to control the external PHY attached to the MII. A rising edge on bit 7 causes EMAC_CTLD control data to be transmitted to an external PHY if the MII is not busy. Similarly, a rising edge on bit 6 causes status to be read from the external PHY via EMAC_PRSD if the MII is not busy.
EMAC PHY Configuration Data Register EMAC_CTLD_L EMAC_CTLD_H	These registers are loaded with data to be sent via the MDIO pin to the PHY. The PHY is selected by setting the EMAC_FIAD. The register inside of the PHY is selected by setting EMAC_RGAD. When bit 7 of EMAC_MIIMGT sees a rising edge, it causes data to be transferred from EMAC_CTLD to the PHY.
EMAC PHY Address Register EMAC_RGAD	Used to select a specific register inside the external PHY.
EMAC PHY Unit Select Address Register EMAC_FIAD	Used to select the external PHY.
EMAC Transmit Polling Timer Register EMAC_PTMR	Used to set the value after which the EMAC checks the transmit descriptor table in EMAC memory to determine if it owns any of the packets in EMAC shared memory.
EMAC Reset Control Register EMAC_RST	Used to reset EMAC functionality.
EMAC Tx Lower Boundary Pointer Register EMAC_TLBP_L EMAC_TLBP_H	Sets the start of transmit buffer in EMAC shared memory.
EMAC Boundary Pointer Register EMAC_BP_L EMAC_BP_H EMAC_BP_U	Sets the start of the receive buffer (i.e., end of transmit buffer + 1) in EMAC shared memory.
EMAC Rx High Boundary Pointer Register EMAC_RHBP_L EMAC_RHBP_H	Must be set to the end of the receive buffer + 1 in EMAC shared memory.
EMAC Rx Read Pointer Register EMAC_RRP_L EMAC_RRP_H	Points to the address location from which the next receive packet must be read. Must be initialized to the EMAC_BP value.

Table 2. eZ80F91 MCU Registers for EMAC (Continued)

Register	Description
EMAC Buffer Size Register EMAC_BUFZ	Used to control the size of buffers in EMAC shared memory and the level at which a pause control frame is to be transmitted.
EMAC Interrupt Enable Register EMAC_IEN	Used to enable the interrupts related to the EMAC.
EMAC Interrupt Status Register EMAC_ISTAT	Used to determine the status of multiple interrupts. It is cleared by writing 1 into the selected bit.
EMAC PHY Read Status Data Register EMAC_PRSD_L EMAC_PRSD_H	The status read by the EMAC from the external PHY is stored in this register.
EMAC MII Status Register EMAC_MIISTAT	The current state of the PHY is stored in this register.
EMAC Receive Write Pointer Register EMAC_RWP_L EMAC_RWP_H	A read-only register that reports the current RxDMA receive write pointer. This pointer is initialized to EMAC_TLBP whenever the EMAC is reset.
EMAC Tx Read Pointer Register EMAC_TRP_L EMAC_TRP_H	A read-only register that reports the current TxDMA transmit read pointer. This pointer is initialized to EMAC_TLBP whenever the EMAC is reset.
EMAC Rx Blocks Left Register EMAC_BLKSLFT_L EMAC_BLKSLFT_H	A read-only register that reports the number of buffers remaining in Rx EMAC shared memory.
EMAC FIFO Data Register EMAC_FDATA_L EMAC_FDATA_H	Allows the writing and reading of the FIFO during EMAC_TEST.
RAM Control Register RAM_CTL	Enables or disables internal RAM.
RAM Address Upper Byte Register RAM_ADDR_U	Defines the upper byte of the address for internal RAM.

EMAC Shared Memory

The eZ80 CPU features 8KB of internal RAM for general-purpose use, and 8KB of internal RAM for EMAC use. Each can be enabled or disabled, and can altogether be relocated anywhere within the address space. When enabled, general-purpose RAM occupies addresses in the range {RAM_ADDR_U, E000h} to {RAM_ADDR_U, FFFFh}, while EMAC RAM occupies addresses in the range {RAM_ADDR_U, C000h} to {RAM_ADDR_U, DFFFh}. When using the EMAC, it is mandatory to at least enable EMAC RAM and set RAM_ADDR_U.

The EMAC shared memory (or EMAC RAM) is used by both the eZ80 CPU and the on-chip EMAC unit. This memory is divided into two parts: the transmit buffer and the receive buffer. The bounds of these two buffers are defined by three registers, as follows.

Transmit Lower Boundary Pointer (EMAC_TLBP). Defines the start address of the transmit buffer.

Boundary Pointer (EMAC_BP). Defines the start address of the receive buffer.

