CD1284

IEEE 1284-Compatible Parallel Interface Controller with Two High-Speed Asynchronous Serial Ports

Datasheet

Product Features

Parallel Port (Peripheral-side)

High-speed, bidirectional, multi-protocol parallel port:

- Hardware implementation of all modes of the IEEE STD (Standard) 1284 specification (including automatic negotiation)
  - —Centronics®-compatible mode
  - —Reverse Byte mode
  - —Reverse Nibble mode
  - —ECP (extended capabilities port) mode with run-length encoding/decoding
  - —EPP (enhanced parallel port) mode
  - —Up to 2-Mbytes/sec. transfer rate in ECP and EPP modes
- 64-byte parallel FIFO with DMA interface

Two Serial UARTs

- Serial channel asynchronous protocol support to 115.2 kbps (register-set-compatible and functionally identical to CD1400)
  - —Twelve-byte FIFOs for each transmitter and receiver with programmable threshold for receive FIFO interrupt generation
  - —Improved interrupt schemes: Good Data™ interrupts eliminate the need for character status check
  - —User-programmable and automatic flow control for serial channels
  - —Special character recognition and generation.
  - —Special character processing, particularly useful for UNIX® environments, optionally handled automatically by the serial channels.
  - —Six modem control signals per channel (DTR, DSR, RTS, CTS, CD, and RI)

Information in this document is provided in connection with Intel® products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The CD1284 may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature may be obtained by calling 1-800-548-4725 or by visiting Intel's website at http://www.intel.com.

Copyright © Intel Corporation, 2001

*Third-party brands and names are the property of their respective owners.

Datasheet

Contents

1.0 Overview 12

2.0 Conventions 15

2.1 Abbreviations 15
2.2 Acronyms 15

3.0 Pin Information 17

3.1 Pin Diagram 17
3.2 Pin List 18

4.0 Register Summary 24

4.1 Register Summary Tables 24
4.2 Register Usage 27

5.0 Functional Description 31

5.1 Device Architecture 31
5.2 CPU Interface 33
5.2.1 Read Cycles 33
5.2.2 Write Cycles 34
5.2.3 Service-Acknowledge Cycles 34
5.2.4 DMA Cycles 34
5.3 Serial Port Service Requests 35
5.3.1 Interrupts 36
5.3.2 DMAREQ* as Parallel Interrupt Source 36
5.3.3 Serial Service Request Polling 40
5.3.4 Daisy-Chaining Service Requests with CD1400s 41
5.4 Parallel Port Service Requests 43
5.4.1 Hardware-Activated Context Switch, Parallel 48
5.4.2 Software-Activated Context Switch, Parallel 49
5.5 Serial Data Reception and Transmission 49
5.5.1 Receiver Operation 50
5.5.2 Receiver Timer Operations 51
5.5.3 Receive Exceptions 52
5.5.4 Transmitter Operation 54
5.6 Flow Control 55
5.6.1 In-Band Flow Control 55
5.6.2 Receiver In-Band Flow Control 55
5.6.3 Out-of-Band Flow Control 58
5.6.4 Modem Signals and General-Purpose I/O 59
5.7 Receive Special Character Processing 61
5.7.1 UNIX, Character Processing 61
5.7.2 Non-UNIX, Receive Special Character Processing 63
5.8 Transmit Special Character Processing 67
5.8.1 Line Terminating Characters 67
5.8.2 Embedded Transmit Commands 67
5.8.3 Send Special Character Command 68
5.9 Baud Rate Generation 72

5.10 Serial Diagnostic Facilities — Loopback 73
5.11 Parallel Port FIFO and Data Pipeline Overview 73
5.11.1 IEEE STD 1284 Protocols 73
5.11.2 Bus Interface 74
5.11.3 Parallel Port FIFO 74
5.11.4 Receive Direction 75
5.11.5 Receiving Compressed Data 75
5.11.6 Stale Data (Stale, OneChar, and Timeout Status Bits) 76
5.11.7 Transmit Direction 76
5.12 CD1284 Parallel Port Overview 77
5.12.1 Terminology 77
5.12.2 Signal Names 77
5.12.3 State Machine 78
5.12.4 Configuration 78
5.12.5 Interrupts 79
5.12.6 Manual Mode 79
5.12.7 Control Signals 79
5.12.8 Parallel Port Interface to the FIFO 80
5.12.9 1284 Negotiations 80
5.12.10 Data Transfers 81
5.12.11 Compatible Mode Status 81
5.13 1284 Parallel Protocol Support 82
5.13.1 Compatibility Mode 82
5.13.2 Reverse-Nibble and Reverse-Byte Modes 82
5.13.3 ID Request 82
5.13.4 ECP Mode 82
5.13.5 EPP Mode 83
5.14 Protocol Timing 83
5.15 General-Purpose I/O Port 83
5.16 Parallel Port Interface 84
5.17 Hardware Configurations 86
5.17.1 Interfacing to an Intel, Microprocessor-Based System 86
5.17.2 Interfacing to a Motorola, Microprocessor-Based System 86
5.17.3 Interfacing to a National Semiconductor, Microprocessor-Based System86

6.0 Programming 90

6.1 Overview 90
6.2 Initialization 90
6.2.1 Device Reset 90
6.2.2 Global Function Initialization 93
6.2.3 Serial Channel Initialization 93
6.3 Serial Poll Mode Examples 94
6.3.1 Polling Routine Examples 94
6.4 Hardware-Activated Service Examples 97
6.4.1 Serial Receive Service 97
6.4.2 Serial Transmit Service 98
6.4.3 Modem Service 99
6.5 Parallel Channel Service Routines 99
6.5.1 Software-Activated Service Examples (Poll) 100

6.5.2 Hardware-Activated Service Examples 102
6.6 Baud Rate Derivation 102
6.7 Baud Rate Tables 103
6.8 ASCII Code Tables 106
6.8.1 Hexadecimal — Character 106
6.8.2 Decimal — Character 107

7.0 Detailed Register Descriptions 108

7.1 Global Registers 108
7.1.1 Channel Access Register 108
7.1.2 Global Firmware Revision Code Register 108
7.1.3 General-Purpose I/O Direction Register 109
7.1.4 General-Purpose I/O Register 109
7.1.5 Modem Interrupting Channel Register 109
7.1.6 Modem Interrupt Register 110
7.1.7 Parallel Interrupt Register 111
7.1.8 Prescaler Period Register 111
7.1.9 Receive Interrupting Channel Register 112
7.1.10 Receive Interrupt Register 112
7.1.11 Service Request Register 112
7.1.12 Transmit Interrupting Channel Register 113
7.1.13 Transmit Interrupt Register 113
7.2 Virtual Registers 113
7.2.1 Modem Interrupt Status Register 114
7.2.2 Modem Interrupt Vector Register 114
7.2.3 Parallel Interrupt Vector Register 115
7.2.4 Receive Data/Status Registers 115
7.2.5 Receive Interrupt Vector Register 116
7.2.6 Transmit Data Register 117
7.2.7 Transmit Interrupt Vector Register 117
7.2.8 End of Service Request Register 118
7.3 Channel Registers 118
7.3.1 Channel Command Register 118
7.3.2 Channel Control Status Register 122
7.4 Channel Registers — Parallel Pipeline 123
7.4.1 Channel Option Register 1 123
7.4.2 Channel Option Register 2 124
7.4.3 Channel Option Register 3 125
7.4.4 Channel Option Register 4 126
7.4.5 Channel Option Register 5 128
7.4.6 Local Interrupt Vector Register 128
7.4.7 LNext Character Register 129
7.5 Modem Change Option Registers 129
7.5.1 Modem Change Option Register 1 129
7.5.2 Modem Change Option Register 2 130
7.5.3 Modem Signal Value Register 1 130
7.5.4 Modem Signal Value Register 2 131
7.5.5 Receive Baud Rate Period Register 131
7.5.6 Receive Clock Option Register 131
7.5.7 Received Data Count Register 132

7.5.8 Receive Timeout Period Register....133
7.6 *Special Character Registers....133*
7.6.1 Special Character Register 1....133
7.6.2 Special Character Register 2....133
7.6.3 Special Character Register 3....134
7.6.4 Special Character Register 4....134
7.6.5 *Received Character Range Detection....134*
7.6.6 Special Character Range — High....134
7.6.7 Special Character Range — Low....134
7.6.8 Serial Service Request Enable Register....135
7.6.9 Transmit Baud Rate Period Register....135
7.6.10 Transmit Clock Option Register....136
7.7 Channel Registers — Parallel Pipeline....136
7.7.1 Data Error Register....136
7.7.2 DMA Buffer Data Register — High....137
7.7.3 DMA Buffer Data Register — Low....137
7.7.4 Firmware Revision Code Holding Register Status Register....138
7.7.5 Local Interrupt Vector Register....138
7.7.6 Parallel Auxiliary Control Register....139
7.7.7 Parallel Channel Reset Register....140
7.7.8 Parallel FIFO Control Register....140
7.7.9 Parallel FIFO Empty Pointer Register....141
7.7.10 Parallel FIFO Fill Pointer Register....142
7.7.11 Parallel FIFO Holding Register 1....142
7.7.12 Parallel FIFO Holding Register 2....142
7.7.13 Parallel FIFO Quantity Register....143
7.7.14 Parallel FIFO Status Register....143
7.7.15 Parallel FIFO Threshold Register....144
7.7.16 Run Length Count Register....144
7.7.17 Stale Data Timer Count Register....145
7.7.18 Stale Data Timer Period Register....145
7.8 Channel Registers — Parallel Port....146
7.8.1 EPP Address Register....146
7.8.2 Host Timeout Value Register....146
7.8.3 Input Value Register....147
7.8.4 Manual Data Register....148
7.8.5 Negotiation Enable Register....148
7.8.6 Negotiation Status Register....148
7.8.7 Ones Detect Register....149
7.8.8 Output Value Register....150
7.8.9 Parallel Channel Interrupt Enable Register....150
7.8.10 Parallel Channel Interrupt Status Register....150
7.8.11 Parallel Configuration Register....151
7.8.12 Special Command Register....152
7.8.13 Short Pulse Register....153
7.9 Pin Control Registers....154
7.9.1 Signal Status Register....154
7.9.2 Zeros Detect Register....154

8.0 Electrical Specifications 155
8.1 Absolute Maximum Ratings 155
8.2 Recommended Operating Conditions 155
8.3 AC Characteristics 157
8.3.1 Asynchronous Timing 157
8.3.2 Synchronous Timing 163
9.0 Package Dimensions 169
10.0 Ordering Information 170
11.0 Appendix A 171
11.1 Commonly Asked Questions 171
12.0 Appendix B 172
Index 173

Figures

1 Functional Block Diagram 11
2 CD1284 Sample System Block Diagram 14
3 CD1284 Functional Block Diagram 32
4 Internal Address Generation 32
5 Control Signal Generation 38
6 CD1284 Daisy-Chain Connections 42
7 Interrupt Generation Logic 45
8 FIFO Timer Processing 53
9 CD1284 Receive Character Processing 64
10 CD1284 Transmit Character Processing 70
11 FIFO Data Path Functional Diagram — Receive 78
12 FIFO Data Path Functional Diagram — Transmit 80
13 Cable Connection 85
14 External Buffer Control 86
15 Intel, 80x86 Family Interface 87
16 Motorola, 68020 Interface 88
17 National Semiconductor, 32000 Interface 89
18 Flow Diagram of CD1284 Master Initialization Sequence 92
19 Polling Flow Chart 100
20 Reset Timing 158
21 Clock Timing 159
22 Asynchronous Read Cycle Timing 159
23 Asynchronous Write Cycle Timing 160
24 Asynchronous Service Acknowledge Cycle Timing 161
25 Asynchronous DMA Read Cycle Timing 162
26 Asynchronous DMA Read Cycle Timing (Two Back-to-Back DMA Reads) 162
27 Asynchronous DMA Write Cycle Timing 163
28 Asynchronous DMA Write Cycle Timing (Two Back-to-Back DMA Writes) 163
29 Synchronous Read Cycle Timing 165
30 Synchronous Write Cycle Timing 166
31 Synchronous Service Acknowledge Cycle Timing 167
32 Synchronous DMA Write Cycle Timing (Two Back-to-Back 3-Cycle DMA Writes) 168
33 Synchronous DMA Read Cycle Timing (Two Back-to-Back 3-Cycle DMA Reads) 168
34 UART to RS232 and IR Port Interface Motherboard Example Schematic 172

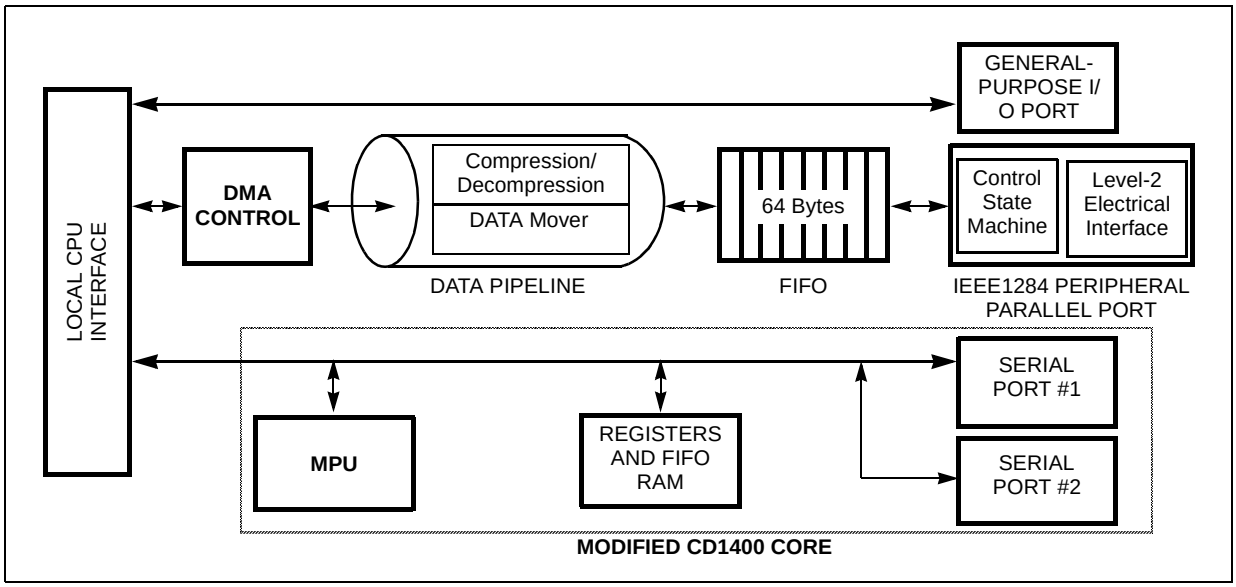
Tables

1 Pin Descriptions 20
2 Global Registers 24
3 Virtual Registers — Serial 24
4 Virtual Registers — Serial and Parallel 24
5 Channel Registers — Serial 25
6 Channel Registers — Parallel Pipeline (Selected by Channel 0 in CAR) 26
7 Channel Registers — Parallel Port (Selected by Channel 0 in CAR) 26
8 Global Registers 27
9 Virtual Registers 27
10 Virtual Registers — Serial and Parallel 28
11 Channel Registers — Serial 28
12 Channel Registers — Parallel Pipeline (Selected by Channel 0 in CAR) 29
13 Channel Registers — Parallel Port (Selected by Channel 0 in CAR) 29
14 Request-Type Bit Assignments 38
15 CCSR[6:5] Encoding 56
16 CCSR[2:1] Encoding 57
17 COR Control Bits 58
18 Out-of-Band Pin Connections 59
19 Modem Control Pin Functions 60
20 Signal Names 77
21 System Clock Settings 83
22 Baud Rate Constants — CLK = 25 MHz 103
23 Baud Rate Constants — CLK = 20.2752 MHz 104
24 Baud Rate Constants — CLK = 20.00 MHz 104
25 Baud Rate Constants — CLK = 18.432 MHz 105
26 Baud Rate Constants — CLK = 16 MHz 105
27 Asynchronous Timing Reference Parameters 157
28 Synchronous Timing Reference Parameters 164

Revision History

Revision	Date	Description
1.0	May 2001	Initial release.

