eZ80 Webserver

Write your own EMAC Driver for Metro IPWorks™

TA000303-0602 **Technical Article**

Driving an Ethernet MAC

The eZ80 Webserver is equipped with two Ethernet Media Access Controller (EMAC) drivers that control the Realtek RTL8019AS and the Cirrus Logic CS8900A Crystal EMAC devices. These two EMACs are connected to the eZ80 Webserver via the I/O address space and one of the GPIO pins used to receive external interrupts from the eZ80 Webserver device. The source code for these drivers is provided as example EMAC driver code in the Device Driver Kit (DDK).

The two EMAC drivers are contained in the following files:

Device	File
Realtek 8019	rt8019.c
Crystal 8900	cs8900.c

Each of these drivers provides an interface between the EMAC device and the Metro IPWorks™ stack. Each includes an Ethernet Interrupt Service Routine (ISR) to manage EMAC interrupts. The Metro IPWorks™ TCP/IP stack uses the XINU **open**, **control**, **read**, **write**, **close** model for interfacing to the drivers. Metro IPWorks™ has enhanced this model by adding an initialization interface and omitting the **open** function[1].

The general approach to writing your own EMAC driver for Metro IPWorks™ is to write driver code with a template file (`MAC_Template.c`, provided in the DDK). This template file contains a skeleton of required routines that can be used as code examples. These skeleton routines use the same names as those in the driver files provided. The `MAC_Template.c` template is listed in the Appendix to this document.

The user EMAC driver file, along with the EMAC library, `emac.lib`, is added to the ZDS project source files, where these routines are linked into the project build.

Note: The `ez80cs.lib` or `ez80rt.lib` files from the original Metro IPWorks™ must be removed before the build.

Required Routines

EMAC driver code must be written for the following four routines.

- EthInitFunc
- ReceivePkt
- TransmitPkt
- EthISR

The following paragraphs describe the role of each of these four functions. References in this Technical Article are made to sections in the eZ80 Webserver Programmer's Guide by their verbatim names.

[1]When it is initialized, the EMAC operates continuously, thereby making the **open** function unnecessary.

Initializing the EMAC Device

When the **init_ether** function is called (see `netconfig.c` in the *Network Configuration*), Metro IPWorks™ adds entries to the **devsw** table for the EMAC device (see *How to Use Other eZ80190 Devices*) using the **adddevice** function. **init_ether** then calls the initialization routine identified in the **devsw** table. The initialization routine calls the **EthInitFunc** and returns. The **init_ether** function creates a thread that indefinitely calls the **read** function identified in the **devsw** table. The **read** function checks whether a packet is available on an internal queue. If a packet is available, it is returned to the calling function; otherwise the current thread is suspended.

As can be seen from the sequence, the **EthInitFunc** function must initialize the EMAC device. As shown in the Appendix, **EthInitFunc** calls the **set_evec** function (see the *How to Use Other eZ80190 Devices*), which maps the interrupt vector to the EMAC ISR function **EthISR**. **EthInitFunc** must also save the **txnotify** and **rxnotify** callback addresses passed as input parameters. These callback addresses are saved as functions **txnotifyfunc** and **rxnotifyfunc**, as shown in the example EMAC driver code.

The functions represented by these callback addresses are not available to the user. Therefore, the EMAC driver must use the appropriate callback address when a hardware event indicates that a packet is available for reception, or when the hardware indicates that a transmit operation is complete.

The **EthISR** function manages the transmission and reception of packets. When an EMAC Receive interrupt occurs, the **ReceivePkt** routine is called via **rxnotifyfunc**. When an EMAC transmit interrupt occurs, **EthISR** must call **txnotifyfunc** (a saved parameter from **EthInitFunc**). The **txnotifyfunc** function is also not available to the user.

As noted above for packet Receives, Metro IPWorks™ sets up a standing call to the **read** function in a thread created during initialization. When **EthISR** determines that the hardware has received a packet, it should call the **rxnotifyfunc** callback. This routine performs some internal processing and calls the **ReceivePkt** routine. This routine reads packet data from the hardware registers and stores it in a software packet structure that is created by **rxnotifyfunc**. The **rxnotifyfunc** then places the thread, suspended on the **read** call, into the Ready state. The **read** function is then allowed to pass the received data to the calling process.

Transmitting Packets to the EMAC

When Metro IPWorks™ is ready to write to the EMAC, it calls the **write** function. The **write** function calls the EMAC **write** routine from the **devsw** table. The EMAC **write** routine then calls the **TransmitPkt** function. Special attention must be focused on the value returned by the **TransmitPkt** function. The four possible return values are:

- TX_FullBuf
- TX_Waiting
- PKTTooBig
- TX_Done

The choice of which value to return depends on whether the **TransmitPkt** routine is able to send the data; and if so, whether this transfer is completed synchronously or asynchronously. In cases where an internal

transmit queue is implemented and the queue is full at the time **TransmitPkt** is called, the **TransmitPkt** routine must return **TX_FullBuf**. A full buffer situation causes upper layers of Metro IPWorks™ to discard the packet.