Receive High Boundary Pointer (EMAC_RHBP). Defines the end address of the receive buffer + 1.

Figure 4 illustrates how these three buffers are used to divide eZ80F91 MCU internal EMAC RAM.

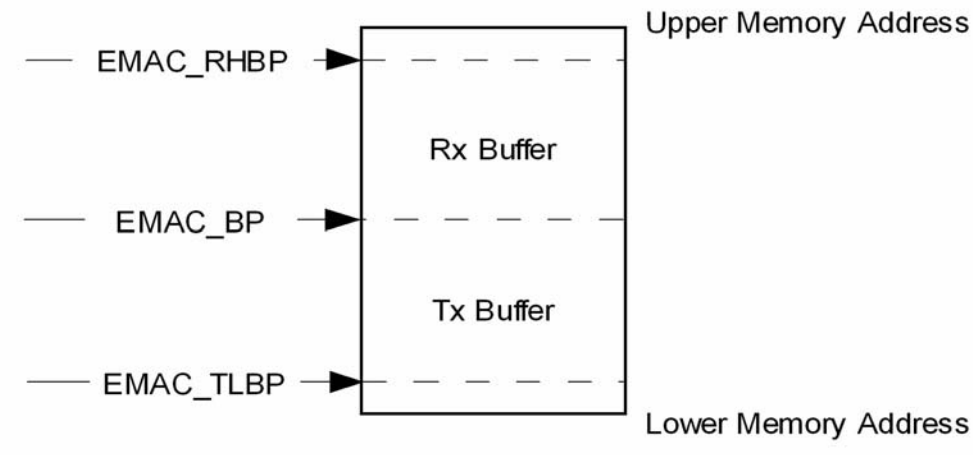

Figure 4. EMAC Shared Memory Organization

The eZ80 CPU and the on-chip EMAC uses this area of internal RAM exclusively as a storage medium for packets to be transmitted and for packets received. These Tx and Rx buffers are further subdivided into packet buffers with configurable buffer sizes of 32, 64, 128, or 256 bytes, as defined by EMAC_BUFZ. Each packet buffer can be accessed using the following four pointers.

Transmit Write Pointer. Defines the address of the next available packet buffer in the Tx buffer into which the software can place a new packet that is waiting for transmission. Initially, this pointer must be set equal to the value of EMAC_TLBP. This pointer, `_asEMAC_twp`, is defined by software because it is only relevant within the context of the software, which is responsible for using and updating this pointer value as it adds packet buffers into the Tx buffer.

Transmit Read Pointer (EMAC_TRP). Defines the address of the current packet in the Tx buffer that is waiting to be transmitted by the EMAC. When EMAC_RST[SRST or HRRTN] are set, this pointer is reset to the value of EMAC_TLBP. A copy of this pointer, `_asEMAC_trp`, is maintained by the software.

Receive Write Pointer (EMAC_RWP). Defines the address of the next available packet buffer in the Rx buffer into which the EMAC can place newly-received packets. When EMAC_RST[SRST or HRRTN] are set, this pointer is reset to the value of EMAC_TLBP. A copy of this pointer, `_asEMAC_rwp`, is maintained by the software.

Receive Read Pointer (EMAC_RRP). Defines the address of the current packet in the Rx buffer that is waiting to be read by software. Initially, this pointer must be set equal to the value of EMAC_BP. A copy of this pointer, `_asEMAC_rrp`, is maintained by the software.

A packet buffer consists of two major blocks, the descriptor table and the Ethernet packet, as shown in Figure 5. The block containing the Ethernet packet (or Ethernet frame) follows a standard format defined by the IEEE 802.3 specification. A brief overview is discussed in the Ethernet IEEE 802.3 Frame Format section on page 4.

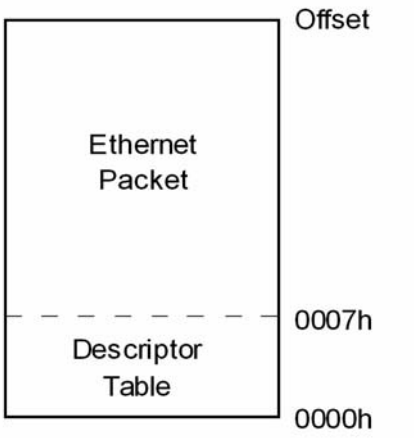

Figure 5. Contents of a Packet Buffer

The block containing the descriptor table describes the packet it handles. It contains three entries: the next pointer (`np`), the packet size (`pktsz`), and the packet status (`stat`), as shown in Figure 6. `np` is a 24-bit pointer to the start of the next packet, `pktsz` contains the number of bytes of data in the Ethernet packet, and `stat` contains the status of the packet. The packet status definitions for `stat` are different for transmit and receive packets.