Figure 1. Functional Block Diagram



1.0 Overview

Ideal for printers, scanners, tape drives, set-top boxes, and data acquisition applications, the CD1284 is a multi-function interface controller that implements a high-speed, multi-protocol parallel port and two asynchronous serial ports. The device has both programmed I/O and DMA operation (parallel port only), providing flexibility in local CPU interface design and high-speed data transfers between the device and main memory.

The parallel port implements all modes of the *IEEE STD 1284 Standard Signaling Method for Bidirectional Parallel Peripheral Interface for Personal Computers* specification, including EPP, ECP, Reverse Byte, Reverse Nibble, and Compatible. Data transfer rates (up to 2 Mbytes/sec.) are achievable on the parallel port when the device operates with a 25-MHz clock. The parallel port data and control signals implement the IEEE STD 1284-defined Level-2 interface in drive type (symmetrical), current capability (±14 mA), slew rate (0.4 V/ns), and 0.8 V hysteresis (2.0 V to +7.0 V protection is not implemented).

The two serial ports implement the standard asynchronous protocol. Functionally, the serial ports are identical and register-set-compatible with the CD1400. The table below, shows the differences between the CD1283 and CD1284.

Device	Number of Serial Channels	Number of Parallel Channels
CD1283	0	1
CD1284	2	1

Also included is a general-purpose port that provides eight bits of individual direction programmable I/O that can be used for status and control of external functions.

Theory of Operation

The CD1284 is an efficient high-performance communications controller using an on-chip RISC processor, which off-loads much of the work of sending and receiving data from the CPU. Specifically for data communications applications, the RISC processor employs a high-performance architecture developed by Intel. This internal CPU executes all instructions in one clock cycle, and uses a windowed architecture to ensure zero-overhead context switching for each type of internal interrupt. The processor is transparent to the user and does not require any programming. It manages all serial data movement between the CPU and the two serial channels and provides a flexible interrupt interface for the parallel channel. The parallel channel, being separate and having its own intelligence, implements a very high-speed, peripheral-side parallel data interface.

Each of the serial channels consist of separate 12- byte receive and transmit FIFOs. The parallel channel has a single 64-byte FIFO to support the higher speeds obtainable on the parallel data port. The serial receive FIFOs all have programmable thresholds to minimize interrupt latency requirements. The parallel port FIFO has a programmable DMA threshold in both the receive and transmit directions. The deep FIFOs reduce both the number of interrupt requests made of the CPU and the time required to service them. The time required to service the requests is reduced by four unique vectors that provide internal interrupt conditions. Whether it is receive, transmit, modem signal change, or parallel port, the system spends less time determining the source of the interrupt.The serial receive interrupt service time is further reduced by providing two types of receive vectors: one for ‘good’ data and the other for ‘exception’ data. The CPU does not spend

time determining the status of every character. When the receive vector signifies good data, the CPU removes the data from the FIFO. Checking status is not necessary. Exception data (framing error, overrun, break, etc.) causes an interrupt with a vector that the CPU can immediately identify and manage.

The RISC processor is assisted in the process of sending and receiving serial data by specialized hardware called ‘bit engines’. These logic blocks perform the actual task of sending and receiving the individual bits of a character, thus removing the task of timing the bit duration from the on-chip processor. The processor assembles the bits into characters and tests various parameters (for example, parity, framing, etc.) then places the characters in the FIFO. Since it is managing every character, special character processing is possible such as looking for and responding to flow-control characters (XON/XOFF) and performing UNIX®-style character substitutions and range checking. This reduces interrupt overhead by automatically performing many of the operations that the CPU normally does. Flow-control, for example, can be performed without CPU involvement. Those operations can be completely removed from its responsibility.

The CD1284 can be daisy-chained with other CD1284 or CD1400 devices to implement larger and more complex systems. The Fair Share feature assures equal access for service requests across multiple devices (Fair Share is not implemented on a parallel port interrupt request).

The parallel channel within the CD1284 implements all protocols defined for the peripheral side by the IEEE STD 1284. This specification defines four bidirectional protocols that allow a peripheral device to communicate with a host system (IBM® PC or equivalent) through the parallel printer channel. The modes include Reverse Nibble, Reverse Byte (IBM® PS/2® style), ECP, and EPP (as implemented on the Intel® 80386SL processor). ECP and EPP both operate at data rates as high as 2 Mbytes/sec.

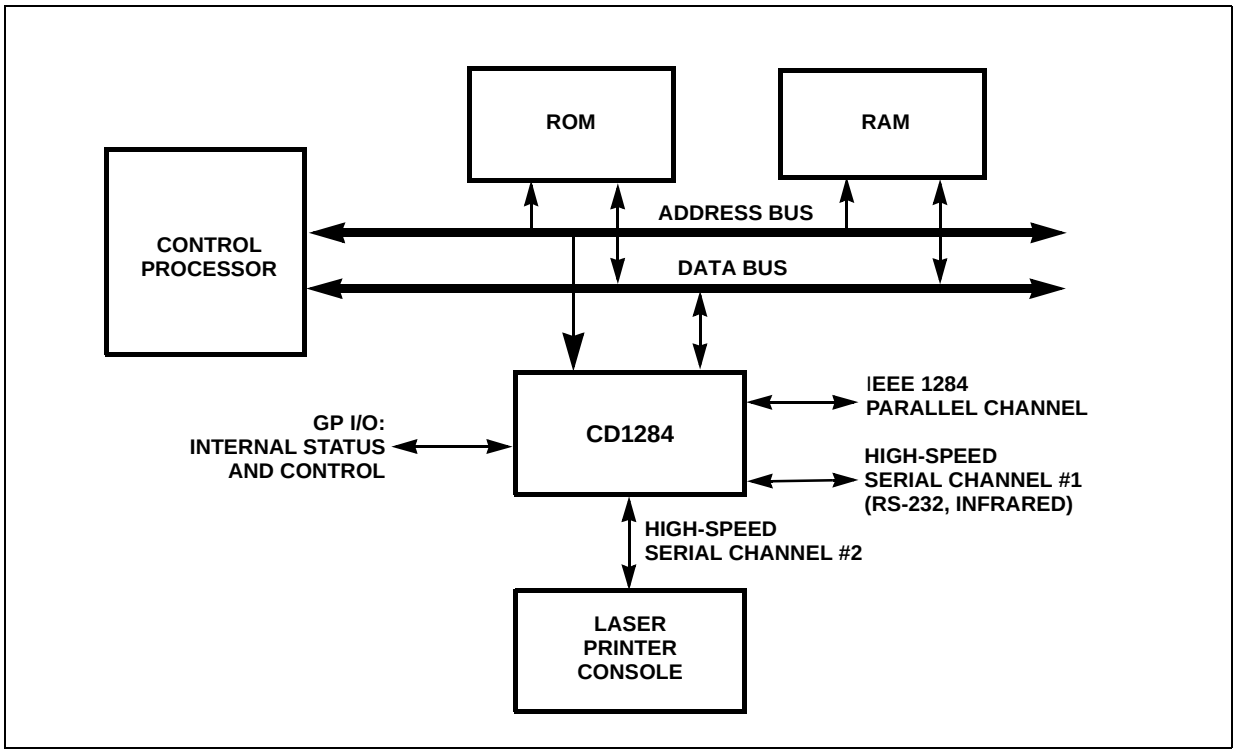
The IEEE 1284 port is implemented as two functional blocks: a data pipeline, which includes the 64-byte FIFO and the DMA interface, and a high-speed state-machine, which controls the parallel port and implements the slave-side IEEE 1284 protocols. The internal RISC processor assists the parallel channel by providing interrupt generation, acknowledgment functions, and a data interface to the Parallel Port registers.

As defined in the IEEE 1284 specification, the CD1284 in ECP mode, provides RLE (run length encoded) data compression in both directions. This data compression is performed automatically (if enabled) and is capable of compressing long strings (up to 128 bytes) of identical data into a two-byte sequence (command/count and data). Since it is common for bit patterns to have large amounts of identical data, the CD1284 greatly reduces data transmission times in printer applications.

EPP mode defines a means of sending address and data over the parallel channel much like a processor address and data interface. This has found widespread use in LAN and SCSI interface adapters that provide these services on laptop computers.

The following figure shows a possible configuration for a CD1284 in a laser-printer application. In this example, the CD1284 provides a parallel and serial data interface to a host system or server. It also provides a serial channel for control communication with the printer console, as well as general-purpose I/O for static control/status.

Figure 2. CD1284 Sample System Block Diagram



2.0 Conventions

2.1 Abbreviations

Symbol	Units of Measure
°C	degree Celsius
Hz	hertz (cycles per second)
Kbyte	kilobyte (1,024 bytes)
kHz	kilohertz
kΩ	kilohm
Mbyte	megabyte (1,048,576 bytes)
MHz	megahertz (1,000 kilohertz)
µF	microfarad
µs	microsecond (1,000 nanoseconds)
mA	milliampere
ms	millisecond (1,000 microseconds)
ns	nanosecond
pV	picovolt

The use of ‘tbd’ indicates values that are ‘to be determined’, ‘n/a’ designates ‘not available’, and ‘N/C’ indicates a pin that is a ‘no connect’.

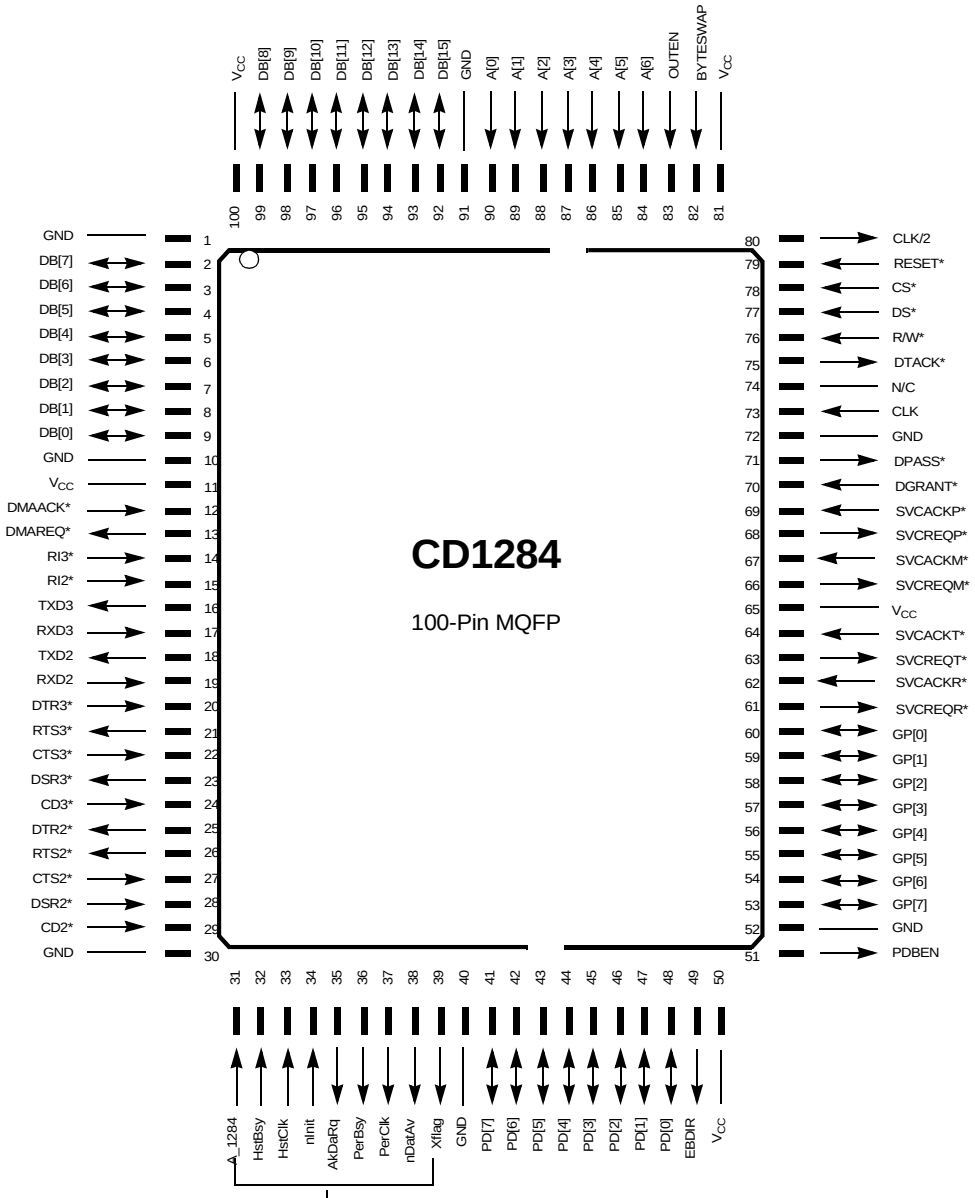
2.2 Acronyms

Acronym	Definition
(Sheet 1 of 2)	
AC	alternating current
BIOS	basic input/output system
CISC	complex instruction set computer
CMOS	complementary metal-oxide semiconductor
DC	direct current
DMA	direct-memory access
DRAM	dynamic random-access memory
ECP	extended capibilities port
EPP	enhanced parallel port
FIFO	first in/first out
GPIO	general-purpose IO

Acronym	Definition
(Sheet 2 of 2)	
HCMOS	high-performance complementary metal-oxide semiconductor
HDLC	high-level data link control
IC	integrated circuit
IDC	instruction and data cache
ISA	industry standard architecture
LSB	least-significant bit
MPU	microprocessing unit
MSB	most-significant bit
PIO	programmed I/O
PPP	point-to-point protocol
MQFP	metric quad flat pack
RAM	random-access memory
RLE	run-length encoded
R/W	read/write
SDLC	synchronous data link control
SRAM	static random-access memory
SWI	software interrupt instruction
TLB	translation look-aside buffer
TTB	translation table base
TTL	transitor-transitor logic
VRAM	video random-access memory
WB	write buffer

3.0 Pin Information

3.1 Pin Diagram



NOTE: (*) Denotes negative-true signal.
'N/C' indicates no connection;
make no connection to these pins.

