APPLICATION NOTE 712

DS80C400 Ethernet Drivers

Jun 06, 2003

Abstract: The DS80C400 high-speed microcontroller has a built-in Ethernet media-access controller (MAC) with an industry-standard media independent interface (MII). This application note presents design considerations and fully tested example assembly code for an Ethernet interrupt handler, and code for sending and receiving Ethernet packets. Using these routines, you can develop custom applications such as TCP/IP routers.

Introduction

The DS80C400 high-speed microcontroller has a built-in Ethernet media-access controller (MAC) with an industry-standard media independent interface (MII). Please refer to the High-Speed Microcontroller User’s Guide: DS80C400 Supplement and the DS80C400 data sheet for details.

This application note presents design considerations and fully tested example assembly code for an Ethernet interrupt handler, and code for sending and receiving Ethernet packets. Using these routines, you can develop custom application such as TCP/IP routers. Full source code and the header files defining the symbolic constants can be found on the Dallas Semiconductor ftp site at http://files.maximintegrated.com/microcontroller/mxtni/ds80c400/ethdriver/.
The DS80C400 MAC Hardware

Ethernet Buffer Memory
The DS80C400 communicates with the network via a set of special function registers (SFRs) and 8kB of dual port buffer memory. The buffer memory is divided into the receive and send memory and can be addressed in blocks of 256 bytes ("pages"). The receive pages are organized in a circular fashion, managed by the DS80C400 hardware. The send buffer is managed by the user’s application.

The location for the Ethernet buffer is usually address 0FFE000h (default configuration established by ROM loader), assigned to the constant ETH_RECEIVE_BUFFER.

Ethernet Control Status Registers
The primitives ReadCSR and WriteCSR are used to read and write the DS80C400 Ethernet control status registers (CSRs). Note that the example code does not save the processor registers across function calls. When using this code, ensure that you don’t destroy the processor state (this is especially important when using interrupt driven data transfer).
**Read CSR**

ReadCSR reads a control status register.

```asm
;******************************************************************************
;* Function Name: ETH_ReadCSR
;* Description: Read from specified register.
;* Input(s): a -> register address
;* Outputs(s): r3:r2:r1:r0 -> 32 bit register byte value
;******************************************************************************
ETH_ReadCSR:
    push eie
    clr eie.5
    mov csra, a ; Load CSRA SFR with the LSB of the targeted CSR
    anl bcuc, #0f0h ; Clear BCUC command bits
    orl bcuc, #BCU_READ_CSR ; Write read CSR command to BCUC SFR
    push acc
    eth_readcsr_busy: ; Wait until Busy bit in BCUC SFR is reset
        mov a, bcuc ; Move to acc since BCUC is not bit cap.
        jb acc.7, eth_readcsr_busy
    pop acc
    mov r3, csrd ; Read CSRD SFR for MSB of 32 bit data
    mov r2, csrd
    mov r1, csrd
    mov r0, csrd ; LSB
    pop eie
    ret
```

Listing 1. **ReadCSR Reads a Control Status Register**

Note that this code saves, disables, and restores the Ethernet activity interrupt enable (eie.5) to make sure that a write to the CSR is not interrupted by an Ethernet activity interrupt. The definition for the bcuc, csrd and csra SFRs can be found in the include file ds80c400.inc. Constant values such as BCUC_READ_CSR are defined in eth400.inc.

**Write CSR**

The **WriteCSR** function writes a 32 bit value to a control status register.

```asm
;******************************************************************************
;* Function Name: ETH_WriteCSR
;* Description: Write to specified register.
;* Input(s): a -> register address
;* r3:r2:r1:r0 -> 32 bit value
;* Outputs(s): N/A
;******************************************************************************
ETH_WriteCSR:
    push eie
    clr eie.5
    mov csra, a ; Load CSRA SFR with the LSB of the targeted CSR
    anl bcuc, #0f0h ; Clear BCUC command bits
    orl bcuc, #BCU_READ_CSR ; Write read CSR command to BCUC SFR
    push acc
    eth_readcsr_busy: ; Wait until Busy bit in BCUC SFR is reset
        mov a, bcuc ; Move to acc since BCUC is not bit cap.
        jb acc.7, eth_readcsr_busy
    pop acc
    mov r3, csrd ; Read CSRD SFR for MSB of 32 bit data
    mov r2, csrd
    mov r1, csrd
    mov r0, csrd ; LSB
    pop eie
    ret
```
### Initialization

#### MAC Address

In order to use the DS80C400 on the network, a globally unique MAC address needs to be programmed into the device. The MAC address can either be acquired from the DS2502-E48 MAC address 1-Wire® part (Dallas Semiconductor has registered a range of ready-to-go MAC addresses in order to simplify building embedded devices) or from another IEEE® registered source.