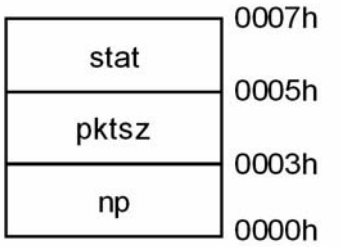

Figure 6. Descriptor Table Entries

Table 3 describes the status of all transmit descriptors. The most important bit in the transmit descriptor status is TxOwner, bit 15. This bit must be written by software to start/stop EMAC transmission of the affected packet. While preparing data to be transferred, this bit

must be set to 0 (i.e., `Host owns`). Setting this bit to 0 signals the EMAC that the packet is not yet ready for transmission. To start transmission, this bit must be set to 1 (i.e., `EMAC owns`). Setting this bit to 1 signals the EMAC that the packet can already be sent out. The remainder of the bits reflect packet status upon transmission; there is no requirement for software to modify these bits. When the EMAC has completed transmitting a packet, it automatically updates the value of its transmit descriptor status.

Table 3. Transmit Descriptor Status

Bit	Name	Description
15	TxOwner	0 = Host (eZ80) owns; 1 = EMAC owns.
14	TxAbort	1 = Packet aborted (not transmitted).
13	TxBPA	1 = Back pressure is applied.
12	TxHUGE	1 = Packet size is greater than maximum frame length.
11	TxLOOR	1 = Type/length field out of range (larger than 1518 bytes).
10	TxLCError	1 = Type/length field is not a Type field and it does not match the actual data byte length of the Ethernet packet.
9	TxCrcError	1 = Packet contains an invalid CRC. This flag is set when CRCEN=0 and the last 4 bytes of the packet are not the valid CRC.
8	TxPktDeferred	1 = Packet is deferred.
7	TxXsDfr	1 = Packet is excessively deferred.
6	TxFifoUnderrun	1 = TxFIFO has underrun.
5	TxLateCol	1 = Late collision has occurred.
4	TxMaxCol	1 = Maximum collisions have occurred.
3:0	TxNumberOfCollisions	This field contains the number of collisions that occurred while transmitting the packet.

Table 4 describes the status of all receive descriptors. This field is written by the EMAC when the packet has been completely written into the receive buffer.

Table 4. Receive Descriptor Status

Bit	Name	Description
15	RxOK	1 = Packet received.
14	RxAlignError	1 = Odd number of nibbles received.
13	RxCrcError	1 = The CRC is in error.
12	Rx:LongEvent	1 = A long or dropped event has occurred.
11	RxPCF	1 = The packet is a pause control frame.
10	RxCF	1 = The packet is a control frame.
9	RxMcPkt	1 = The packet has a multicast address.
8	RxBcPkt	1 = The packet has a broadcast address.
7	RxVLAN	1 = The packet is a VLAN packet.

Table 4. Receive Descriptor Status (Continued)

Bit	Name	Description
6	RxUOpCode	1 = Unsupported Op Code is indicated in the Op Code field of the Ethernet packet.
5	RxLOOR	1 = The type/length field is out of range (larger than 1518 bytes).
4	RxLCError	1 = The type/length field is not a Type field and it does not match the actual data byte length of the Ethernet packet.
3	RxCodeV	1 = A code violation is detected.
2	RxCEvent	1 = A carrier event is previously seen.
1	RxDvError	1 = A rxdv event is previously seen.
0	RxOVR	1 = Receive overrun has occurred.

Hardware Architecture

Two eZ80F91 MCU development boards are connected together by a crossover cable. To act as user interface, each of these boards is connected to a PC running HyperTerminal over an RS232 cable. If no COM port is available, a USB-to-serial cable can also be used with the eZ80F910300KITG hardware instead of the RS232 cable. Figure 7 shows a block diagram of the connections used in this application, which has been developed and tested using three development boards provided by Zilog. Therefore, the source code that supports this application can be used in any of these development boards and in any combination.