(See table on next page.)

Pin Names	Compati-bility	Reverse Nibble Mode	Reverse Byte Mode	ECP Mode	EPP Mode
Inputs					
A_1284	SLCTIN*	A_1284	A_1284	A_1284	nAStrb
HstBsy	AUTOFD*	HstBsy	HstBsy	HstAck	nDStrb
HstClk	STROBE*	HstClk	HstClk	HstClk	nWrite
nInit	INIT*	nInit	nInit	nRevReq	nInit
Outputs					
AkDaRq	PError	AkDaRq	AkDaRq	nAkRev	USER1
PerBsy	BUSY	PerBsy	PerBsy	PerAck	nWait
PerClk	ACK*	PerClk	PerClk	PerClk	Intr
nDatAv	FAULT*	nDatAv	nDatAv	nPerReq	USER2
XFlag	SELECT	XFlag	XFlag	XFlag	USER3

3.2 Pin List

The following conventions are used in the pin-description tables:

- (*) after a name indicates that the signal is active-low
- ‘I’ indicates the pin is input-only
- ‘O’ indicates the pin is output-only
- ‘I/O’ indicates the pin is bidirectional
- ‘OD’ indicates an open-drain output that the user must tie to V_{CC} through a pull-up resistor (usually about 1 kΩ)
- ‘AR’ indicates active release (pin drives to ‘1’ and releases to ‘OD’)
- ‘TS’ indicates tristate
- a ‘–’ indicates ascending pin numbers
- a ‘:’ indicates descending pin numbers

Pin Name	Type	Number of Pins	Pin Number	Reset State
(Sheet 1 of 3)				
5V	–	5	11, 50, 65, 81, 100	
GND	–	7	1,10, 30, 40, 52, 72, 91	
RESET*	I	1	79	

Pin Name	Type	Number of Pins	Pin Number	Reset State
(Sheet 2 of 3)				
OUTEN	I	1	83	
CLK	I	1	73	
CLK/2	O	1	80	n/a
DB[15:0]	I/O	16	92–99, 2–9	TS
A[6:0]	I	7	84–90	
R/W*	I	1	76	
CS*	I	1	78	
DS*	I	1	77	
BYTESWAP	I	1	82	
DTACK*	AR	1	75	
DMAREQ*	O	1	13	High
DMAACK*	I	1	12	
SVCREQR*	OD	1	61	
SVCACKR*	I	1	62	
SVCREQT*	OD	1	63	
SVCACKT*	I	1	64	
SVCREQP*	OD	1	68	
SVCACKP*	I	1	69	High
SVCREQM*	OD	1	66	
SVCACKM*	I	1	67	
DGRANT*	I	1	70	
DPASS*	O	1	71	High
PD[7:0]	I/O	8	41–48	TS
GP[7:0]	I/O	8	53–60	TS
A_1284	I	1	31	
HstBsy	I	1	32	
HstClk	I	1	33	
nInit	I	1	34	
AkDaRq	O	1	35	
PerBsy	O	1	36	Low
PerClk	O	1	37	High
nDatAv	O	1	38	High
Xflag	O	1	39	Low
EBDIR	O	1	49	High
PDBEN	O	1	51	Low
TXD3	O	1	16	High

Pin Name	Type	Number of Pins	Pin Number	Reset State
(Sheet 3 of 3)				
RXD3	I	1	17	
TXD2	O	1	18	High
RXD2	I	1	19	
RTS2*	O	1	26	High
RTS3*	O	1	21	High
DTR2*	O	1	25	High
DTR3*	O	1	20	High
CTS2*	I	1	27	
CTS3*	I	1	22	
DSR2*	I	1	28	
DSR3*	I	1	23	
CD2*	I	1	29	
CD3*	I	1	24	
RI2*	I	1	15	
RI3*	I	1	14	
N/C	–	1	74	

Table 1. Pin Descriptions (Sheet 1 of 4)

Symbol	Pin No.	Type	Description
RESET*	79	I	**ACTIVE-LOW RESET:** This input initializes the device to the default condition. All internal registers are set to their reset condition and all transfer operations are set to the default state.
OUTEN	83	I	**OUTPUT ENABLE:** This pin must be ‘1’ to enable output pin functions. When OUTEN is ‘0’, it forces all output pins to remain in a tristate condition. Typically, OUTEN is used only for test purposes. User designs must tie this pin to V_{CC} through a pull-up resistor.
CLK	73	I	**SYSTEM CLOCK:** This input has a 25-MHz maximum; 16 MHz is the recommended minimum for satisfactory device performance.
CLK/2	80	O	**SYSTEM CLOCK DIVIDED BY TWO OUTPUT:** This signal is equivalent to the internal operating clock of the device.
DB[15:0]	92–99, 2–9	I/O	**BIDIRECTIONAL DATA BUS:** Only DMA transfers and writes to the DMA Buffer register are true 16-bit operations. During all register writes other than to the DMA Buffer register, bits [7:0] are written to the addressed register. Register reads duplicate the register contents on both the lower byte [7:0] and upper byte [15:8].
A[6:0]	84–90	I	**ADDRESS BUS:** Together with CS* or one of the SVCACK* inputs and DS*, this input selects an On-Chip register for a read or write operation or an acknowledgment to an service request.
R/W*	76	I	**READ/WRITE*:** This input must be ‘1’ for a register read operation, and must be ‘0’ for a register write. R/W* is ignored for DMA operations.
CS*	78	I	**ACTIVE-LOW CHIP SELECT:** When active, the input CS* combines with DS*, initiates an I/O cycle with the CD1284. CS* must be ‘1’ during DMA read/write operations.

Table 1. Pin Descriptions (Sheet 2 of 4)

Symbol	Pin No.	Type	Description
DS*	77	I	**ACTIVE-LOW DATA STROBE:** During an active I/O cycle, the input DS* strobes data into On-Chip registers on write cycles or enables data onto the data bus during read cycles. DS* is ignored during DMA operations.
BYTESWAP	82	I	**BYTESWAP:** This input determines the byte order for 2-byte DMA transfers and for writes to the DMA Buffer register. When BYTESWAP is '1', then Data Bus bits [15:8] are driven with the byte transferred first on the parallel port bus. Data Bus bits [7:0] are driven with the byte transferred second on the parallel port bus. When BYTESWAP is '0', the data order is reversed, bits [7:0] are driven with the byte transferred first and bits [15:8] are driven with the byte transferred second.
DTACK*	75	AR	**ACTIVE-LOW DATA TRANSFER ACKNOWLEDGE:** This output indicates: 1) when the device completes the requested I/O operation, and, 2) when the current cycle can finish. This signal can implement wait-state insertion for the local CPU. DTACK* does not activate on DMA cycles.It is an active-release output, driving to a logic '1' then releasing to OD. DTACK* must be ties to external V_{cc} through a pull-up resistor.
DMAREQ*	13	O	**ACTIVE-LOW DMA REQUEST:** When the internal control bit DMAen is set, the output DMAREQ* is asserted if internal FIFO conditions warrant a DMA transfer. DMAREQ* is deasserted on the falling edge of DMAACK* when DMA transfers cannot continue past the current transfer.
DMAACK*	12	I	**ACTIVE-LOW DMA ACKNOWLEDGE:** This input is never asserted unless in response to a DMAREQ* from the chip. DMAACK* is the only bus handshake signal recognized during a DMA transfer. (CS* must be high whenever DMAACK* is asserted). The direction of DMA transfer is determined by internal control bit DMAdir.
SVCREQR*	61	OD	**ACTIVE-LOW SERVICE REQUEST RECEIVE:** This is an open-drain output and must be tied to external V_{CC} through a pull-up resistor. When active, the device serial-receive FIFO has either reached the programmed threshold or an exception condition exists that requires CPU attention.
SVCACKR*	62	I	**ACTIVE-LOW SERVICE ACKNOWLEDGE RECEIVE:** This input is driven low during service acknowledge cycles to begin servicing a receive-service request. It must not be driven active except in response to a receive-service request presented by the device.
SVCREQT*	63	OD	**ACTIVE-LOW SERVICE REQUEST TRANSMIT:** This is an open-drain output and must be tied to external V_{CC} through a pull-up resistor. When active, the device serial transmit FIFO or serial transmitter is empty and requires CPU attention.
SVCACKT*	64	I	**ACTIVE-LOW SERVICE ACKNOWLEDGE TRANSMIT INPUT:** This input is driven low during service acknowledge cycles to begin servicing a transmit-service request. It must not be driven active except in response to a transmit-service request presented by the device.
SVCREQP*	68	OD	**ACTIVE-LOW SERVICE REQUEST PARALLEL:** This is an open-drain output and must be tied to external V_{CC} through a pull-up resistor. SVCREQP* is not activated by FIFO threshold or FIFO full/empty conditions.
SVCACKP*	69	I	**ACTIVE-LOW SERVICE ACKNOWLEDGE PARALLEL:** This input cannot be driven active except in response to a parallel service request presented by the device.
SVCREQM*	66	OD	**ACTIVE-LOW SERVICE REQUEST STATUS** (Modem)**:** This is an open-drain output that must be tied to external V_{CC} through a pull-up resistor. When active, a programmed modem signal change occurs and requires CPU attention.
SVCACKM*	67	I	**ACTIVE-LOW SERVICE ACKNOWLEDGE STATUS** (Modem)**:** This input is driven low during service acknowledge cycles to begin servicing a modem-service request. It must not be driven active except in response to a modem-service request presented by the device.
DGRANT*	70	I	**ACTIVE-LOW DAISY GRANT:** This input is driven active during service acknowledge cycles to enable the daisy-chain function. This input, when qualified with DS* and a valid service acknowledge (SVCACKR*, SVCACKT*, SVCACKM*, or SVCACKP*), activates the CD1284 service-acknowledge cycle.

Table 1. Pin Descriptions (Sheet 3 of 4)

Symbol	Pin No.	Type	Description
DPASS*	71	O	**ACTIVE-LOW DAISY PASS:** This output is driven active during service acknowledge cycles to enable the next device in the daisy chain. It is driven active when no valid service request exists for the type of service acknowledge input active. In multiple CD1284 designs, this signal is normally connected to the DGRANT* input of the next device in the chain.
PD[7:0]	41–48	I/O	**PARALLEL PORT DATA LINES [7:0]:** Bidirectional (depending on the protocol being used), these signals are used to transfer data through the interface between the master and slave.
GP[7:0]	53–60	I/O	**GENERAL PURPOSE I/O [7:0]:** General-purpose input/output port data lines. These signals are individually direction programmable and act as inputs or outputs. The corresponding bit in the GPDIR register controls the direction of each signal. The GPIO register provides the control/status of the actual signals.
A_1284	31	I	**1284 ACTIVE INPUT:** (SLCTIN* in Compatibility mode). Active-high.
nInit	34	I	**INIT SIGNAL:** (INIT* in Compatibility mode). Active-low.
HstBsy	32	I	**HOST BUSY:** (AUTOFD* in Compatibility mode). Active-high.
HstClk	33	I	**HOST CLOCK:** (STROBE* in Compatibility mode). Active-low.
The above four parallel handshake signals are driven by the master in an IEEE STD 1284 interface, and as such are inputs to the CD1284. Their functions depend on the transfer protocol selected. Refer to the IEEE STD 1284 document for protocol functions.			
PerClk	37	O	**PERIPHERAL CLOCK:** (ACK* in Compatibility mode). Active-low.
PerBsy	36	O	**PERIPHERAL BUSY:** (BUSY in Compatibility mode). Active-high.
AkDaRq	35	O	**ACKNOWLEDGE DATA REQUEST:** (PError in Compatibility mode).
Xflag	39	O	**EXTENSIBILITY FLAG:** (SELECT in Compatibility mode).
nDatAv	38	O	**DATA AVAILABLE**: (FAULT* in Compatibility mode). Active-low.
The above five parallel handshake signals are driven by the slave in an IEEE STD 1284 interface and are outputs from the CD1284. Their functions depend on the transfer protocol selected. Refer to the IEEE STD 1284 document for protocol functions.			
EBDIR	49	O	**EXTERNAL BUFFER DIRECTION:** This signal is controlled by the internal parallel-port-control state machine and is used to control the direction of an external buffer connected to the parallel-port data bus. An external buffer could be desirable in applications that require higher drive capacity than those provided by the CD1284. EBDIR can be used in conjunction with PDBEN to control this buffer. EBDIR is a logic ‘0’ when the parallel data bus is in an output mode and a logic ‘1’ when in an input mode. It can be connected directly to the direction control input of a 74245-type device.
PDBEN	51	O	**PARALLEL DATA BUS ENABLE:** This signal can be used to control a buffer on the parallel port data lines in applications requiring more signal drive capability than that provided by the CD1284. The signal is controlled by the internal parallel port control state-machine. When low, the parallel port data bus is off (not driving); when high, the port is in an output mode and is actively driving. The signal toggles between on and off states during output modes and is active (high) only when the data bus pins are in the active driving state. This signal can be logically connected to the enable control of 74245 (or equivalent) bidirectional buffers.
TXD[3,2]	16, 18	O	**TRANSMIT DATA:** TXD[3,2] are outputs of serial channel numbers two and three.
RXD[3,2]	17, 19	I	**RECEIVE DATA:** RXD[3, 2] are outputs of serial channel numbers two and three.
RTS[3,2]*	21, 26	O	**REQUEST TO SEND:** These are active-low outputs of serial channel numbers two and three.
DTR[3,2]*	20, 25	O	**DATA TERMINAL READY:** These are active-low outputs of serial channels two and three.
CTS[3,2]*	22, 27	I	**CLEAR TO SEND:** These are active-low inputs for serial channels two and three.

Table 1. Pin Descriptions (Sheet 4 of 4)

Symbol	Pin No.	Type	Description
DSR[3,2]*	23, 28	I	**DATA SET READY:** These are active-low inputs for serial channels two and three.
CD[3,2]	24, 29	I	**CARRIER DETECT:** These are active-low inputs for serial chanels two and three.
RI[3,2]	14, 15	I	**RING INDICATOR:** These are active-low inputs for chanels two and three.

