Introduction
The CS8900A is a good candidate for designs with an 8-bit data bus. Because of its small size and built-in filters the chip will take up a minimum of board space while providing a cost effective, high performance Ethernet connection. This application note shows how to use the CS8900A in 8 bit mode, including software information for the programmer and a typical connection diagram for the design engineer.

References
The designer should familiarize himself with the Connecting to non-ISA bus systems chapter in the CS8900A Technical Reference Manual, Low cost, high performance Ethernet Controller for non-ISA systems. This chapter is a reference on how to easily connect the chip to a non-ISA processor. It includes diagrams connecting the CS8900A to a MC68302, a Cirrus Logic CL-PS7111, and a Hitachi SH3. That chapter contains most of the data needed for the design engineer. The data sheet is the source for functional descriptions of the registers, receive operation, transmit operation, timing etc. Only the 8-bit specific issues will be covered in this application note.

Software Drivers
There are many software drivers available for the CS8900A in 16-bit mode, including VxWorks™, Pos®, Linux®, Packet Driver and ATI Nucleus. Source code for the VxWorks, Linux, and Pos are available on the Cirrus Website. The Linux driver, in particular, is a good starting point for writing a custom driver in C. Porting any driver for 8-bit operation is the customer’s responsibility.

I/O Ports
In 8 bit mode the CS8900A is accessed through its eight 16 bit I/O ports.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000h</td>
<td>Read/Write</td>
<td>Receive/Transmit Data (Port 0)</td>
</tr>
<tr>
<td>0002h</td>
<td>Read/Write</td>
<td>Receive/Transmit Data (Port 1)</td>
</tr>
<tr>
<td>0004h</td>
<td>Write-only</td>
<td>TxCMD (Transmit Command)</td>
</tr>
<tr>
<td>0006h</td>
<td>Write-only</td>
<td>TxLength (Transmit Length)</td>
</tr>
<tr>
<td>0008h</td>
<td>Read-only</td>
<td>Interrupt Status Queue</td>
</tr>
<tr>
<td>00Ah</td>
<td>Read/Write</td>
<td>PacketPage Pointer</td>
</tr>
<tr>
<td>00Ch</td>
<td>Read/Write</td>
<td>PacketPage Data (Port 0)</td>
</tr>
<tr>
<td>00Eh</td>
<td>Read/Write</td>
<td>PacketPage Data (Port 1)</td>
</tr>
</tbody>
</table>

Table 1. I/O Mode Mapping

In a non-ISA system these ports are usually memory mapped into standard system memory. Please note that the driver should read or write both bytes when accessing any CS8900A status or event register.

Frame Transmission
Transmission and reception of frames is done through these data ports. The basic steps in transmitting a frame are 1) bid for buffer space on the chip by writing the transmit command to the TxCMD port and the length to TxLength port then checking the BusSt register. 2) if space is available begin writing the data, a byte at a time, to Receive/Transmit data port 0. Refer to the section I/O Space Operation of the data sheet for more details.

For instance, the CS8900A is at its default I/O location of 300h. To transmit a frame that is 81 bytes in length the driver would first write the transmit command 00C0h (Start transmitting after all bytes transferred) to the TxCMD port. This is done by writing the low order byte, C0h, to 304h then writ-
ing the high order byte, 00h, to 305h. Next write
0051h (81 decimal) to the TxLENGTH port. Low
byte, 51h, to 306h then high byte, 00h, to 307h.
Now check to see if transmit space is available.
This is done by checking the BusST register, bit 8.
To check this register you will use the packet page
pointer port and the packet page data port.
Write 0138h to Packet Page Pointer (starts at 30Ah)
then read the Packet Page Data Port 0 (starts at
30Ch). If bit 8 (Rdy4TxNow) is set then you can
start transferring data to Transmit Data Port 0. Do
so in the following manner: write the first byte to
300h, the second byte to 301h, byte 3 to 300h, byte
4 to 301h and so on until the whole frame is written.
The chip will automatically send the frame after the
last byte is written.

Frame Reception
The host is notified of an incoming frame by poll-
ing the Rx Event Register. When the host is aware
of an incoming frame the software should read the
frame data following these steps (assuming I/O
base 300h):
• read the RxStatus word (same data as RxEvent,
  register) from data port 0. Read this high order
  byte 301h first, then low order byte 300h.

Note: it is very important to read the RxStatus
and RxLength high order byte first.
• read the RxLength word (the frame length)
  from data port 0. Read this high order byte
  301h first, then low order byte 300h.
• begin reading the frame data, 300h then 301h,
  300h then 301h until the entire frame has been
  transferred to host memory.

Schematic and Layout Review Service
Prevent problems early in the design phase of your
product. Have your schematic or layout reviewed
free of charge by our experts before you build your
board. Call Applications Engineering at (512) 442-
7555 or send e-mail to ethernet@crystal.cir-
rus.com.

Unsupported functions in 8 bit mode
• Interrupts are not supported. Polled mode must
  be used.
• The DMA engine only uses 16 bit memory ac-
  cesses and does not support 8 bit transfers.
• The packet page pointer has an auto increment
  feature that cannot be used in 8 bit mode.
• An EEPROM is not supported. Most 8 bit de-
  signs should not require one and can eliminate
  the added cost.
Typical Connection Diagram
SAMPLE POLLING ROUTINE

Pseudo Code
#define EventMask = 0xFFC0
#define RegisterMask = 0x003F
#define RxEvent = 0x0004
#define TxEvent = 0x0008
#define BufEvent = 0x000C

Poll-Chip{
    unsigned short Event;
    Event = Poll-Registers();
    While Event <> 0x0000 {
        Switch (RegisterMask & Event) {
            Case RxEvent:
                result = Process-RxEvent(Event);
                break;
            Case TxEvent:
                result = Process-TxEvent(Event);
                break;
            Case BufEvent:
                result = Process-BufEvent(Event);
                break;
        } // End Switch
        Event = Poll-Registers();
    } // End While
} // End Poll-Chip

Poll-Registers{
    unsigned short Event;
    Event = Read-RxEventRegister();
    If (EventMask & Event) {
        return Event;
    }
    Event = Read-TxEventRegister();
    If (EventMask & Event) {
        return Event;
    }
    Event = Read-BufEventRegister();
    If (EventMask & Event) {
        return Event;
    }
    Return 0x0000
} // End Poll-Registers