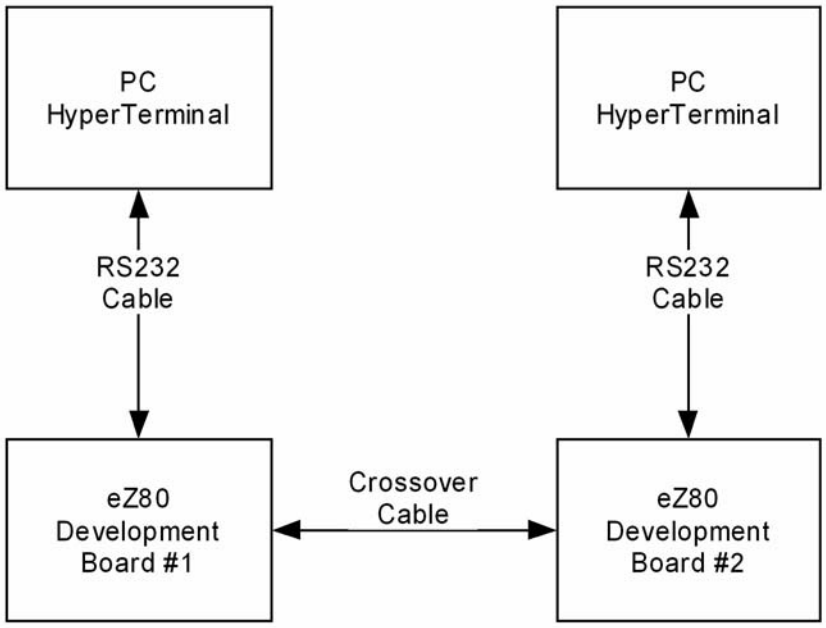

Figure 7. System Block Diagram

eZ80F91 MCU Development Boards

As indicated in the preceding note, three development boards are available for the eZ80F91 MCU, represented by part numbers eZ80F910300ZCOG, eZ80F910200KITG, and eZ80F910300KITG. Each of these development boards contains a number of peripheral devices, including memory and connectors, to aid developers in evaluating the eZ80Acclaim! and eZ80Acclaim*Plus!* family of devices for a number of applications. A console port on each of these development boards can be used to connect a serial cable to a computer to provide a user interface via the HyperTerminal program.

eZ80F910300ZCOG

The eZ80F910300ZCOG Development Board provides a general-purpose platform for evaluating the capabilities and operation of the eZ80F91 MCU. This development board contains two primary boards: the eZ80 Evaluation Platform (99C0858-001G) and the eZ80F91 Module (99C0879-001G or 99C1380-001G). The main features of the eZ80F910300ZCOG Development Board relevant to this application include:

- 512KB SRAM via CS2
- 512KB SRAM via CS1
- 1MB Flash memory via CS0
- An eZ80F91 device operating at 50MHz, with 256KB of internal Flash memory and 8KB of internal SRAM
- An RS232 console connector with an interface circuit for UART0
- On-chip Ethernet Media Access Controller (EMAC)
- An Ethernet port and AMD NetPHY AM79C874 (when using the 99C0879-00xG module)
- An Ethernet port and IDT PHY ICS1894 (when using the 99C1380-001G module)

eZ80F910200KITG

The eZ80F910200KITG Development Board provides a set of tools for designing an application based on the eZ80F91 MCU. This development board contains two primary boards: the eZ80Acclaim! MDS adapter board (98C0945-001) and the eZ80F91 Mini Ethernet Module (99C0942-001G). The main features of the eZ80F910200KITG Development Board relevant to this application include:

- 128KB SRAM via CS1
- An eZ80F91 device operating at 50MHz, with 256KB of internal Flash memory and 8KB of internal SRAM
- An RS232 console connector with an interface circuit for UART0
- On-chip Ethernet Media Access Controller (EMAC)
- An Ethernet port and Micrel PHY KS8721

eZ80F910300KITG

The eZ80F910300KITG Development Board provides a general-purpose platform for creating a design based on the eZ80F91 MCU. The main features of the eZ80F910300KITG Development Board relevant to this application include:

- 512 KB SRAM via CS1
- 512 KB SRAM via CS2
- 8 MB Flash memory via CS0
- An eZ80F91 device operating at 50 MHz, with 256 KB of internal Flash memory and 8 KB of internal SRAM
- A USB interface that provides power to the board and a connection to the MCU's UART0 block
- An on-chip Ethernet Media Access Controller (EMAC)
- An Ethernet port and IDT PHY ICS1894

Software Implementation

This application utilizes the EMAC peripheral to enable Ethernet connectivity on the eZ80F91 MCU. Data transmission over the Ethernet medium can be divided into the following three main blocks, each of which is described below.