4.0 Register Summary

4.1 Register Summary Tables

Table 2. Global Registers

Name	Hex	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
CAR	68	Poll	Poll	Poll	Poll	Poll	0	C1	C0	108
GFRCR	4F	Firmware Revision Code								108
GPDIR	71	Dir 7	Dir 6	Dir 5	Dir 4	Dir 3	Dir 2	Dir 1	Dir 0	109
GPIO	70	Data 7	Data 6	Data 5	Data 4	Data 3	Data 2	Data 1	Data 0	109
MICR	45	X	X	X	X	C1	C0	X	X	109
MIR	69	MdIreq	Mdbusy	Mdunfair	0	1	0	ch[1]	ch[0]	110
PIR	61	PPIreq	PPort	Pipeline	0	0	0	0	0	111
PPR	7E	8-Bit Binary Value								111
RICR	44	X	X	X	X	C1	C0	X	X	112
RIR	6B	RxIreq	Rxbusy	Rxunfair	1	1	0	ch[1]	ch[0]	112
SVRR	67	DMAREQ	ExtM	ExtT	ExtR	SRP	SRM	SRT	SRR	112
TICR	45	X	X	X	X	C1	C0	X	X	113
TIR	6A	TxIreq	Txbusy	Txunfair	1	0	0	ch[1]	ch[0]	113

Table 3. Virtual Registers — Serial

Name	Hex	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
MISR	4C	DSRch	CTSch	RIch	CDch	0	0	0	0	114
MIVR	41	X	X	X	X	X	IT2	IT1	IT0	114
PIVR	40	X	X	X	X	X	IT2	IT1	IT0	115
RDSR (data)	62	Received Character								115
RDSR (status)	62	Timeout	SC Det2	SC Det1	SC Det0	Break	PE	FE	OE	115
RIVR	43	X	X	X	X	X	IT2	IT1	IT0	116
TDR	63	Transmit Character								117
TIVR	42	X	X	X	X	X	IT2	IT1	IT0	117

Table 4. Virtual Registers — Serial and Parallel

Name	Hex	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
EOSRR	60	X	X	X	X	X	X	X	X	118

Table 5. Channel Registers — Serial

Name	Hex	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
CCR[1]	05	Res Chan	COR Chg	Send SC	Chan Ctl	D3	D2	D1	D0	118
CCSR	0B	RxEN	RxFloff	RxFlon	0	TxEN	TxFloff	TxFlon	0	122
COR1	08	Parity	ParM1	ParM0	Ignore	Stop1	Stop0	ChL1	ChL0	123
COR2	09	IXM	TxIBE	ETC	LLM	RLM	RtsAO	CtsAE	DsrAE	124
COR3	0A	SCDRNG	SCD34	FCT	SCD12	RxTh3	RxTh2	RxTh1	RxTh0	125
COR4	1E	IGNCR	ICRNL	INLCR	IGNBRK	–BRKINT	PEH[2]	PEH[1]	PEH[0]	126
COR5	1F	ISTRIP	LNE	CMOE	0	0	EBD	ONLCR	OCRNL	128
LIVR	18	X	X	X	X	X	IT2	IT1	IT0	128
LNC	24	LNext Character								129
MCOR1	15	DSRzd	CTSzd	RIzd	CDzd	DTRth3	DTRth2	DTRth1	DTRth0	129
MCOR2	16	DSRod	CTSod	RIod	CDod	0	0	0	0	130
MSVR1	6C	DSR	CTS	RI	CD	0	0	0	RTS	130
MSVR2	6D	DSR	CTS	RI	CD	0	0	DTR	0	131
RBPR	78	Binary Divisor Value								131
RCOR	7C	0	0	0	0	0	ClkSel2	ClkSel1	ClkSel0	131
RDCR	0E	0	0	0	0	CT3	CT2	CT1	CT0	132
RTPR	21	Binary Count Value								133
SCHR1	1A	Special Character 1								133
SCHR2	1B	Special Character 2								133
SCHR3	1C	Special Character 3								134
SCHR4	1D	Special Character 4								134
SCRH	23	Character Range — high								134
SCRL	22	Character Range — low								134
SRER	06	MdmChg	0	0	RxData	0	TxRdy	TxEmpty	NNDT	135
TBPR	72	Binary Divisor Value								135
TCOR	76	0	0	0	0	0	ClkSel2	ClkSel1	ClkSel0	136

NOTE:
1. The CCR contents and offsets apply to any of the channels; the channel being access at any given time is controlled by the CAR. See Section 7.3.1.1 through Section 7.3.1.4 for channel-specific bit settings.

Table 6. Channel Registers — Parallel Pipeline (Selected by Channel 0 in CAR)

Name	Hex	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
DER	33	DMAwrerr	DMArderr	Bufwrerr	Bufrderr	HR1wrerr	HR1rderr	HR2wrerr	HR2rderr	136
DMABUF (H)	30	15	14	13	12	11	10	9	8	137
DMABUF (L)	30	7	6	5	4	3	2	1	0	137
HRSR	34	HR1full	HR1tag	HR2full	HR2tag	DMAfull	DMAmpty	DMAact	Ctnot0	138
LIVR	18	User-Defined Bits					IT2	IT1	IT0	138
PACR	3F	ShrtTen	ShrtStal	StaleOff	FIFOlock	ClearTO	0	AsyncDMA	Unfair	139
PCRR	6C	0	0	0	0	0	0	0	PChReset	140
PFCR	31	FIFOres	DMAen	DMAdir	IntEn	RLEen	setTAG	ErrEn	DMAbufWe	140
PFEP	39	0	0	6-Bit Binary FIFO Pointer Value						141
PFFP	38	0	0	6-Bit Binary FIFO Pointer Value						142
PFHR1	35	8-Bit Character Data								142
PFHR2	36	8-Bit Character Data								142
PFQR	3A	Data or Space Available in FIFO — Max 0x'40								143
PFSR	32	FFfull	FFempty	Timeout	HRtag	HRdata	Stale	OneChar	DataErr	143
PFTR	3B	0	DMA Transfer Threshold							144
RLCR	37	0	7-Bit Unsigned Binary Count							144
SDTCR	3D	8-Bit Stale Data Timer Count								145
SDTPR	3C	8-Bit Stale Data Timeout Value								145

Table 7. Channel Registers — Parallel Port (Selected by Channel 0 in CAR) (Sheet 1 of 2)

Name	Hex	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
EAR	25	8-Bit Binary Value								146
HTVR	24	HTVR[7]	HTVR[6]	HTVR[5]	HTVR[4]	HTVR[3]	HTVR[2]	HTVR[1]	HTVR[0]	146
IVR	2E	0	0	0	0	A1284	nInit	HstBsy	HstClk	147
MDR	21	8-Bit Binary Data								148
NER	28	0	RID	0	EPP	RLE	ECP	RVB	RVN	148
NSR	29	NegOK	NegFl	HostTO	ImedTerm	4-Bit Negotiation Result Code				148
ODR	2D	0	0	0	0	A1284	nInit	HstBsy	HstClk	149
OVR	2B	PerBsy	PerClk	AkDaRq	xFlag	nDatAv	0	0	0	150
PCIER	22	0	TimEn	NegCh	SigCh	EPPAW	DirCh	IDReq	nINIT	150
PCISR	23	0	TimeOvr	NegCh	SigCh	EPPAW	DirCh	IDReq	nINIT	150
PCR	20	ManMd	E1284	ETxfr	Ig_SEL	HTmrTst[1:0]		MMDir	ManOE	151
SCR	2A	0	0	0	TstMux	ClrPs	SetPs	EPIrq	RevRq	152

Table 7. Channel Registers — Parallel Port (Selected by Channel 0 in CAR) (Sheet 2 of 2)

Name	Hex	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
SPR	26	8-Bit Binary Value								153
SSR	2F	0	0	0	0	A1284	nInit	HstBsy	HstClk	154
ZDR	2C	0	0	0	0	A1284	nInit	HstBsy	HstClk	154

4.2 Register Usage

Table 8 through Table 13 present register tunctionality.

Table 8. Global Registers

Name	Reset	Parallel Init	Parallel Tx	Parallel Rx	Serial Init	Serial Tx	Serial Rx	GPIO Pin Control
CAR	√	√	√	√	√	√	√	
GFRCR	√							
GPDIR								√
GPIO								√
MICR					√	√	√	
MIR						√	√	
PIR			√	√				
PPR					√			
RICR					√	√	√	
RIR								
SVRR								√
TICR					√	√	√	
TIR						√		

Table 9. Virtual Registers

Name	Reset	Parallel Init	Parallel Tx	Parallel Rx	Serial Init	Serial Tx	Serial Rx
MISR						√	√
MIVR						√	√
PIVR			√	√			
RDSR (data)							√
RDSR (status)							√
RIVR							√
TDR						√	
TIVR						√	

Table 10. Virtual Registers — Serial and Parallel

Name	Reset	Parallel Init	Parallel Tx	Parallel Rx	Serial Init	Serial Tx	Serial Rx
EOSRR			√	√		√	√

Table 11. Channel Registers — Serial

Name	Reset	Parallel Init	Parallel Tx	Parallel Rx	Serial Init	Serial Tx	Serial Rx
CCR	√	√			√		
CCSR		√			√		
COR1					√		
COR2					√		
COR3					√		
COR4					√		
COR5					√		
LIVR		√			√		
LNC					√		
MCOR1					√		
MCOR2					√		
MSVR1						√	√
MSVR2						√	√
RBPR					√		
RCOR					√		
RDCR							√
RTPR					√		
SCHR1					√		
SCHR2					÷		
SCHR3					÷		
SCHR4					÷		
SCRH					÷		
SCRL					÷		
SRER					÷		
TBPR					÷		
TCOR					÷		

Table 12. Channel Registers — Parallel Pipeline (Selected by Channel 0 in CAR)

Name	Reset	Parallel Init	Parallel Tx	Parallel Rx	Parallel Error	Parallel Status	Serial Init
DER					√		
DMABUF(H)			√	√			
DMABUF(L)			√	√			
HRSR						√	
HTVR		√					
LIVR		√					
PACR		√					
PCRR	√						
PFCR		√					
PFEP							
PFFP							
PFHR1				√			
PFHR2			√	√			
PFQR			√	√			
PFSR			√	√			
PFTR		√					
RLCR				√			
SDTCR		√					√
SDTPR		√					

Table 13. Channel Registers — Parallel Port (Selected by Channel 0 in CAR) (Sheet 1 of 2)

Name	Reset	Parallel Init	Parallel Tx	Parallel Rx	Parallel Error	Parallel Status
EAR				√ (EPP)[1]		
IVR			√	√		
MDR			√	√		
NER		√				
NSR			√	√		
ODR		√				
OVR			√ (Manual)	√ (Manual)		
PCIER		√				
PCISR			√	√		
PCR		√				

Table 13. Channel Registers — Parallel Port (Selected by Channel 0 in CAR) (Sheet 2 of 2)

Name	Reset	Parallel Init	Parallel Tx	Parallel Rx	Parallel Error	Parallel Status
SCR			√ (RevRequest)			
SPR		√				
SSR			√	√		
ZDR		√				

NOTE:
1. Items in parentheses ( ) denote Operational mode.

5.0 Functional Description

5.1 Device Architecture

The CD1284 can be described as a small computer system designed for the purpose of sending and receiving both serial and parallel data. It comprises a RISC processor (Multi-Channel Processing Unit or MPU), RAM, ROM, local CPU bus interface logic, two serial data channels, and one IEEE 1284-compliant parallel port with a specialized data pipeline designed for high-speed transfers.

Architecturally, the CD1284 is two devices merged into a single unit. One part is a modified, two-channel version of the Intel CD1400. The other part is a specialized parallel interface port supported by its own deep FIFO and DMA interface logic. The interrupt structure of the CD1400 has been enhanced to include the interrupt requirements of the parallel port. This section describes the modified CD1400 core and overall device architecture. Further sections provide details specific to the parallel channel. Chapter 7.0 provides detailed bit descriptions and encoding for the registers discussed in this chapter.

The MPU is a true RISC processor. In addition to having compact and efficient instructions, the MPU has a ‘windowed’ architecture that allows it to handle one channel and its registers at a time. Before beginning operations on a given channel, it loads an internal Index register that forces all accesses to the appropriate set of registers. The Index register becomes part of the internal address and allows direct addressing of the register bank and all hardware resources of the selected channel. No address computation is required to select the proper channel.

This same windowed scheme is carried through to the CPU interface as well (Figure 4). For all channel-specific accesses, the CPU first loads the CAR (Channel Access register) with a pointer to the channel to be accessed. Thereafter, all read and write operations occur with the proper channel. The software defines the register address once and this is valid for all channels because the CAR is part of the internal addressing.

Figure 3. CD1284 Functional Block Diagram

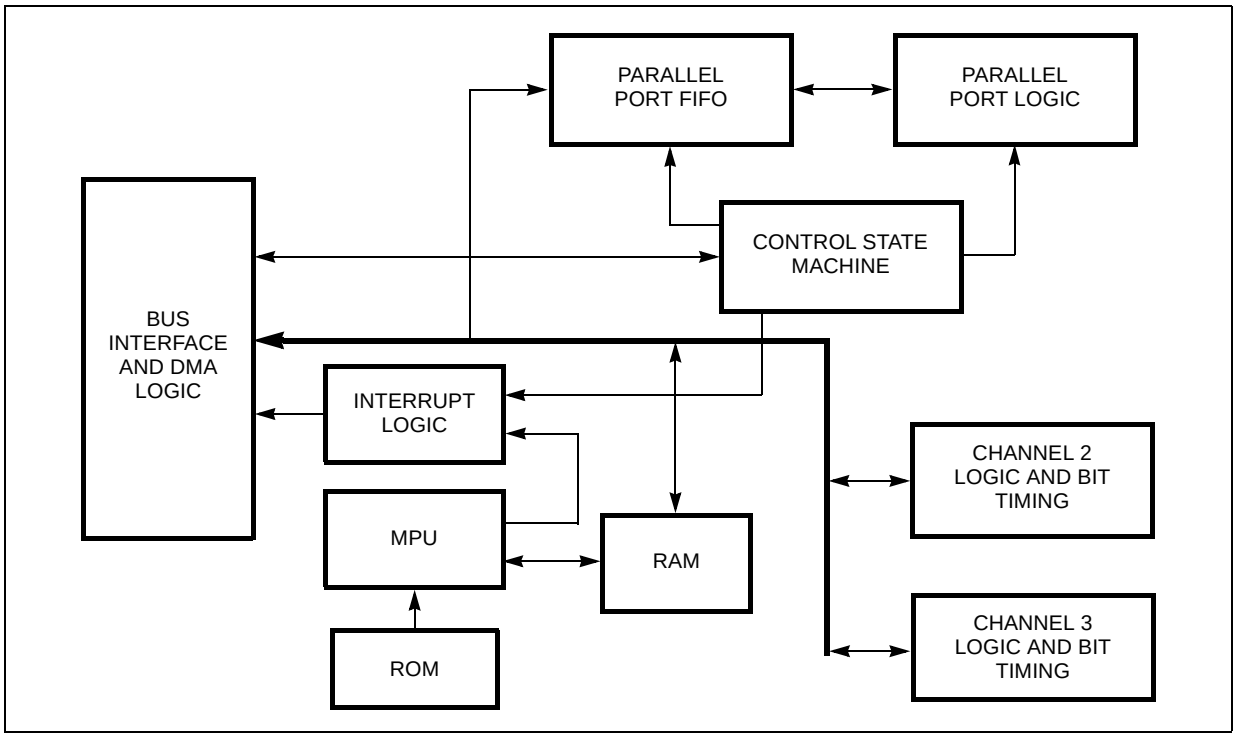
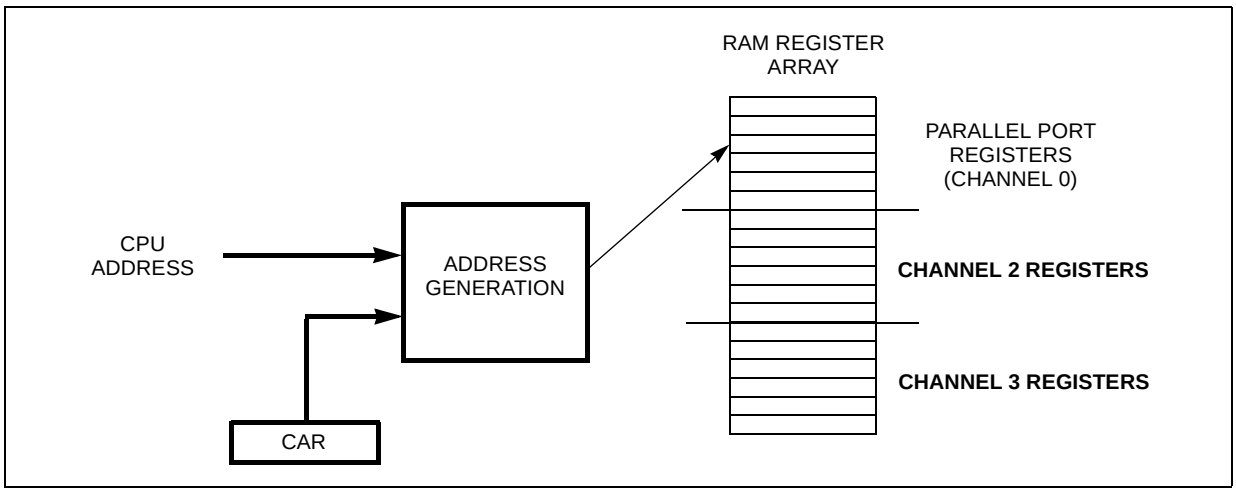