**Very important:** Under NO circumstances select a random MAC address or the address of another existing device. MAC addresses are globally unique and network stability depends on well behaved devices!

```assembly
; Function Name: ETH_LoadEthernetAddress
; Description: Load the 48 bit ethernet address into the controller.
; Input(s): dptr0 -> pointer to the Ethernet address (big-endian)
; Outputs(s): N/A

ETH_LoadEthernetAddress:
  movx a, @dptr
  mov r0, a
  inc dptr
  movx a, @dptr
  mov r1, a
  inc dptr
  movx a, @dptr
  mov r2, a
```
inc     dptr
movx    a, @dptr
mov     r3, a
inc     dptr
mov     a, #CSR_MAC_LO
acall   ETH_WriteCSR
movx    a, @dptr
mov     r0, a
inc     dptr
mov     a, @dptr
mov     r1, a
clr     a
mov     r2, a
mov     r3, a
mov     a, #CSR_MAC_HI
acall   ETH_WriteCSR
ret

Listing 3. LoadEthernetAddress Loads the MAC Address into the DS80C400

Note that two CSR writes are required to fully load the 6-byte Ethernet MAC address. Since this code is
only called during initialization, it is not protected against Ethernet activity interrupts.

Initializing the Ethernet MAC further requires configuration of the partition between receive buffer
(incoming packets) and send buffer (outgoing packets). Figure 1 shows this partition between page n-1
and page n.

To simplify code and avoid dropping inbound packets, most applications will benefit from partitioning the
buffer memory in a fashion that reserves most of the pages for inbound packets and only allocates
enough pages for one outbound packet. The reason for this is that Ethernet is a shared medium and—even in switched networks—only a fraction of incoming packets are of interest to an application.
Therefore, we define the constants ETH_TRANSMIT_PAGE to 17h and ETH_SEND_BUFFER to
ETH_RECEIVE_BUFFER + 17h x 256.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETH_TRANSMIT_PAGE</td>
<td>17h</td>
</tr>
<tr>
<td>ETH_SEND_BUFFER</td>
<td>0FFF700h</td>
</tr>
</tbody>
</table>

The following code first disables the transmitter and then initializes the DS80C400 buffer memory to
select the 23:9 receive:send partition. The code then sets the half/full duplex status (this status can be
acquired from the MII, see below) and enables the transmitter.

Enabling the Transceiver

;****************************************************************************
;* Function Name: ETH_EnableTransceiver
;*
;* Description: Enable receiver and transmitter for Ethernet controller.
;*
;* Input(s): N/A
;*
;* Outputs(s): N/A
Listing 4. EnableTransceiver Partitions the Buffer Memory and Enables the Transceiver

Note that this code assumes the duplex status information is stored at location ETH_DUPLEX_STATUS in MOVX memory.

Flushing the Buffer

Next, the Ethernet buffer is flushed to ensure clean startup.
Listing 5. Flush Flushes the Receive Buffer

Sending and Receiving

Sending a Packet
To send a packet, the user's application must first place the packet data in the Ethernet send buffer. If a previous packet was placed at the same address, the application must wait for the transmit to be complete before modifying the buffer memory.

Note that the first four bytes of the send buffer are reserved for the send status word. The first byte that will be transmitted is at location ETH_SEND BUFFER +4.