- Ethernet Initialization
- Ethernet Frame Transmission
- Ethernet Frame Reception

Ethernet Initialization

To enable Ethernet functions in the eZ80F91 MCU, three items must be initialized: the EMAC, the EMAC shared memory, and the external PHY. Ethernet initialization is performed by the `sETH_Init()` routine. This routine requires two parameters: a pointer to the location where MAC address is stored, and a pointer to the function to be executed when EMAC Rx interrupt occurs. After initialization, frame Ethernet transmission and reception can be performed. To initialize the Ethernet, `sETH_Init()` performs the following sequence of steps:

1. Initialize EMAC RAM that can be used by the EMAC shared memory later. This application reserves internal RAM address starting at `FFC000h` up to `FFE000h` for EMAC shared memory.
2. Place the EMAC into a reset state (EMAC_RST = `0x3Fh`) to temporarily disable EMAC operation caused by the EMAC data lines.
3. Set up an EMAC Station Address which will be the device MAC address, and which must be unique for each device. For demonstration purposes, this application uses a fixed MAC address (00:90:23:AB:CD:EF) defined by aucMAC_Address[6] in the `main.c` file.

4. Specify an EMAC buffer size (EMAC_BUFSZ) to determine the size of buffers stored in EMAC shared memory. This application sets buffer size to the minimum value (32 bytes) to maximize the use of this shared memory. This value can be changed by modifying the value of the `bufSize` element of `MAC_F91Config struct`.
5. Set the transmit pause timer value (EMAC_TPTV) to `1400h` to set pause duration when EMAC buffers are full.
6. Locate boundary pointers EMAC_TLBP, EMAC_BP, EMAC_RHBP, and EMAC_RRP within the shared memory and create pointers to these addresses.
7. Set the transmit polling timer value (EMAC_PTMR) to 1 to reflect the minimum time between successive polls of the EMAC shared memory.
8. Disable EMAC test modes by setting EMAC_TEST to zero.
9. Configure EMAC functionality by writing to the EMAC configuration registers.
10. Set the Address Filter (EMAC_AFR) Register to allow broadcast and multicast addresses. Although multicast addresses are allowed in this application, hash table entries (EMAC_HTBL) must still be configured by the user to fully support multicast addresses.
11. Specify the maximum frame length for frames to be received in EMAC_MAXF.
12. Reenable EMAC operation by setting EMAC_RST = `0x00`.
13. Set the MII management clock to enable external PHY access.
14. Initialize the external PHY. The external PHY initialization is handled by the `sPHY_Init()` routine . To initialize the external PHY, perform the following steps .
    a. Inform the EMAC of the address of the external PHY attached to it by setting the EMAC PHY unit select address, EMAC_FIAD, to the address of the external PHY. A discussion of this external PHY addressing can be determined in its corresponding data sheet.
    b. Determine if the PHY is detected and is correct by reading the values stored in the PHY_ID1_REG and PHY_ID2_REG registers of the external PHY. Values read from these registers must correspond to the values specified in the PHY's corresponding data sheet.
    c. Reset the PHY by writing to PHY_CREG.
    d. Determine if autonegotiation is enabled by reading PHY_SREG.
    e. Establish a link according to user-defined options in the `MAC_F91Config.mode` file if Autonegotiation Mode is disabled.
15. Clear the Interrupt Status (EMAC_ISTAT) Register to ensure that there are no pending interrupts to be executed.
16. Enable the EMAC Tx and Rx interrupts.

Ethernet Frame Transmission

Ethernet frame transmission is the process by which the MCU sends a data packet to another device via the Ethernet medium. This transmit process consists of the following steps:

1. Generate a frame for transmission. The Ethernet packet is defined by `_tETH_PKT`. Table 5 lists the information that is stored in a typical Ethernet packet.

Table 5. Information of Ethernet Packet

Packet	Description
ep_order	Byte order mask for debugging.
ep_len	Total length of the packet, in bytes, from eh_dst up to ep_data.
eh_dst	Destination address (see Figure 2).
eh_src	Source address (see Figure 2).
eh_type	Length of ep_data, in bytes (see Figure 2).
ep_data[]	Ethernet data field (see Figure 2).

For this application, the data field of the Ethernet frame, `ep_data[]`, can come from a user input via the HyperTerminal. If the length of this user input is less than the minimum length allowed (ETHPKT_MINDLEN), a series of *x* characters are appended to the data field. This step is handled by the `SendData()` routine found in the `main.c` file.