Figure 4. Internal Address Generation



The serial data channels are made of ‘bit engines’ that off-load the task of receiving and transmitting each bit from the MPU. When receiving data and after processing a complete bit, the bit engines interrupt the MPU so that it can perform the next required task. For example, the MPU takes the bit and adds it to a character being assembled. When transmitting, it sends the bit engine the next bit of the character being transmitted. The MPU is not concerned with basic bit timing; this task is handled by the bit engines, leaving the MPU free to perform higher-level processing, such as detecting special characters.

As described above, Channel 0 is a separate entity comprised of its own FIFO and DMA data interface, as well as a high-speed state machine that handles all of the modes defined in the IEEE STD 1284 specification. Channel 0 performs the slave, or peripheral, function of the IEEE STD 1284 interface and can be programmed to accept negotiations into any or all of the defined modes. The MPU aids the parallel port by providing the local access (through the CAR) and provides interrupt support (generation and response). However, this is the only action where the MPU is involved in parallel port service-request activities.

5.2 CPU Interface

The CPU interface comprises an 8-bit bidirectional data bus, a 7-bit address bus, a 16-bit DMA port and control inputs to identify the type of I/O cycle occurring. Although the strobe names and basic timing match that of the Motorola® 68000 family, the CD1284 fits easily into any CPU environment.

In most cases, when the CPU reads or writes an internal CD1284 location, it actually accesses a location in a RAM array to serve as a bank of registers. Some locations however, are mapped to actual hardware resources for example, when a hard output signal is required (such as a service-request output in the SVRR) or when it is necessary to read the actual state of an input (such as a modem input).

The CD1284 is a synchronous device. All internal operations occur on edges and levels (phases) of the internal clock. The internal clock is generated by dividing the external (system) clock by two. When the CPU performs an I/O cycle with the CD1284, it strobes; address, and data are sampled on the rising edges of the internal clock. As illustrated in Chapter 8.0, the external control signals must meet setup times with respect to system clock edges. Once a cycle starts, the sequence of events is locked to the clock of the CD1284. With events (address setup, write data setup, and read data available) occurring at predictable times.

It is not necessary to design a synchronous interface to the CD1284. In an asynchronous design, the DTACK* (Data Transfer Acknowledge) signal indicates that the CD1284 has completed the requested data transfer for all I/O cycles except DMA. DTACK* can be an input to wait-state generation logic that pauses the CPU until the operation is complete. If the CS* and DS* strobes (Chip Select and Data Strobe) do not meet the minimum setup time with respect to the system clock edge, the CD1284 does not detect the I/O request, and the cycle delays for two full-system clock cycles, meeting the setup time. The I/O cycle commences and follows the predictable timing with DTACK* signaling the end.

5.2.1 Read Cycles

Read cycles are initiated when both the CS* and DS* inputs are activated and the R/W* (read/write) input is high. All strobes and address inputs must meet the setup times as specified in Chapter 8.0. Both the CS* and DS* signals must be valid for a cycle to start. Cycle times are measured from whichever of the two signals goes active last. The CD1284 signals the completion of the read cycle (placing the data from the addressed register on the data bus pins) by activating DTACK*. The read cycle terminates when the CPU removes CS* and DS*.

5.2.2 Write Cycles

Write cycle timing and strobe activity is nearly identical to read cycles except that the R/W* signal must be held low. Write data, strobes, and address inputs must meet setup and hold times as specified in Chapter 8.0. DTACK* indicates that the cycle is complete and the CD1284 has accepted the data. Removing both CS* and DS* terminates the cycle.

5.2.3 Service-Acknowledge Cycles

Service-acknowledge cycles are a special-case read cycle. Timing is basically the same as a normal read cycle, but one of the SVCACK* inputs is activated instead of the CS* input (a slightly longer setup time is required on the SVCACK* input than on the CS* input). The data that the CD1284 provides during the read cycle is the contents of the Interrupt Vector register associated with the type of request being acknowledged (RIVR for receive, TIVR for transmit, MIVR for modem, and PIVR for parallel port) of the channel requesting service (see Section 5.3.1 for more information). As with read and write cycles, DTACK* indicates the end of the cycle. When the CPU removes DS* and SVCACK* the cycle terminates.

When the CPU has completed the service routine and writes to the EOSRR, a subsequent I/O cycle, if started immediately, is delayed by approximately 1 µs. This is due to the time required by the internal processor to complete activities associated with the switch out of the service-acknowledge context. These activities involve FIFO pointer updates and restoration of the environment prior to the service-request/service-acknowledge procedure. These must be completed before any internal registers are modified by the CPU.

If the situation occurs that the CPU attempts an access before the internal procedures are complete, the CD1284 holds off the cycle until it is ready. This does not cause a problem in system designs that monitor DTACK*; the cycle is extended until DTACK* becomes active and the delay is automatically met. If a system design does not monitor DTACK*, a mechanism must be provided to introduce the required delay.

Warning: Failure to observe the delay requirement can cause a device malfunction.

5.2.4 DMA Cycles

The CD1284 provides a bidirectional 16-bit DMA interface to the parallel port. This is the only direct data interface to the port; other 8-bit register accesses use of the normal CPU interface, as described above.

The handshake between the CD1284 and the DMA circuitry uses two signals: the DMAREQ* (DMA Request) and the DMAACK* (DMA Acknowledge). The address bus is ignored during DMA transfers. When internal conditions warrant a DMA transfer (as when the FIFO falls below the programmed threshold in the forward direction or rises above the threshold in the reverse direction) and DMA transfers are enabled by the PFCR, the device requests a DMA service by driving the DMAREQ* signal low. DMAREQ* remains active until the FIFO has less than two empty locations remaining (forward direction) or until the FIFO has less than 2 bytes remaining (reverse direction).

In the forward direction, the DMA controller logic responds by placing data on the 16-bit data bus and driving DMAACK* low. This cycle is repeated until the FIFO has less than two empty locations remaining or there is no more data to send. In the reverse direction, the CD1284 responds to the active DMAACK* signal by driving the contents of the DMABUF register onto the data bus.

Odd-byte transfers in the reverse direction are handled on an interrupt basis. When the number of bytes in the FIFO is odd, all bytes, except the last, are transferred by a number of 16-bit DMA cycles (two bytes per cycle). The odd byte remaining is held in the PFHR1 and an interrupt generated when the stale data timer expires. Status indicating that PFHR1 has data is shown in the PFSR. The CPU interrupt service routine must manually remove the remaining byte from the interface. In the forward direction, an odd remaining byte can be directly written to the PFHR1 once the last DMA cycle is complete.

One additional input signal determines the endian format (whether the least-significant byte is on data bits 7:0 or 15:8) of the 16-bit DMA buffer. BYTESWAP selects whether the lower or upper byte of the DMA buffer moves into the FIFO data pipeline first in the forward direction or from the FIFO data pipeline to the DMA buffer first in the reverse direction. If BYTESWAP is low, the least-significant byte (DB[7:0]) immediately moves into or out of the data pipeline. If BYTESWAP is high, the opposite occurs (DB[15:8] move into or out of the pipeline first).

The effective duration of the DMA transfer block (burst) is determined by the threshold value in the PFTR. Regardless of where the port is moving data, when this threshold is reached (exceeded in receive; less than in transmit) a DMA cycle begins and remains active until the FIFO has less than 2 bytes remaining (receive) or less than two empty locations remaining (transmit).

The SVRR provides a way to determine if a DMA cycle is being requested. SVRR[7] is true if a DMA cycle is currently being requested. This status indication is provided as a general system status.

Refer to Chapter 8.0 for detailed information on DMA cycle options and timing values.

5.3 Serial Port Service Requests

This section describes the service-request structure of the serial ports in the CD1284. Refer to Section 5.4 for a detailed description of the parallel port service-request architecture.

From the CPU point of view, the CD1284 operates in one of three modes: normal operation, service request/acknowledge, and DMA. Normal mode allows the CPU to make changes and obtain current operating status on a global and per-channel basis. Service-request/acknowledge mode determines when a particular channel requires service, for example, when a serial receive FIFO has reached its programmed threshold and requires emptying.

A unique behavior of the CD1284 is that a service request can only be responded to after the device is placed in a service-acknowledge ‘context’. This context switch occurs when the request is acknowledged, either by activating the appropriate SVCACK* input pin or by proper manipulation of two internal registers (software-activated mode).

When the MPU detects a condition on a channel that requires CPU attention, it posts a service request internally and externally. The external request is the activation of one of the SVCREQ* output pins, depending on whether the type of service needed is for receive, transmit, or modem signal change. Included with the internal request is a channel pointer to the channel requiring service. When the service acknowledge begins, this pointer is loaded into the CAR, thus the request automatically services the proper channel. This is the purpose of the context switch, it prepares the CD1284 for servicing of the proper channel.

At the completion of the acknowledge procedure, the CD1284 must be taken out of the acknowledge context by informing it that the procedure is complete. This restores the original internal state before the context change. This operation occurs after the CPU performs a ‘dummy’ write to the EOSRR.

Several registers within the serial channel portion of the CD1284 can only be accessed when the context switch has been made. These are the *Virtual* registers. For example, the CPU cannot place data directly in the serial transmit FIFO at an arbitrary time. It must wait for a transmit service request indicating that the FIFO is empty, then acknowledge it. Once the acknowledge procedure begins, the transmit FIFO is available for loading.

The CD1284 makes requests for service when an enabled need exists. The two basic ways that the CPU can be made aware of these service requests is through hardware (interrupt) or software (polling internal CD1284 registers). Which method is dependent on the hardware/software design of the system; the CD1284 functions well in either environment. The following section discusses the trade-offs of either basic method and how to combine the two for maximum performance.

5.3.1 Interrupts

The term interrupt is a generalized description of the method where the CD1284 gains the attention of the CPU. Interrupt is used interchangeably with ‘service request’ as the two are the same function. Interrupt often describes an unconditional response on the part of the CPU. Whether or not this is the case, the source is still the same — a service request from the CD1284. Hardware signals generated by the CD1284 (SVCREQR*, SVCREQT*, and SVCREQM*) can be connected to the CPU interrupt input to start an interrupt service routine. The service routine can then begin servicing the request from the CD1284 by starting an acknowledge sequence.

The SVCREQ* outputs can be connected to the interrupt circuitry individually using three unique interrupt-level inputs or they can be logically OR’ed together (not wire-OR’ed) into a single interrupt and applied to one interrupt-level input. In the latter case, the CPU can examine the SVRR to determine which service requests are active. The method (single or multiple interrupts) chosen by the designer is dependent on the system requirements and hardware and/or board-space limitations. The CD1284 has no restrictions. It is likely that interrupt latency is slightly shorter with the first method since the individual interrupt levels can cause a software vector directly to the correct service routine without first checking for the source of the interrupt.

No matter which interrupt method is used, the end result is the same. Once the CPU has recognized that a service request is active, a service-acknowledge routine must be executed to process the request. There are two ways to start the acknowledge and force the context switch: by four hardware input pins or by making specific reads/writes to internal registers.

5.3.2 DMAREQ* as Parallel Interrupt Source

Interrupts are not generated by FIFO threshold conditions; therefore, if the system design requires data to move through interrupts, connect DMAREQ* directly to a CPU interrupt input or logically OR it into the same CPU interrupt input as SVCREQP*. If DMAREQ* is used to generate interrupts, the following are required:

- A 16-bit data interface must be implemented to support 16-bit reads of the DMABUF register.
- The DMA threshold value in the PFTR must be initialized.
- DMAREQ* remains active until the FIFO is nearly empty (Rx) or nearly full (Tx), followed by the toggling of DMAen if data is moved to/from FIFO through PIO (refer to Section 5.2.4).

However, software can easily change this by clearing the DMAen bit (PFCR[6]) at the start of the interrupt service routine and resetting it at the end.

- If SVCREQP* and DMAREQ* are logically OR'ed together, the service routine must start by checking the SVRR to determine which signal is active.
- SVCACKP* must not be activated in response to DMAREQ* and likewise, DMAACK* must not be activated in response to SVCREQ*.
- The DMAdir bit (PFCR[5]) can determine whether to write or read to/from the DMABUF register.
- The PFQR can determine how many reads of the 16-bit DMABUF register are necessary to empty the pipeline. Note however, four must be added to the PFQR value, that number must then be divided by two and truncated to the nearest integer (to account for the extra four bytes in the two holding registers and the 16-bit DMABUF register, as well as 16-bit reads instead of 8-bit reads).