Listing 6. Transmit Sends a Packet Onto the Network

Receiving a Packet
When a packet is received (usually indicated by an interrupt, see below), the user code needs to unload the packet from the Ethernet buffer memory and then release the buffer memory, unlike the send buffer, which is managed by the user, the receive buffer is managed by the DS80C400.
Unloading the Packet Data

Note that a received packet can span several pages in the receive buffer and it can wrap from the last page in the receive buffer to the first page in the receive buffer. Ensure that your packet copy routine properly handles this case.

```assembly
;******************************************************************************
;* Function Name: ETH_Receive
;* Description: Start unloading the last packet from the Ethernet controller.
;* Input(s): N/A
;* Outputs(s): N/A
;******************************************************************************

ETH_Receive:
    ; Get location of buffer and set dptr0 accordingly
    mov a, bcud
    anl a, #1fh ; we are not interested in the page count
    mov dptr, #ETH_RECEIVE_BUFFER ; receive buffer starting address
    mov b, a ; "multiply"
    clr a
    acall Add_Dptr0_16 ; and add it to receive buffer starting address
    ; dptr0 now points to the receive status word of the packet
    movx a, @dptr
    inc dptr
    mov r2, a ; save LSB of frame length
    movx a, @dptr
    inc dptr
    mov r3, a ; save this

    ; check runt frame, watchdog time-out
    anl a, #(80h or 40h)
    jnz eth_ueh_release
    mov a, r3 ; restore and get frame length
    anl a, #3fh
    mov r3, a ; save HSB of frame length

    ; check CRC error, MII error, collision seen, frame too long
    anl a, #(20h or 08h or 02h or 01h)
    jnz eth_ueh_release
    movx a, @dptr
    inc dptr

    ; check for length error, control frame, unsupported ctrl frame
    mov b, a
```
Listing 7. Receive Receives a Packet from the Network

Releasing the Buffer

After processing an incoming packet, the user code needs to release the buffer memory in the Ethernet receive buffer.

```
anl     bcuc, #0f0h                     ; Clear bcuc command bits
orl     bcuc, #BCU_INV_CURR             ; Write release command to
bcuc SFR
ret
```

Listing 8. Release Releases a Packet from the Receive Buffer

Interrupt Driven Operation

Instead of polling the bit flags in the bcuc SFR, an application should use the Ethernet activity interrupt for better performance. There is one interrupt handler for both receive and transmit complete interrupts. The Ethernet activity interrupt calls location 000073h. Since there are only 8 bytes per interrupt, we suggest installing a long jump to the actual function:

```
or 73h
ljmp ETH_ProcessInterrupt
```

Processing Interrupts

The following code handles both receive and transmit complete interrupts.

```
anl     a, #(80h or 20h or 04h or 02h or 01h)
jnz    eth_ueh_release                    ; bad bad bad
frame!
mov     a, b
anl     a, #40h                                         ; check for
filter match
jz      eth_ueh_release
; XXX Copy the packet into your buffer here.
; XXX r3:r2 contain the length of the packet,
; XXX dptr0 points to the beginning of the data.
; XXX Note that the buffer can wrap!
eth_ueh_release:
    ret
```

Listing 7. Receive Receives a Packet from the Network

Releasing the Buffer

After processing an incoming packet, the user code needs to release the buffer memory in the Ethernet receive buffer.

```
anl     bcuc, #0f0h                     ; Clear bcuc command bits
orl     bcuc, #BCU_INV_CURR             ; Write release command to
bcuc SFR
ret
```

Listing 8. Release Releases a Packet from the Receive Buffer

Interrupt Driven Operation

Instead of polling the bit flags in the bcuc SFR, an application should use the Ethernet activity interrupt for better performance. There is one interrupt handler for both receive and transmit complete interrupts. The Ethernet activity interrupt calls location 000073h. Since there are only 8 bytes per interrupt, we suggest installing a long jump to the actual function:

```
or 73h
ljmp ETH_ProcessInterrupt
```

Processing Interrupts

The following code handles both receive and transmit complete interrupts.

```
anl     a, #(80h or 20h or 04h or 02h or 01h)
jnz    eth_ueh_release                    ; bad bad bad
frame!
mov     a, b
anl     a, #40h                                         ; check for
filter match
jz      eth_ueh_release
; XXX Copy the packet into your buffer here.
; XXX r3:r2 contain the length of the packet,
; XXX dptr0 points to the beginning of the data.
; XXX Note that the buffer can wrap!
eth_ueh_release:
    ret
```
ETH_ProcessInterrupt:
    push    acc
    mov     a, bcuc
    anl     a, #rif       ; Received data?
    jz      eth_pi_no_receive
    ; XXX Call your receive packet handler here.
    ; XXX Ensure it saves and restores all registers!
    ; XXX E.g.: acall ETH_ProcessPacket

eth_pi_no_receive:
    mov     a, bcuc
    anl     a, #tif        ; Transmitted data?
    jz      eth_pi_exit    ; Transmitted data?
    ; XXX If you keep track of a send in progress, here's the place
    ; XXX to clear the flag.
    ; XXX E.g.: clr ds400_xmit
    ; XXX If you keep transmit queue, send next packet from queue
    ; XXX E.g.: acall ETH_SendNextFromQueue

eth_pi_exit:
    pop     acc
    reti

Listing 9. ProcessInterrupt Handles Ethernet Activity Interrupts

Enabling Interrupts

Finally, after enabling the Ethernet interrupt, the DS80C400 is ready to receive and send packets.