2. Determine if the generated frame is within the total maximum length limits of a frame, as illustrated in Figure 2. In essence, the total frame length must be less than or equal to ETHPKT_MAXDLEN + ETH_HLEN, or 1514 bytes. This determination is performed in the `sETH_TransmitPkt()` routine found in the `emac.c` file. If the generated frame exceeds the maximum size allowed for an Ethernet packet, the frame can be rejected, and no further processing will occur.
3. Predetermine the location of the next packet in EMAC shared memory. This next packet pertains to a frame which can immediately follow the current frame that can be added to memory in succeeding steps.
4. Set the status of the next packet to HOST OWNS to prevent the EMAC from sending it.
5. Generate entries for the Tx buffer descriptor of the current frame. These entries can fill up the descriptor table portion of the Tx packet buffer presented in Figure 5 on page 9.
6. Copy the Ethernet packet into the Tx buffer of the EMAC shared memory. This data can fill up the Ethernet packet portion of the Tx packet buffer presented in Figure 5.
7. Enable EMAC interrupts.
8. Start transmitting the frame by setting the status of the current frame to EMAC OWNS.

9. Update the transmit write pointer, `_asEMAC_twp`, to the value of the predetermined location of the next packet acquired in Step 3.
10. As soon as Step 8 is performed, the EMAC takes control toward handling the remainder of the operation. When the EMAC completed transmission, it updates the frame by writing to the status field of the current Tx buffer descriptor (see Table 3 on page 10). Next, a Transmit Done (TX_DONE_STAT) interrupt occurs. Other transmitter-related interrupts include the transmit control frame (TX_CF_STAT) and transmit state machine error (TXFSMERR_STAT) interrupts. This application only handles the TX_DONE_STAT interrupt. In the interrupt routine, perform the following steps:
    a. Clear TX_DONE_STAT by writing 1 to the EMAC_ISTAT Register.
    b. Verify that the current frame was transmitted by checking TxAbort (bit 14) of the status field in the Tx buffer descriptor. If the packet is not transmitted, the application must handle this transmission, usually by resending the packet. For this application, further processing stops when a packet is aborted.

Ethernet Frame Reception

Ethernet frame reception is the process by which the MCU receives a data packet from another device via the Ethernet medium. Frame reception starts when the EMAC receives a frame from the Ethernet. The EMAC handles the process of storing this received data in EMAC shared memory. When a complete packet has been transferred into memory, the EMAC generates a Receive Done (RX_DONE_STAT) interrupt. Other interrupts include the Receive Pause, Receive Overrun, and Receive Control packets. Each interrupt source must be handled according to application requirements. For this application, only the Receive Done interrupt is being handled, and is therefore enabled.

The interrupt routine handles these received packets by observing the following steps:

1. Ensure that a new packet has already been added to memory.
2. Clear the RX_DONE_STAT interrupt by writing a 1 to the EMAC_ISTAT Register.
3. Process the data packet received. For this application, the received data packet is decoded and displayed in HyperTerminal. This process is performed by the `ETH_ReceivePkt()` routine in `emac.c`.
4. Update the Receive Read pointer, `_asEMAC_rrp`, to point to the next available packet buffer in EMAC shared memory.
5. Update the EMAC Receive Read Pointer Register, EMAC_RRP, to point to the next available packet buffer in EMAC shared memory. EMAC_RRP and `_asEMAC_rrp` must always be synchronized together.

Testing/Demonstrating the Application

Observe the following procedure to test/demonstrate the Ethernet Frame Transmission application.

1. Set up hardware connections as shown in Figure 7 on page 11. If using a single PC, connect both development boards to the PC using two serial cables (or USB-to-serial

cables), then open two HyperTerminal applications. HyperTerminal settings must be 57600bps, 8 data bits, 1 stop bit, and no parity.

2. In ZDSII – eZ80Acclaim!, open the `AN0212.zdsproj` project file.
3. From the **Build** menu, select **Set Active Configuration**; the Select Configuration Dialog will appear. Choose the appropriate build configuration for the development board you are working with. Table 6 lists the available development boards and the appropriate build configurations for each board. For the application to run without ZDSII, select the appropriate release configuration. The debug configuration loads the code into the RAM area only; therefore, the application cannot run when removed from ZDSII.

Table 6. Build Configurations for Each Development Board

Development Board	Build Configuration
eZ80F910300ZCOG	Debug_DevPlatform_RAM
	Release_DevPlatform_FLASH
eZ80F910200KITG	Debug_ModDevKit_RAM
	Release_ModDevKit_FLASH
eZ80F910300KITG	Debug_300KITG_RAM
	Release_300KITG_FLASH

Note: When using the eZ80F91x150MODG module together with the eZ80F910300ZCOG evaluation platform, use the `ICS1894_phy.h` header file instead of the `AMD79C874_phy.h` header file. To learn more, refer to the Zilog application note titled Porting RZK/ZTP Applications to the eZ80F91x150MODG Module (AN0362).