5.3.2.1 Hardware-Activated Context Switch — Serial Channels

The internal register manipulation involved in a context switch can be forced by SVCACK* (Service Acknowledge input pins on the CD1284). There is one SVCACK* for each service request type: SVCACKR* for receive service requests, SVCACKT* for transmit service requests, and SVCACKM* for modem signal-change service requests. Each of these inputs is a special-case chip select. These cause the MPU to set up the CD1284 for servicing that particular service request type for the requesting channel.

Note that the CS* input is not activated on service-acknowledge cycles. Instead, the appropriate SVCACK* input and the DGRANT* inputs are used. Later in this section, DGRANT* is discussed in a description about daisy-chaining the CD1284 with one or more CD1400s. Figure 5 shows a generalized logic diagram of the hardware interface to the SVCACK* inputs. For a service acknowledge, one of the SVCACK* address locations is accessed instead of the CS* location.

To the CPU, the service-acknowledge cycle is a read cycle. The data that the CD1284 places on the bus for an SVCACK* during the read cycle are the contents of the appropriate Interrupt Vector register (RIVR, TIVR or MIVR). These IVRs are associated with the active service-acknowledge input (SVCACKR*, SVCACKT*, or SVCACKM*). The upper five bits of the IVR are whatever was previously loaded into the LIVR by the CPU. The lower three bits are supplied by the CD1284 and indicate the type of interrupt (vector).

When the CD1284 is ready to post a service request for a serial channel, it copies the upper five bits of the LIVR into the appropriate vector register (RIVR, TIVR, MIVR), then places the request type vector in the lower three bits. Table 14 shows the assignment of the request type bits.

Figure 5. Control Signal Generation

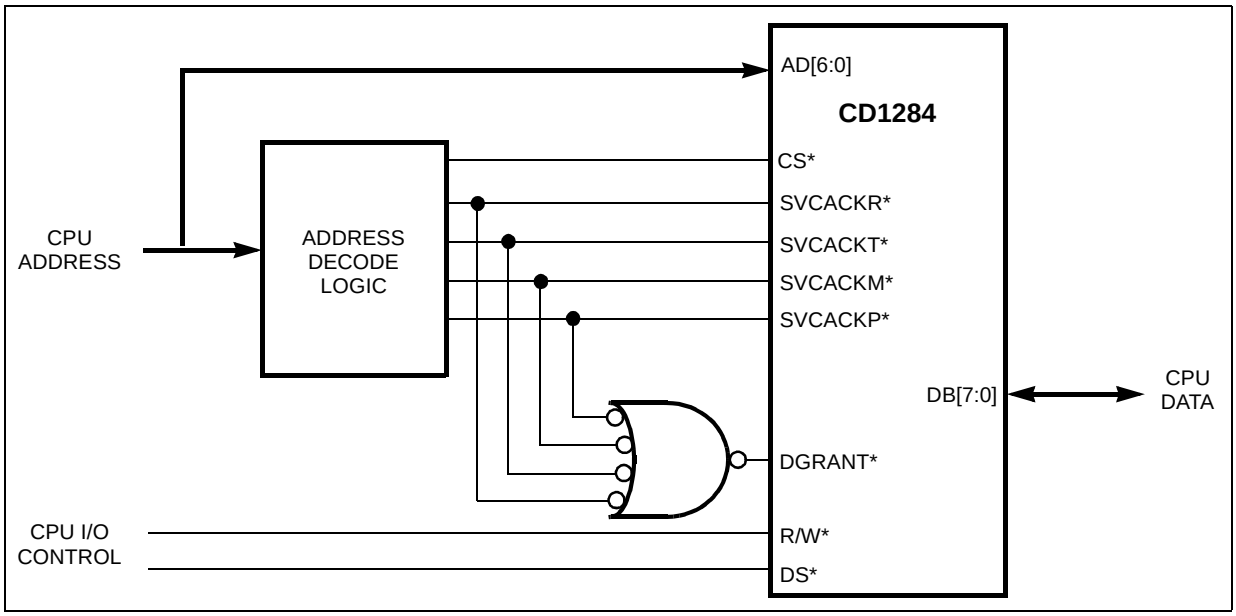


Table 14. Request-Type Bit Assignments

Bit 2	Bit 1	Bit 0	Request Type
0	0	0	Not used
0	0	1	Group 1: Modem signal change service request
0	1	0	Group 2: Transmit data service request
0	1	1	Group 3: Received good data service request
1	0	0	Parallel port state-machine requests service (refer to Section 5.4)
1	0	1	Parallel port data pipeline request service (refer to Section 5.4)
1	1	0	Both the parallel port state-machine and data pipeline request service (refer to Section 5.4)
1	1	1	Group 3: Received exception data service request

For transmit and modem service-acknowledge cycles, the data in the lower three bits is redundant to the software because the corresponding acknowledge has occurred. These bits are important in the case of a serial receive-data service acknowledge because they provide an indication of whether the request is for ‘good’ data or exception data. They are important to the parallel port because they indicate if the state-machine or data pipeline (or both) are requesting service.

The value contained in the upper five bits of the LIVR can be used for a number of purposes. The primary purpose of the LIVR is as a source of a software vector used by the system as an index into a interrupt dispatch table. However, systems that cannot use this or do not need it can use these bits for any purpose. In multiple–CD1284 designs that use daisy-chaining, a logical value to place in these bits is a chip identification number. This is detailed in the daisy-chaining description in Section 5.3.4.

Another use for these bits is channel encoding. This is applicable in a single-CD1284 design and any design not using daisy-chaining (requiring a unique address range for each device). This applies where the value in the LIVR as a vector for a hardware interrupt response is not necessary. Since each channel has its own LIVR, these five bits have a unique value identifying the channel. There is no need to read the RICR, TICR, or MICR to find the channel number; in a single I/O operation, the CPU determines both the type of interrupt and the number of the channel requesting service. With five bits available, systems with small numbers of CD1284s are able to encode both the channel number and chip identification number in the LIVR.

Once the acknowledge procedure is complete, the CD1284 is ready to be serviced for the type of interrupt acknowledged. For example, if the interrupt was for receive good data, the CPU would read the RDCR to determine the number of characters available in the receive FIFO. It then reads the same number of characters, by successive reads, from the RDSR. Other tasks, such as disabling future interrupts or changing channel parameters, could also be performed at this time.

Once all tasks involved in servicing the interrupt are complete, one more operation is performed. To inform the CD1284 that the service acknowledge is complete, the CPU writes a dummy value to the EOSRR. Although the data written does not matter, the write operation is important. This write forces the internal context switch back to normal operating mode.

5.3.2.2 Summary of Interrupt Driven Service Requests, Serial Channels

The actions that occur during an interrupt request/service are:

- The CPU senses service request from one of the CD1284 service-request outputs through its interrupt request input.
- The CPU responds by performing a read cycle to activate the appropriate SVCACK* input pin.
- The CPU decodes the value read from the vector register during step 2, and decides on the type of service request (if necessary).
- The CPU reads the R/T/M/ICR to determine the channel number.
- The CPU services the request (load transmit FIFO, read receive FIFO, and so on).
- The CPU writes a dummy value to the EOSRR to terminate the service routine.

5.3.2.3 Common Service Acknowledge

One method of hardware-activated, service-acknowledge request is the common service acknowledge. In this method, all SVCACKx* inputs are tied together and are driven from the same source. In this configuration, the CD1284 internally prioritizes the acknowledge as receive, transmit, parallel, and modem. If a device has both a receive and a parallel request pending, the common acknowledge causes it to respond with the vector for the receiver. Then a subsequent service acknowledge allows the parallel channel request to be serviced.

5.3.2.4 Software-Activated Context Switch — Serial Channels

It is possible, by CPU manipulation of some internal registers, to cause the context switch without activating any of the SVCACK* hardware inputs. The method is the same used in the poll-mode–CD1284 design. Once the CPU has detected the service request through its interrupt response circuitry, it follows the same procedures that a polling method uses when it detects an active service request. Refer to the context switching description in the following section.

One reason a design might make use of this method is that limited board space is available for the additional hardware address decoding required to generate the four SVCACK* and DGRANT* control signals. The advantage is that the system need not constantly poll the CD1284 for active service requests. It is interrupted when a request is posted, then examines internal CD1284 registers to determine the source and channel number generating the request. For this method, tie the four SVCACK* and DGRANT* input pins inactive (logic ‘1’). This prevents possible false activation of a service-acknowledge cycle that occurs due to noise. Terminate these pins with a resistor (approximately 1 kΩ) not hardwired to V_{CC}.

5.3.3 Serial Service Request Polling

In Poll mode, the CPU periodically checks the CD1284 to see if there are any active service requests. If it detects any, it proceeds to service them by a software-driven technique. There are several registers within the CD1284 specifically provided to facilitate Poll-mode service-request detection and acknowledgment. These are the SVRR, RIR, TIR, PIR, MIR, RIVR, TIVR, and MIVR. Chapter 7.0 provides detailed bit definitions for these registers.

The SVRR is the master service-request register. The least-significant three bits (bits 2:0 — SRM, SRT, and SRR) reflect the inverse of the state of the three service-request output pins (SVCREQM*, SVCREQT*, and SVCREQR*). For example, if SRR[0] is ‘1’, it indicates that there is a pending active serial receive data service request, and that the SVCREQR* output pin is active (low). The CPU now can determine with a single read if the CD1284 requires any service and which pins are active.

Each service request type has an interrupt request register: RIR for receive, TIR for transmit, and MIR for modem. These are the special purpose registers used with the CAR to force the context switch and start a service-acknowledge procedure. When a service request of a particular type is pending, the corresponding Interrupt Request register is set by the MPU with the appropriate data to cause the context switch to the requested type and the requesting channel.

When the CPU is ready to service the request, it reads the contents of the request register and copies it into the CAR. This write into the CAR forces the context switch and the CD1284 is ready to be serviced. The result is the same as performing a service-acknowledge cycle with the SVCACK* pin.

Each of the Interrupt Request registers provide the channel number by requesting service in the least-significant two bits. The most-significant three bits provide status and control over internal interrupt sequencing. The middle three bits contain a code used by the MPU at the end of a hardware service-acknowledge cycle (write to the EOSRR) to indicate the type of acknowledge cycle that is ending. Each of the three registers has a unique code in these three bits to select the proper service-acknowledge type, but these are meaningless in Poll-mode operation.

At the end of a service-request operation, the CPU must inform the CD1284 that the request is satisfied and to take it out of the service-request context. This is done by rewriting the value that was in the interrupt request register after clearing the upper two bits.

As with the hardware-driven request/acknowledge procedure, the Virtual registers should only be accessed after the context switch is made. Their contents are undefined until this time.

5.3.3.1 Summary of Serial Poll-Mode Service Requests

The major steps involved in a Poll-mode service-request/service-acknowledge sequence are:

1. The CPU scans the SVRR periodically, checking the three least-significant bits. If any of them are true (‘1’), a service request is active.
2. Depending on which of the service-request bits is active, reads the appropriate interrupt request register (RIR, TIR, or MIR) and copies the contents into the CAR.
3. Performs a service routine.
4. Writes the original contents of the interrupt request register back with the most-significant two bits cleared.

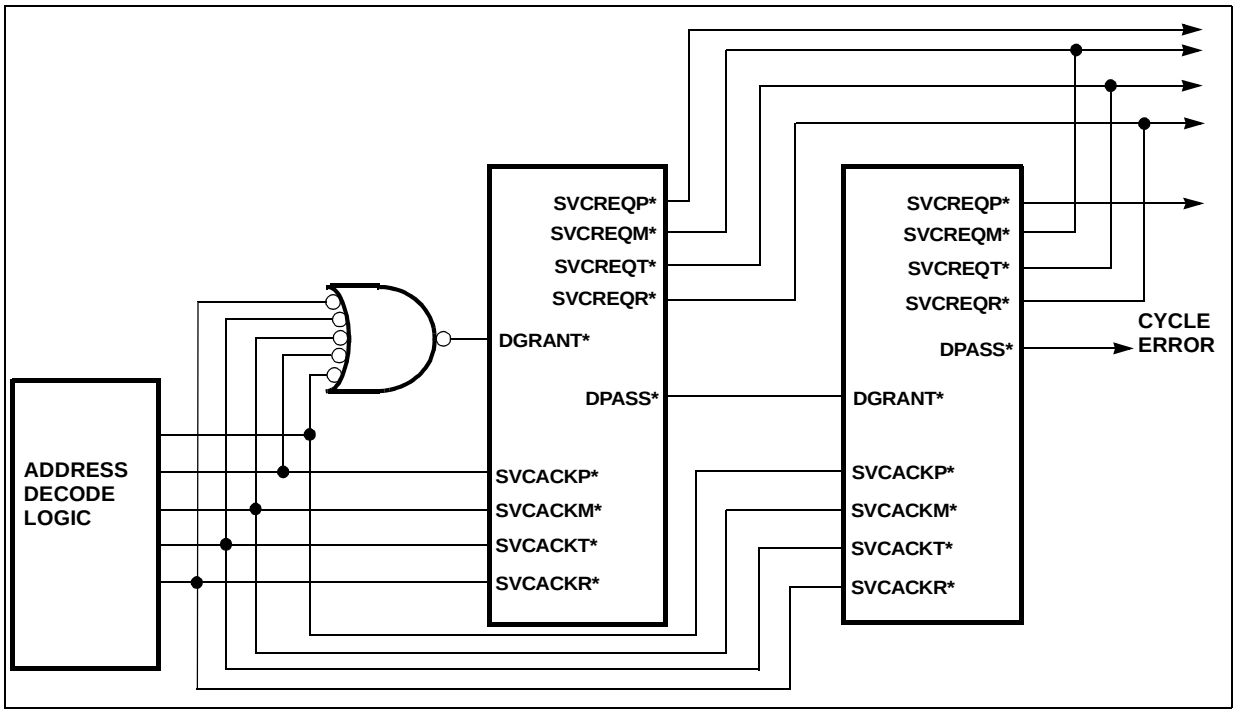
5.3.4 Daisy-Chaining Service Requests with CD1400s

The CD1284 can be combined with other CD1284 or CD1400 devices to form systems with more than two serial channels and one parallel channel. There are a number of ways that these can be connected, but one way provides a more efficient service-request/service-acknowledge sequence. This method allows the CD1284s and/or CD1400s to arbitrate between themselves. This mode only works if hardware-activated service acknowledges are being utilized. The Fair Share mechanism is not functional on the parallel channel service-request (SVCREQP*) outputs. Therefore, two CD1284s can be daisy-chained if SVCREQP* and SVCACKP* are kept separate. The serial channel requests and acknowledges are identical to those on the CD1400 so they can be connected to the equivalent requests and acknowledges on the CD1284.

The CD1284 provides a means of daisy-chaining the service request and service acknowledgments of two or more devices. This allows them to arbitrate and set priorities between themselves regarding which one can post a particular type of service request. This is the Fair Share interrupt scheme. Figure 6 on page 42 illustrates the connection for two CD1284s to enable the Fair Share function.