Listing 10. EnableInterrupts Enables the Ethernet Activity Interrupt

Media Independent Interface (MII)

The Media Independent Interface (MII) defines I/O lines that allow the DS80C400 to communicate with
the physical layer interface (PHY). Even though many PHYs have a vendor-specific command set, there
are common commands that most PHYs share, defined in the IEEE Std. 802.3. Communications with a
PHY can be used to query a PHY for its auto negotiation and duplex state, and to isolate and "un-
isolate" PHYs (in the case of multiple PHYs) and reconfigure a PHY.

The MII on the DS80C400 is accessed through CSR registers. The following routines read and write an
MII register in a given PHY.

Read MII Register

```assembly
ETH_ReadMII:
push eie
clr eie.5
mov r7, a                   ; Save register number
; Wait until MII is not busy
eth_rmii_busy:
    mov a, #CSR_MII_ADDR
    acall ETH_ReadCSR
    mov a, r0
    jb acc.0, eth_rmii_busy
    clr a
    mov r3, a                   ; Reserved - always clear
    mov r2, a
    mov a, r7                   ; Restore register number
    rr a
    rr a
    mov r7, a
    mov r1, a
    anl a, #07h
    mov a, b
    anl a, #1fh
    rl a
    rl a
    orl a, r1
    mov r1, a
    mov a, r7
    anl a, #0c0h
    mov r0, a
    mov a, #CSR_MII_ADDR
    acall ETH_WriteCSR
```

; Wait until MII is not busy
eth_rmii_busy2:
  mov     a, #CSR_MII_ADDR
  acall   ETH_ReadCSR
  mov     a, r0
  jb              acc.0, eth_rmii_busy2

; Read MII data register
  mov     a, #CSR_MII_DATA
  acall   ETH_ReadCSR
  pop     eie
  ret

Listing 11. ReadMII Reads an MII Register from a Given PHY

Write MII Register

;******************************************************************************
;* Function Name: ETH_WriteMII
;******************************************************************************
ETH_WriteMII:
  push    eie
  clr     eie.5
  push    0               ; Save r1 and r0
  push    1
  mov     r7, a           ; Save register number

; Wait until MII is not busy
eth_wmii_busy:
  mov     a, #CSR_MII_ADDR
  acall   ETH_ReadCSR
  mov     a, r0
  jb              acc.0, eth_wmii_busy

  pop     1
  pop     0
  clr     a
  mov     r3, a           ; Reserved - always clear
  mov     r2, a

; Write MII data register
  mov     a, #CSR_MII_DATA
  acall   ETH_WriteCSR
  mov     a, r7           ; Restore register number
  rr              a
  rr              a       ; And shift to pos 0:2
  mov     r7, a
  ; Save result of shift
  anl     a, #07h          ; Select bits 0:2
  mov     r1, a
  mov     a, b
  ; Load PHY address
  anl     a, #1fh
  rl              a
  rl              a       ; shift to 7:3
Listing 12. WriteMII Writes an MII Register to a Given PHY

MII Example

The following code reads the MII status register of a PHY:

```
mov     b, #0
mov     a, #MII_STATUS
acall   ETH_ReadMII
```

Related Parts

<table>
<thead>
<tr>
<th>DS80C400</th>
<th>Network Microcontroller</th>
<th>Free Samples</th>
</tr>
</thead>
</table>

More Information

For Technical Support: http://www.maximintegrated.com/support
For Samples: http://www.maximintegrated.com/samples
Other Questions and Comments: http://www.maximintegrated.com/contact

Application Note 712: http://www.maximintegrated.com/an712
APPLICATION NOTE 712, AN712, AN 712, APP712, Appnote712, Appnote 712
© 2013 Maxim Integrated Products, Inc.
Additional Legal Notices: http://www.maximintegrated.com/legal