4. Click **Rebuild All**, then download the code.
5. Repeat steps 3 and 4 to configure the second development board.
6. Reset both boards. Each of the HyperTerminal applications will display a welcome message, as shown in Figure 8.

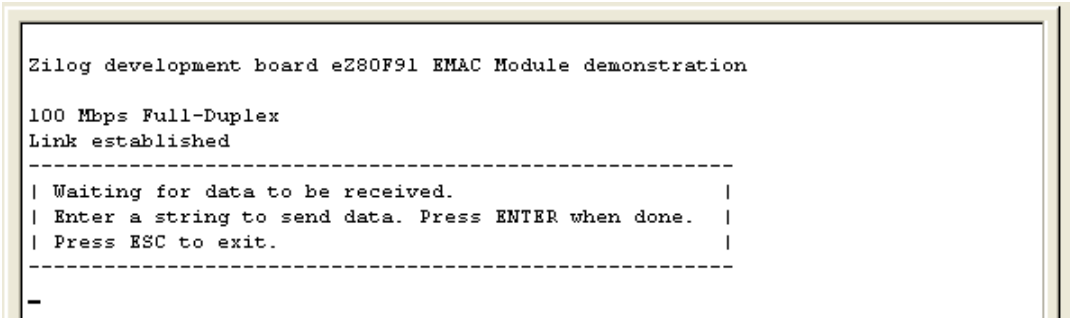

Figure 8. Hyper Terminal Welcome Message

7. On Board #1, enter any string in the HyperTerminal window and press the Enter key on your keyboard. As a result, Board #2 should be able to receive and display data in its HyperTerminal window.