All request outputs of a particular type from the two CD1284s (SVCREQR*, SVCREQT*, and SVCREQM*) are wire-OR’ed together to form one combined request for each type; the SVCREQP* of each is kept separate. This allows both devices to monitor the state of the others output. All of the serial service-acknowledge inputs (SVCACKR*, SVCACKT*, and SVCACKM*) are connected together to form one acknowledge of each type. Note, the SVCACKP* are driven individually. The DGRANT* input of the first CD1284 is connected to ground; the DPASS* output of the first CD1284 drives the DGRANT* input of the second.

Figure 6. CD1284 Daisy-Chain Connections



Before a serial request for service of a particular type is posted, the MPU checks the current state of the request output for that type. If it is inactive, indicating that no other CD1284 is driving that level, a request can be posted; otherwise it waits. This guarantees that each CD1284 has an opportunity to have a request type serviced when required. When the CPU acknowledges the request, both CD1284s receive the acknowledge through SVCACK*. However, only the first receives DGRANT*. If there is an active request of this type pending, the CD1284 takes the acknowledge and drives its vector register (RIVR, TIVR, MIVR) onto the data bus.

If the first device does not have a request pending, it passes the DGRANT* input to the second CD1284 through the DPASS* output. Assuming that the second device has an active request pending, it takes the acknowledge and drives its Vector register onto the data bus.

As previously mentioned, the upper five bits of the LIVR reflects what the CPU loaded into them during its initialization of the CD1284s. These bits are used as a unique chip identification number so the CPU can determine which CD1284 responded to the service acknowledge. These five bits can be set to binary ‘0’ in the LIVRs of the first CD1284, and to binary ‘1’ in those of the second. The CPU is able to test the bit to determine which device responded. Some examples of service-acknowledge software routines that show one way of performing this task are provided in Chapter 6.0.

The common service acknowledge described in Section 5.3.2.3 on page 39 is also usable in daisy-chained environments. In this case, the common acknowledge is applied to all service-acknowledge inputs in all devices of the chain. The daisy-grant ripples down the chain until the requesting device receives the acknowledge.

Note: If a CD1284 further down the chain is requesting service for a receiver and one up the chain is requesting service for a transmitter, the transmit request is serviced first since it precedes the receive requester. Thus, the Fair Share mechanism is not functional in this configuration.

The CD1284 has a fairness override, the Unfair bit (PACR[0]). If this bit is set, the Fair Share function of the device is defeated and the MPU posts requests for service regardless of the state of the external service-request signal. Even when a device in the chain is asserting a request of a particular type, if another device needs to post a request, it proceeds to do so regardless of the current state of the request pin because its fair bits are forced true. If it is upstream from the device already posting the request and if the CPU pipeline has not yet responded to the previous request from the downstream device, then the upstream device accepts the acknowledge on arrival and overrides the priority normally given to the device that made the first request. This is useful in system designs that wire-OR the request signals together, rather than using an external gate, since in these cases, without overriding fairness a request of one type within a device holds off a request of a different type. For example, an existing transmit request prevents the device from posting a receive request.

Note: (***IMPORTANT***) If no CD1284 in the chain has a pending request, the daisy-grant passes by the last and none respond. This causes the bus cycle to hang (no DTACK* is generated). The only time this happens is when an error condition outside the CD1284s cause the CPU to respond to a request that is not made. A mechanism can be provided to terminate or abort the bus cycle if this error occurs. This is accomplished with timeout circuitry. Otherwise the DPASS* output of the last CD1284 activates an abort condition. Other devices, such as the CD1400, can share the daisy-chain mechanism and can be connected to the DPASS* output of the last CD1284 in the chain. The actual implementation is system-dependent, but it is important to provide some way for the CPU to know that the cycle did not complete normally if no device responds to the acknowledge cycle.

5.4 Parallel Port Service Requests

The parallel port service-request structure of the CD1284 is slightly different from that of the serial ports. These differences are highlighted in this section.

Service requests can derive from two internal sources: the data pipeline or the parallel port state machine (see Figure 7 on page 45). If the data pipeline internal service request becomes active, the Pipeline bit (PIR[5]) is set; likewise, if the parallel port state machine internal service request becomes active, the PPort bit (PIR[6]) is set. Internal service requests from these sources are monitored through the Pipeline and PPort bits by microcode running in the internal MPU. When either (or both) of these bits are detected active, the microcode sets the PPireq bit (PIR[7]). The PPireq bit is also mirrored by the SRP bit (SVRR[3]). The SVRR is useful in polled systems because it allows the detection of DMA service requests, as well as parallel port service requests with a single register read operation.

Both internal sources of service requests within the parallel channel have their own enable functions. Interrupts from the data pipeline are enabled through the PFCR; interrupts from the parallel port state machine are enabled through the PCIER.

The PFCR has two enable bits: one for normal interrupts (such as tagged data being received), and one for data errors (such as a CPU write to a holding register that already holds data). The first type of interrupt is enabled through the IntEn bit (PFCR[4]). The second type of interrupt is enabled through the ErrEn bit (PFCR[1]). Note that IntEn must be set for ErrEn to generate an interrupt; however, the CPU need not enable error interrupts if it does not require notification of these types of errors. The error interrupt is generated if the DataErr bit (PFSR[0]) is a non-zero. In this case, the DER indicates the cause of the error interrupt.

The parallel channel-control state machine can generate six types of interrupts. Each of these has its own enable bit in the PCIER:

- NegCh for negotiation changes
- SigCh for signal changes on the port status inputs (Manual mode only)
- EPPAW for EPP protocol address writes
- DirCh for direction changes on the parallel channel
- IDReq for slave ID requests from the remote master.
- nINIT for initialization pulses from the master (Compatibility mode only)

Any or all of these bits may be set, based on the mode of operation.

The NegCh interrupt is issued whenever the remote master performs a protocol change, such as moving from Compatibility mode to ECP; the CPU examines the NSR to determine the new state of the parallel interface. Signal changes can be identified by reading the SSR. In response to the EPPAW interrupt, the CPU would read the EAR to retrieve the value that was written during the EPP address write cycle.

Figure 7. Interrupt Generation Logic

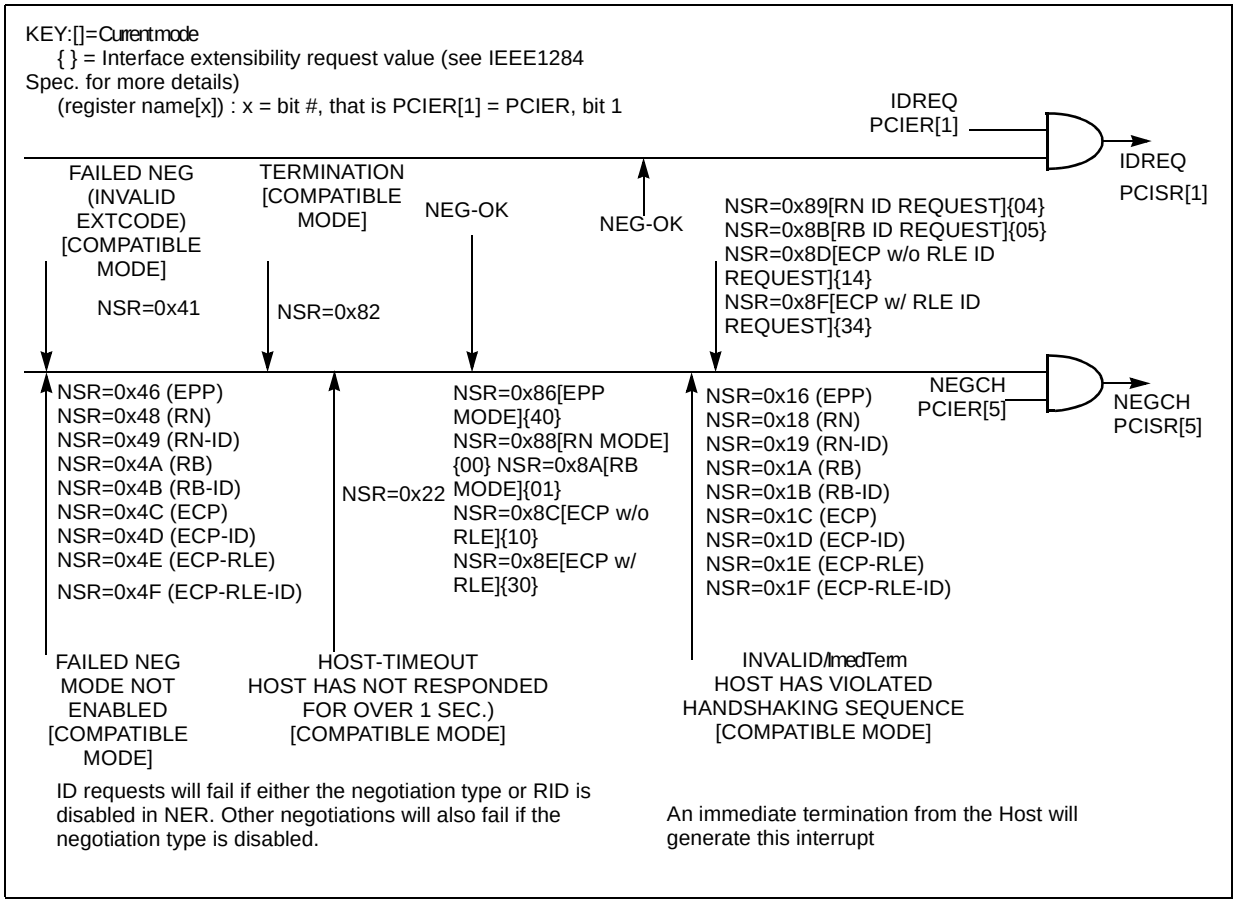


Figure 7. Interrupt Generation Logic (Continued)

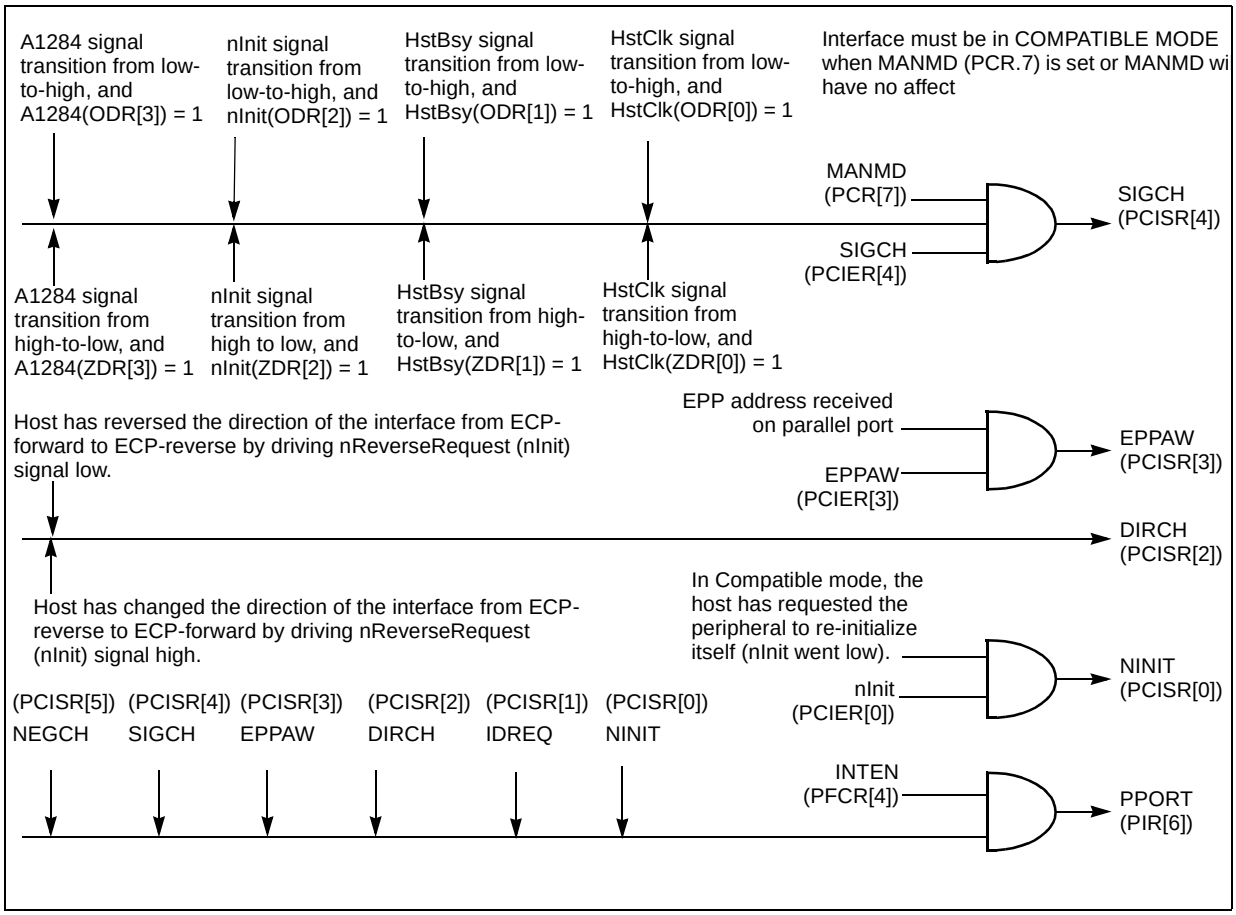
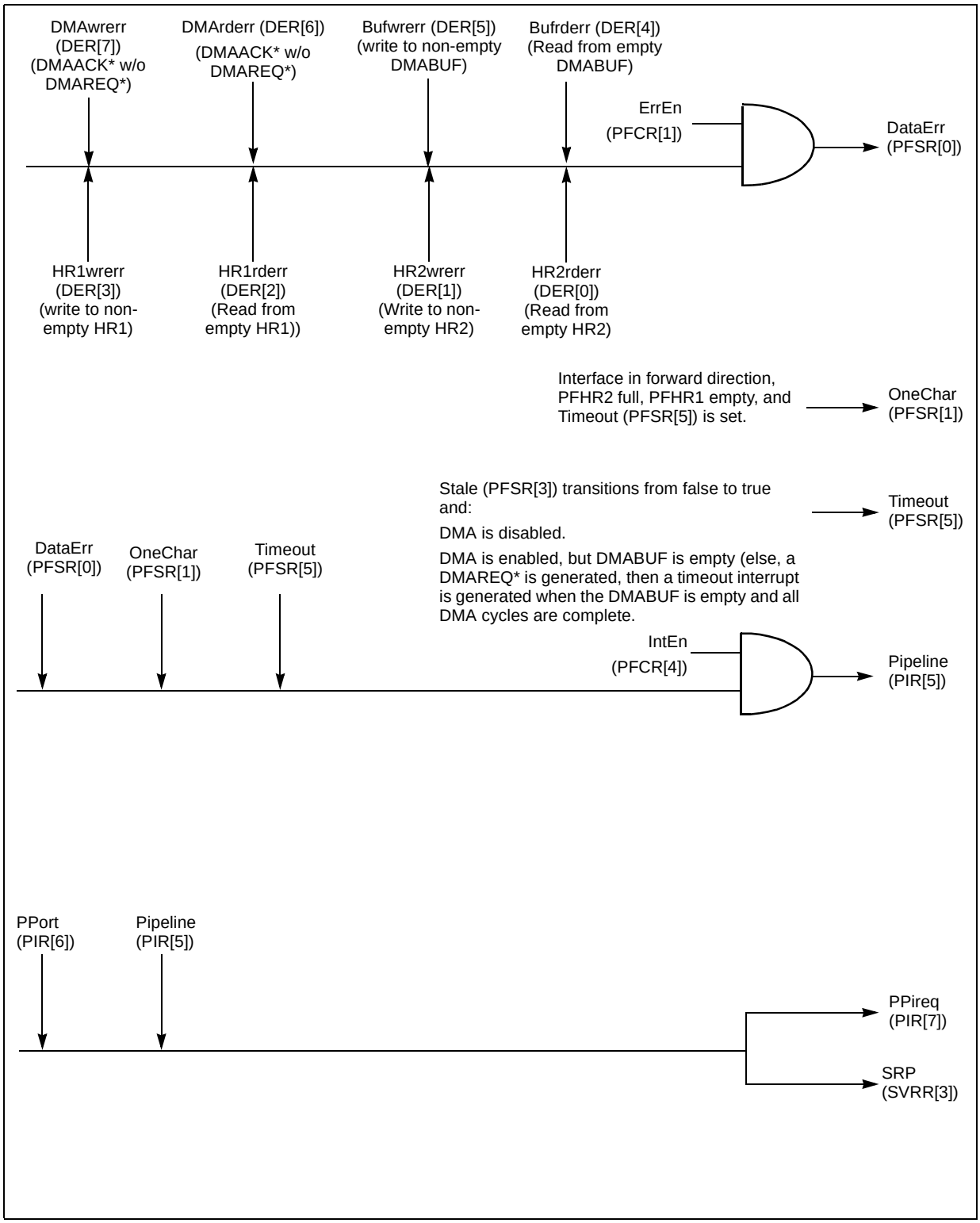