Note: A full buffer can cause performance problems, because upper-layer protocols can take varying amounts of time to acknowledge a lost packet. Acknowledging a lost packet may not even be possible for UDP-based applications.

If the size of a packet is larger than can be accommodated on the physical link, the **TransmitPkt** routine returns a value of **PKTTooBig**.

If the **TransmitPkt** routine is able to completely send the requested data before returning (that is, a synchronous transmission), it returns **TX_Done**.

If the **TransmitPkt** routine merely places a packet on its own internal packet queue (because it is busy transmitting an earlier packet) for transmission at a later time, the **TransmitPkt** routine returns **TX_Waiting**. In this case, the transmission of the packet is handled asynchronously.

Note: When an asynchronous Transmit event completes, the **write** function should call the **txnotifyfunc** routine to inform the upper layers of Metro IPWorks™ that the transmit operation has completed. Although **txnotifyfunc** currently does not perform any useful tasks, future versions of Metro IPWorks™ will release resources associated with this Transmit function. Therefore, to ensure forward compatibility, **txnotifyfunc** should be called when all asynchronous transmissions are completed.

Additional Driver Functions

The EMAC driver also contains multicast functions used by the IGMP protocol. Metro IPWorks™ calls an **EthMaddFunc** routine to add a multicast address to the table of multicast addresses that the driver maintains in hardware and/or software. This routine is called when Metro IPWorks™ calls the EMAC **control** function with a function parameter set to add a multicast address. It is not essential that the EMAC driver implement a multicast function. Without a multicast function, however, IGMP may not be able to receive packets. In effect, IP multicasting on the embedded webserver is disabled.

The remaining routines in the `rt8019.c` and `cs8900.c` files can be used as example code that can be useful to the developer.

