Application Note

USING THE CRYSTAL® CS8900A IN 8-BIT MODE

By James Ayres

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™, Psos®, Linux®, Packet Driver and ATI Nucleus. Source code for the VxWorks, Linux, and Psos 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.

Offset	Type	Description
0000h	Read/Write	Receive/Transmit Data (Port 0)
0002h	Read/Write	Receive/Transmit Data (Port 1)
0004h	Write-only	TxCMD (Transmit Command)
0006h	Write-only	TxLength (Transmit Length)
0008h	Read-only	Interrupt Status Queue
000Ah	Read/Write	PacketPage Pointer
000Ch	Read/Write	PacketPage Data (Port 0)
000Eh	Read/Write	PacketPage Data (Port 1)

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-

CIRRUS LOGIC®
P.O. Box 17847, Austin, Texas 78760
(512) 445 7222 FAX: (512) 445 7581
http://www.cirrus.com

Copyright © Cirrus Logic, Inc. 2000
(All Rights Reserved)

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 polling 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.cirrus.com.

Unsupported functions in 8 bit mode

- Interrupts are not supported. Polled mode must be used.
- The DMA engine only uses 16 bit memory accesses 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 designs should not require one and can eliminate the added cost.

Contacting Cirrus Logic Support

For a complete listing of Direct Sales, Distributor, and Sales Representative contacts, visit the Cirrus Logic web site at:
http://www.cirrus.com/corporate/contacts/

Crystal is a trademark of Cirrus Logic, Inc.

Linux is a registered trademark of Linus Torvalds

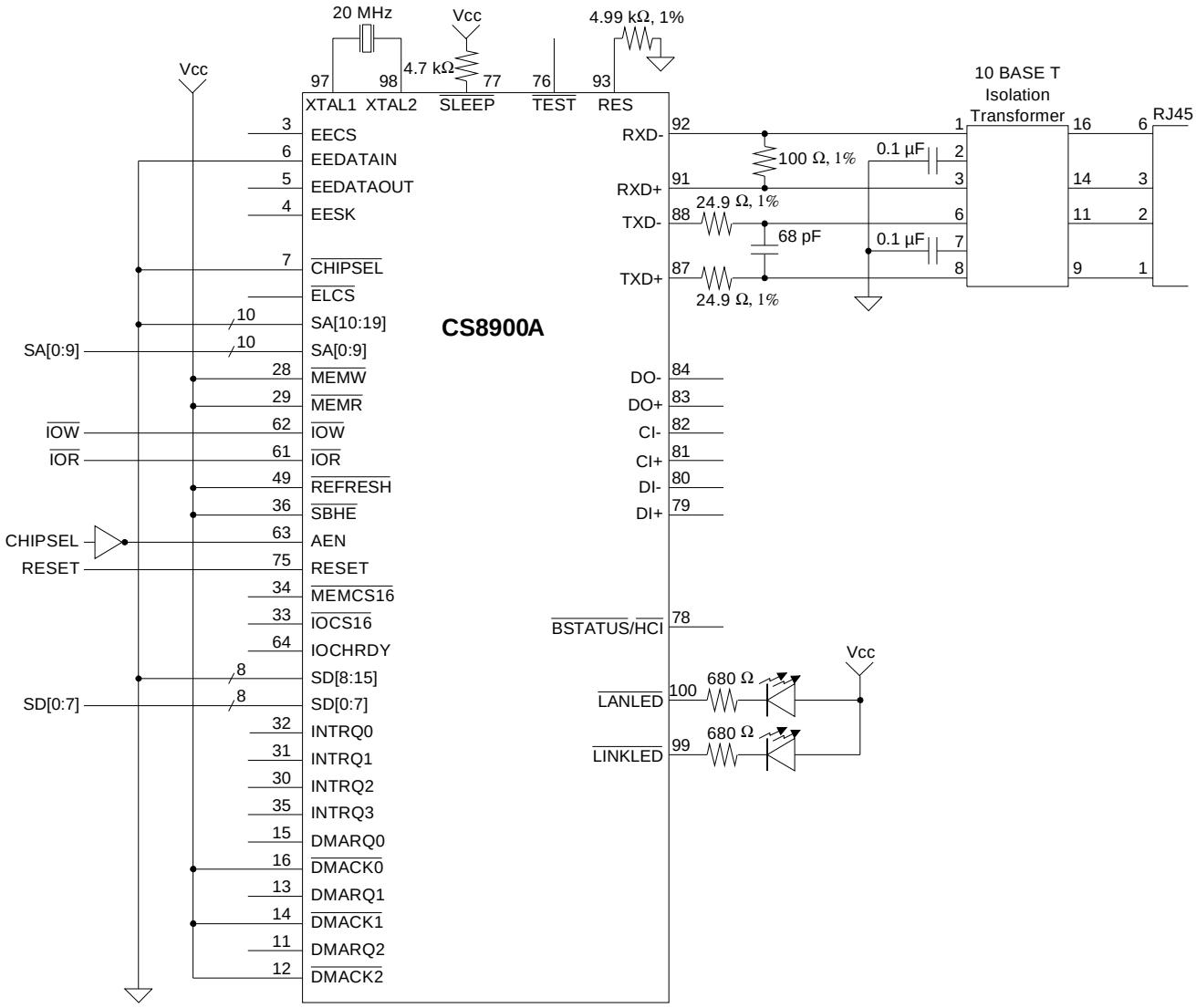
PSOS is register trademark of Integrated System Inc.

VxWorks is a registered trademark of Wind River Systems, Inc.

All other names are trademarks, registered trademarks, or service marks of their respective companies.

Preliminary product information describes products which are in production, but for which full characterization data is not yet available. Advance product information describes products which are in development and subject to development changes. Cirrus Logic, Inc. has made best efforts to ensure that the information contained in this document is accurate and reliable. However, the information is subject to change without notice and is provided "AS IS" without warranty of any kind (express or implied). No responsibility is assumed by Cirrus Logic, Inc. for the use of this information, nor for infringements of patents or other rights of third parties. This document is the property of Cirrus Logic, Inc. and implies no license under patents, copyrights, trademarks, or trade secrets. No part of this publication may be copied, reproduced, stored in a retrieval system, or transmitted, in any form or by any means (electronic, mechanical, photographic, or otherwise) without the prior written consent of Cirrus Logic, Inc. Items from any Cirrus Logic website or disk may be printed for use by the user. However, no part of the printout or electronic files may be copied, reproduced, stored in a retrieval system, or transmitted, in any form or by any means (electronic, mechanical, photographic, or otherwise) without the prior written consent of Cirrus Logic, Inc.Furthermore, no part of this publication may be used as a basis for manufacture or sale of any items without the prior written consent of Cirrus Logic, Inc. The names of products of Cirrus Logic, Inc. or other vendors and suppliers appearing in this document may be trademarks or service marks of their respective owners which may be registered in some jurisdictions. A list of Cirrus Logic, Inc. trademarks and service marks can be found at http://www.cirrus.com.

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

}
```

• Notes •

SMART
Analog
™