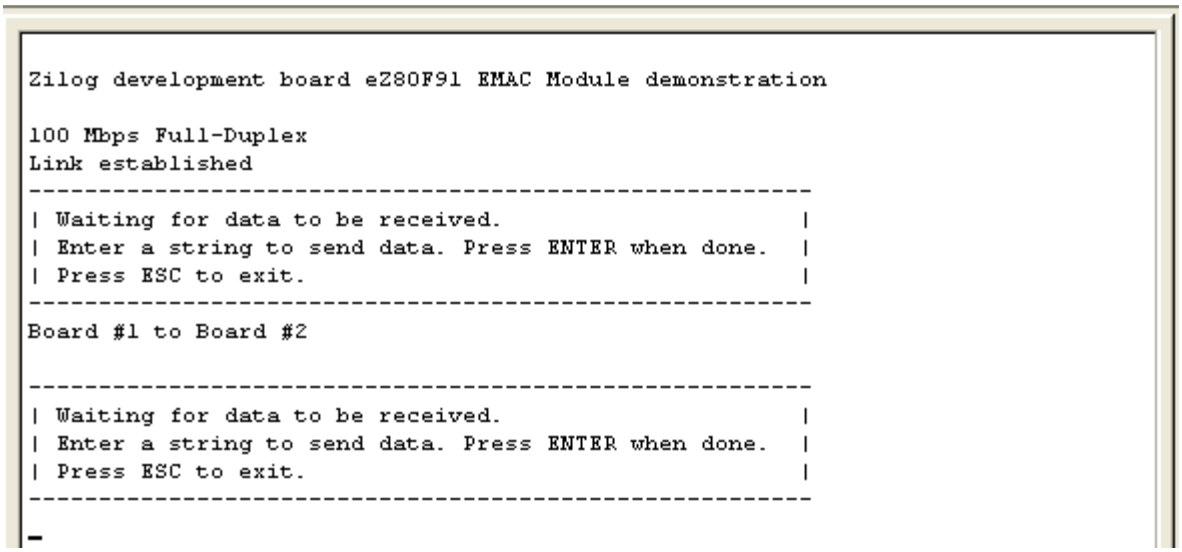

Figure 9. HyperTerminal Window for Board #1

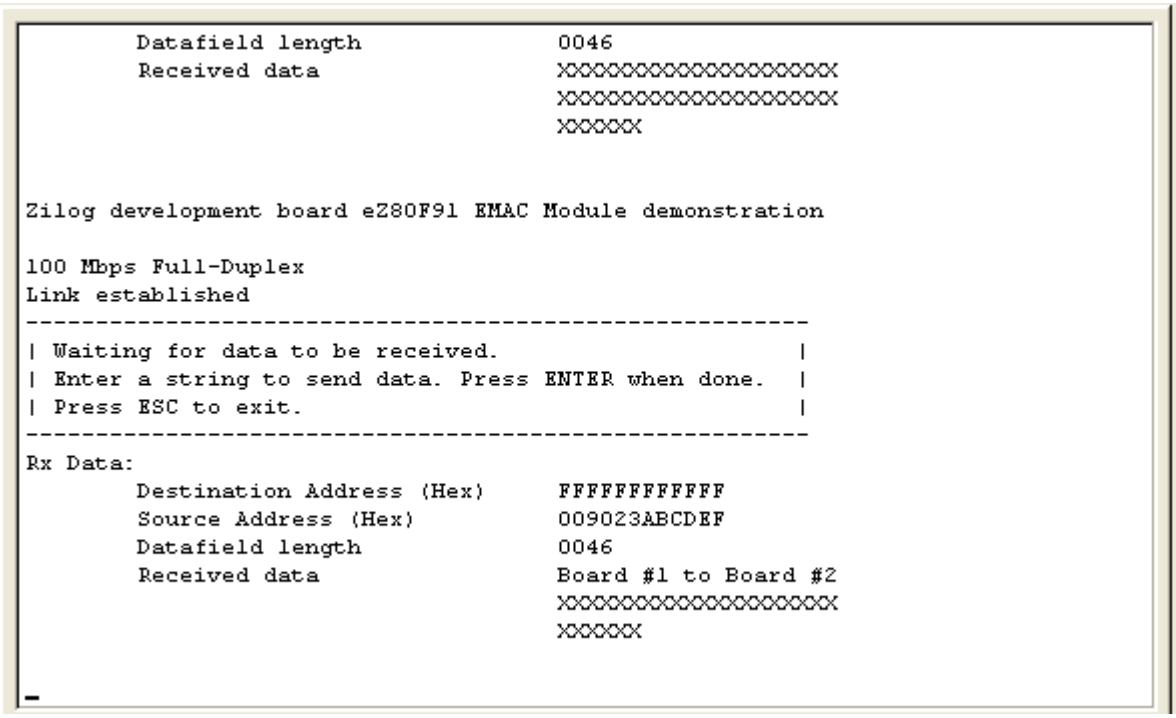

Figure 10. HyperTerminal Window for Board #2

8. Repeat Step 7, but this time, enter a string in the HyperTerminal window of Board #2. The HyperTerminal window for Board #1 should display the received data.

Equipment Used

The tools used to create and test this application are:

- ZDSII – eZ80Acclaim! v5.2.1
- Two eZ80F91 development boards
- One or two USB SmartCables
- Crossover cable
- Two RS232 cables (or USB-to-serial cables)

Summary

The Ethernet Frame Transmission application discussed in this document demonstrates how to use the eZ80F91 MCU to send and receive data across the Ethernet medium. By using the Ethernet IEEE 802.3 frame format, the basic principles of a number of TCP/IP protocols such as UDP, PPP, etc are introduced.

References

The documents listed below are associated with the eZ80F91 MCU; each is available free for download from the Zilog website.

- eZ80F91 Product Specification (PS0192)
- eZ80F91 ASSP Product Specification (PS0270)
- eZ80F91 Modular Development Kit (eZ80F910200KITG) User Manual (UM0170)
- eZ80F91 Development Kit (eZ80F910300ZCOG) User Manual (UM0142)
- eZ80F91 Development Kit (ez80F910300KITG) User Manual (UM0244)
- Zilog Developer Studio II – eZ80Acllaim! User Manual (UM0144)
- Porting RZK/ZTP Applications to the eZ80F91x150MODG Module Application Note (AN0362)
- eZ80Acclaim!/eZ80AcclaimPlus! Ethernet Modules Product Specification (PS0306)

The following documents are published externally, and are available on their respective websites.

- IEEE 802.3 Standard
- AMD AM79C874 PHY Datasheet
- Micrel KS8721 PHY Datasheet
- IDT ICS1894 PHY Datasheet

Customer Support

To share comments, get your technical questions answered, or report issues you may be experiencing with our products, please visit Zilog's Technical Support page at http://support.zilog.com.

To learn more about this product, find additional documentation, or to discover other facets about Zilog product offerings, please visit the Zilog Knowledge Base at http://zilog.com/kb or consider participating in the Zilog Forum at http://zilog.com/forum.

This publication is subject to replacement by a later edition. To determine whether a later edition exists, please visit the Zilog website at http://www.zilog.com.

Warning: DO NOT USE THIS PRODUCT IN LIFE SUPPORT SYSTEMS.

LIFE SUPPORT POLICY

ZILOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZILOG CORPORATION.

As used herein

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Document Disclaimer

 Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZILOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZILOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. The information contained within this document has been verified according to the general principles of electrical and mechanical engineering.