Appendix

EMAC Driver Template (`MAC_Template.c`)

```
/*
 * Copyright 2002, ZiLOG, Inc.
 * All Rights Reserved
 */

/*

*********************************************************************
****
 *                                                                   *
 * Module:     MAC_Template.C                                        *
 * Description: Sample template file for the HW specific EMAC driver.
*
 *                                                                   *
 * Revision Log:                                                     *
 * Date    Who   What                                                *
 * ------   ---   ----                                              *
 * 022022   MCB   Created.                                          *

*********************************************************************
****
 */
#include <kernel.h>
#include <proc.h>
#include <network.h>
#include "emac808.h"



/*

*********************************************************************
****
 *                              Variables                            *

*********************************************************************
****
 */
struct etdev_comm       emac[1];
static EMACFRAME * (* rxnotifyfunc)(void);
static EMACFRAME * (* txnotifyfunc)(void);


/*
 *********************************************************************
 *                              ETHMADDFUNC                          *
 *                                                                   *
 * Ethernet Multicast Add function.                                  *
 *                                                                   *
 * Description: This function is called to add the given Multicast address*
 * to the multicast table maintained within the MAC.
*
```

```
 *                                                                          *
 * Input:                                                                   *
 * ether_addr - Pointer to the 48-bit MAC address to be added to the        *
 *              multicast table. The address should be checked for          *
 *              validity and uniqueness in the table before being added.    *
 *              If the MAC table is full or the HW is not able to support   *
 *               multicast addresses this routine will silently discard the*
 *                the given address.                                        *
 * Output:                                                                  *
 * <none>                                                          *
 *                                                                          *
 * Notes:                                                                   *
 * In future versions of Metro IPWorks™, this routine will be modified to *
 * return a status code so the calling routine can determine if the     *
 * multicast table was updated.
 * Even if the HW is unable to support Multicast filtering the MAC writer*
 * may choose to implement the multicast table filtering in software. (I.e.*
 * the HW can be placed in promiscuous mode and the Rx routine in this *
 * module can filter packets based on the 48-bit DA field.) However, this *
 * may cause a severe degradation in performance on busy networks and is not
 * advised.*
 ****************************************************************************
 */
void
ETHMADDFUNC
(
   unsigned char * ether_addr
)
{
 /*
  * The sample driver does not maintain a multicast table.
  */
}



/*
 ****************************************************************************
 *                              EthInitFunc                                 *
 *                                                                          *
 * Ethernet Init function.                                                  *
 *                                                                          *
 * Description: This function should verify the presence of the expected HW*
 * controller and claim the appropriate system resources (interrupt vector *
*DMA,I/Opins,timers,etc.)andinitializetheHW/SWtoaknownstate.*
 *                                                                          *
 * Input:                                                                   *
 * ieeeaddr - Pointer to the 48-bit MAC address that this driver should    *
 *              use for multicast frame reception.  This address should be *
 *              programmed into the controller.  Initially the MAC should  *
 *               only enable reception of unicast and broadcast frames.    *
 * rxnotify - The MAC should call this function every time a packet has
 *               been successfully received by the HW to allow the upper*
 *              protocol layers to process the packet.                      *
 * txnotify - The MAC must call this function every time the HW interrupt  *
 *               routine indicates the completion of an asynchronous
 *               transmit.*
```

```
 * Output:                                                                *
 * OK Initialization was completed successfully.              *
 *                                                                        *
 * Notes:                                                                 *
 * The MAC writer is free to define other status codes.  Any status code *
 * other than OK is treated as an error by the upper layer protocols.     *
 * It is assumed that this routine has a means to determine which eZ80    *
 * interrupt is connected to the NIC.  (This may simply be a compile time  *
 * constant).                                                             *
 **************************************************************************
 */
int
EthInitFunc
(
 char *ieeeaddr,
 void *(*rxnotify)(void),
 void *(*txnotify)(void)
)
{
 /*
  * Hold on to the callback addresses do they can be used from the ISR.
  */
 rxnotifyfunc = rxnotify;
 txnotifyfunc = txnotify;

 /*
  * If an interrupt is not going to be used for event processing this
  * routine should create a background thread to check for HW event.  This
  * may have a severe impact on performance.
  */

 return(OK);
}



/*
 **************************************************************************
 *                              TransmitPkt                               *
 *                                                                        *
 * Transmit Packet function.                                              *
 *                                                                        *
 * Description:  The mac library will call this routine when a frame needs *
 * to be transmitted.                                                     *
 *                                                                        *
 * Input:                                                                 *
 * pep - Pointer to an Ethernet packet.  This structure contains the      *
 *         Ethernet Header and data that is to be transmitted on the link. *
 *                                                                        *
 * Output:                                                                *
 * TX_DONE    Indicates the successful completion of a synchronous Tx.    *
 * TX_WAITING Indicates the MAC has accepted the given frame for later    *
 *            transmission on the link.  In this case the MAC should call*
 *            txnotifyfunc when the HW has finished sending the frame.    *
 * TX_FULLBUF Indicates that the MAC has run out of SW or HW buffer space  *
 *              and is unable to service the transmit request at this time.*
 * PKTTOOBIG  Indicates that the MAC is unable to transmit this frame     *
```

```
 *            because it exceeds the maximum frame size supported by the   *
 *              HW or SW driver.                                            *
 *                                                                          *
 * Notes:                                                                   *
 * The MAC writer is free to define other status codes.  These will be      *
 * treated as an error by the upper layers resulting in the packet being   *
 * discarded.                                                               *
 ****************************************************************************
 */
int
TransmitPkt
(
 struct ep *pep
)
{
 return(TX_DONE);
}


/*
 ****************************************************************************
 *                               ReceivePkt                                 *
 *                                                                          *
 * HW Packet Reception function.                                            *
 *                                                                          *
 * Description:  The mac library will call this routine when it is ready to*
 * copy the frame data from HW into an internal MAC buffer.                 *
 *                                                                          *
 * Input:                                                                   *
 * databuf - pointer to the MAC structure used to hold received packets.    *
 *                                                                          *
 * Output:                                                                  *
 * <none>                                                                   *
 *                                                                          *
 * Notes:                                                                   *
 * This routine is called by the rxnotifyfunc during processing of an Rx    *
 * packet.                                                                  *
 ****************************************************************************
 */
void
ReceivePkt
(
 EMACFRAME *databuf
)
{
 /*
  * Copy frame from HW into databuf.
  */
}


/*
 ****************************************************************************
 *                               ethisr                                     *
 *                                                                          *
```

```
 * Ethernet Interrupt Service Routine.                                    *
 *                                                                        *
 * Description:  The routine is called if the Init function claims an     *
 * interrupt.  Otherwise the types of functions done in this routine would *
 * be done by a polling thread.                                           *
 *                                                                        *
 * Input:                                                                 *
 * <none>                                                                 *
 *                                                                        *
 * Output:                                                                *
 * <none>                                                                 *
 *                                                                        *
 * Notes:                                                                 *
 **************************************************************************
 */
void
ethisr
(
void
)
{
 /*
  * Decide whether this is an Rx, Tx or Status event.
  */
 if(rxnotifyfunc)
 {
  rxnotifyfunc(0);
 }
 if(txnotifyfunc)

 {
  txnotifyfunc(0);
 }
}
```

This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

ZiLOG Worldwide Headquarters
532 Race Street
San Jose, CA 95126
Telephone: 408.558.8500
Fax: 408.558.8300
www.zilog.com