Figure 7. Interrupt Generation Logic (Continued)



A direction change (DirCh) interrupt occurs when the remote master has reversed the interface from ECP forward to ECP reverse or ECP reverse to ECP forward. The IDReq interrupt is generated when the remote master issues an ID Request command during IEEE 1284 negotiations. The normal response by the local CPU is to send its ID string after reversing the direction of the data pipeline by setting the DMAdir bit to ‘1’.

In an interrupt-driven system, as with the serial channel requests, the SVCREQP* output normally connects to one of the local CPU interrupt control inputs. It can also be OR’ed together, through an external gate, with the serial request outputs to produce a single interrupt request to the local CPU. The interrupt service routine scans the SVRR and determines the actual source of the interrupt.

The parallel channel has the same Vector register arrangement as the serial channels. The LIVR must be initialized by the local CPU in the same manner as the serial channels; the upper five bits are defined by the local CPU and can be any value appropriate to the system design. The lower three bits should be initialized to zero during the programming of the LIVR, however they are ‘don’t cares’ and masked in the PIVR to provide the vector indicating the source and type of request from the parallel channel.

Access to the parallel channel LIVR is made by first setting the CAR to ‘x’00’, making the Channel Zero register set accessible. Since the LIVR is a read/write register, the local CPU can read it at any time. When read during a normal read cycle, it returns the original value written to it. When a service acknowledge is performed, the upper five bits of LIVR are copied into PIVR.

The encoding of the three least-significant bits of PIVR during a service acknowledge cycle indicates which of the functional blocks in the parallel channel is requesting service and is as follows:

IT2 (Bit 2)	IT1 (Bit 1)	IT0 (Bit 0)	Requestor
1	0	0	Channel control state machine
1	0	1	Data pipeline
1	1	0	Both

The encoding of the parallel channel service-request status was designed using the remaining unused states of the CD1400: ‘100’, ‘101’, and ‘110’. The other states of these three bits are already used to indicate serial interrupt status in RIVR, TIVR, and MIVR.

5.4.1 Hardware-Activated Context Switch, Parallel

When conditions within the parallel channel require attention, a request is made by the SVCREQP* output. If the system is interrupt driven, this output would be connected to the CPU interrupt generation circuitry. In a hardware-activated service-acknowledge system, the CPU responds to the request by activating the SVCACKP* input (along with DGRANT* and DS*) in the same manner as the serial channels; the CS* input is not used and must remain inactive (high). The CD1284 responds to the SVCACKP* cycle by driving the contents of the PIVR onto the data bus with IT2–IT0 encoded as shown above. The SVCACK cycle also places the device in the correct context to service the parallel channel request.

The vector supplied by the PIVR indicates which block of the parallel channel requested service; the cause of the request is indicated in the Request Status registers of each: the PCISR in the channel control state-machine block and/or the PFSR in the data pipeline block. Refer to Chapter 7.0 for detailed descriptions of the various status bits in these registers.

The I/O cycle that activates the SVCACKP* input also removes the active SVCREQP* output. The request output is inactive until after the CPU terminates the acknowledge routine by writing to the EOSRR. As with the serial channels, this is a dummy operation and the data written is ‘don’t care’. The purpose of the write is to clear the internal logic of the current request context and allow it to generate another request when the need arises. Until this write occurs, no further service requests are made from the parallel channel. When the MPU detects the write to the EOSRR, it zeros-out the PIVR in preparation for the next service-request cycle.

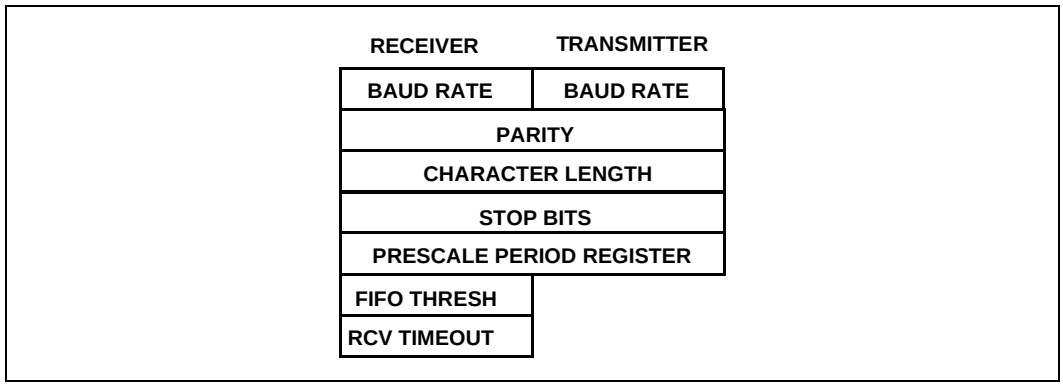
5.4.2 Software-Activated Context Switch, Parallel

Software-activated acknowledges of the parallel channel differ somewhat from those of the serial channels. The start of a software acknowledge of the parallel channel is the same as for the serial channels: the CPU copies the contents of the PIR into the CAR (after first saving the current contents of the CAR) to set the device context. However, at this point the methods (serial versus parallel) diverge. The CPU can read either the LIVR or PIVR (or read the status from the two status registers in the Parallel Port register set) to determine which of the parallel channel blocks is requesting service, copy the PIR into the CAR (or just load it with ‘x’00’) to set the context, then proceed to service that request. Once the CPU has satisfied the request needs of the parallel channel, it must toggle the IntEn bit (PFCR[4]) or clear the PIR. Toggling IntEn clears the PPort and Pipeline bits and the PPIreq bit (PIR[7]). This action informs the MPU to clean up the PIVR and remove the external request. The software should then restore the CAR to its previous contents and exit the service routine.

The PPIreq bit can be cleared at any time by the CPU. If the system design requires the request be removed quickly, the procedure can be performed at the beginning of the polled service routine. If the CPU waits until the end of the service routine, it clears the bit itself or terminates the service in the manner described, letting the MPU do it.

5.5 Serial Data Reception and Transmission

The CD1284 has two serial channels, each with a receiver and a transmitter. Although a receiver and a transmitter pair are associated with each channel, in many respects they operate independently, sharing only parameter settings regarding character format including length, parity type if any, and number of stop bits. Each receiver and transmitter has its own baud rate generation function, allowing a channel to send at one rate and receive at another. Shared and independent parameters are shown in the following diagram.



Channel service needs, such as an empty transmit FIFO, are indicated to the CPU by one of three service-request indicators: one for all receivers, one for all transmitters, and one for all modem signal changes. The internal processor (MPU) scans each channel sequentially for service needs, posting a request when it detects a particular type. It continues the Fair Share scheme used in the external daisy-chain configuration by not allowing a channel to post another request of one type until all other channels have posted their requests of that type, if any. For example, if channel two is currently being serviced for a transmit request and channel three has one pending, the request from channel three is posted before channel two is able to make another request for transmit service.

Each receiver and transmitter has a 12-character FIFO. The receiver has two additional character holding locations: the Receive Character Holding and Receiver Shift registers. The transmitter also has two additional locations, the Transmitter Holding and Transmitter Shift registers. The receive FIFO has a programmable threshold that sets the level at which a service request is posted. When data reaches this FIFO-full threshold, a request is made of the CPU to empty the FIFO (for details see Section 5.5.1). Receive FIFOs also have a programmable threshold that, when reached, causes the DTR output to be deasserted (see the flow-control description).

In the asynchronous serial data protocol, a message consists of one ‘character,’ made up of bits, either high or low, representing a ‘1’ or ‘0’ value. A character can be from five to eight bits plus an optional parity bit bracketed by a start bit and a stop bit. Each bit has a time duration that sets the data transmission rate — or baud rate. The start bit indicates the beginning of a character bitstream and is indicated by a transition from a logic ‘1’ to a logic ‘0’ (mark to space) on the transmission media. The start bit lasts one ‘bit-time’ and is immediately followed by the data bits (8:5), the parity if any, and the stop bit.

As previously discussed, the CD1284 incorporates special hardware to receive and transmit each bit. These are the ‘bit engines’. They perform all timing associated with sending or receiving one serial data bit. A bit engine behaves differently depending on whether it is sending or receiving. When a complete bit is received, the bit engine interrupts the MPU so that it can handle the bit on the character level. This usually entails its addition to the character being assembled. For transmitting, a transmit bit engine interrupt causes the MPU to give it the next bit to transmit. The bit engine interrupt occurs at the end of a bit time that is timed by the engine, thus removing that duty from the MPU.

5.5.1 Receiver Operation

Each channel can be programmed to receive characters with several different parameters, such as character length, parity, number of stop bits, FIFO threshold, and baud rate. Each receiver is independent of any other receiver. It can also be set to a different baud rate from its corresponding transmitter.

Before valid data can be received, the CPU must set up each channel by programming the desired operational parameters in the COR1–COR5, the BRRR, RCOR, and RBPR. Once these registers are set, the channel is enabled by issuing the receiver enable command through the CCR and enabling service requests in the SRER.

Once a receiver is enabled, its bit engine begins to scan the RxD input for a valid start bit. It does this by detecting a falling edge transition on the input. When the transition is detected, the bit engine delays until the middle of the programmed bit time and rechecks the input. If the input is still low, the start bit is considered valid and character assembly begins. At each subsequent full bit time, the input is checked and its level recorded as the value of the next bit. If, at the center of the bit time, the RxD input returns to a mark state, then the start bit is considered invalid and the bit engine returns to the start bit detect mode.

Following a valid start bit, the bit engine begins receiving data bits. At the end of the programmed number of bits, following bits are checked for parity (if enabled) and a valid stop bit. A valid stop bit is defined as a mark or logic ‘1’ on the input. If a valid stop bit is not detected, a framing error is noted for the character. After a properly assembled (no framing error) character has been received, it is checked for several special conditions (see Section 5.6 and Section 5.7) and the overrun condition before it is placed in the receive FIFO. If no errors or special character processing is required, the character is considered ‘good’ data and placed directly in the FIFO. If errors exist, it is placed in the FIFO as ‘exception’ data along with status indicating the type of error. As each good character is placed in the FIFO, the RDCR (Receive Data Count register) is updated to reflect the number of good characters currently in the FIFO.

The receive FIFO has a programmable threshold to determine the level where the CD1284 requests receive data service. This level is programmed through the RxTh[3:0] bits (COR3[3:0]). The CPU can set the threshold to any number of characters from 1 to 12.

Note: This only sets the level where the CD1284 posts a service request and not the depth of the FIFO.

When the CPU responds to a receive good data service request, it can read any number of characters out of the FIFO, from zero up to the number indicated in the RDCR before exiting the service routine. If the number read is zero, the CD1284 posts another request for service almost immediately. If the number of characters read is less than the number indicated by the RDCR, but enough so that the number in the FIFO falls below the threshold, a new request is not made until the threshold is once again exceeded. Since the MPU circularly scans the channels, another channel can post a receive service request before this channel has the opportunity, this is why the request for service is posted ‘almost immediately’.

5.5.2 Receiver Timer Operations

Also associated with each receiver FIFO is a timer that has its duration set in the RTPR. This timer provides two services in relation to the receive FIFO operation: a timeout to prevent ‘stale’ data in the FIFO and a timeout after the last character is removed from the FIFO.

The first type, type 1, occurs if the receive FIFO does not reach the set threshold before the programmed time period expires. The second type, type 2, occurs if the timer expires and no new data has been placed in the FIFO after the last character is removed — this is the NNDT (No New Data Timeout) service request.

The timer is driven by the prescaled clock selected in the PPR in the Global register set. This timer is loaded with the value contained in the RTPR each time a character is placed in the receive FIFO or when the last character is removed from the FIFO. Each ‘tick’ of the prescaler decrements the timer. If the timer reaches zero and the receiver interrupts are enabled, the MPU generates a receive data service request for the valid timeout condition.

Type 1

If there are characters in the FIFO but the threshold level has not been reached, a good data service request is posted when the timer expires. This function is provided to prevent data from remaining in the FIFO for long (potentially infinite) periods of time because the remote did not send enough data to fill the FIFO to the threshold level. This timeout cannot be disabled.

Type 2

If there is no data in the FIFO when the timer expires and the NNDT service request is enabled in the SRER, a receive exception service request is posted with status indicating the timeout condition. This timeout is optional and is provided so that driver software can detect the possible end of a block of data and allow its buffers to be flushed to the higher, operating system level. The NNDT is posted only on the first occurrence of a timeout after the FIFO becomes empty. Also note that the NNDT timer is not started if the last character removed from the FIFO was an exception character, such as a break or parity error.

Figure 8 on page 53 shows the timer process evaluation performed by the MPU when the timer reaches zero.

5.5.3 Receive Exceptions

Several conditions can cause the CD1284 to post the receive exception service request. If an exception condition occurs, two bytes are placed in the receive FIFO. The first byte contains the status indicating the type of error; the second byte contains the data.

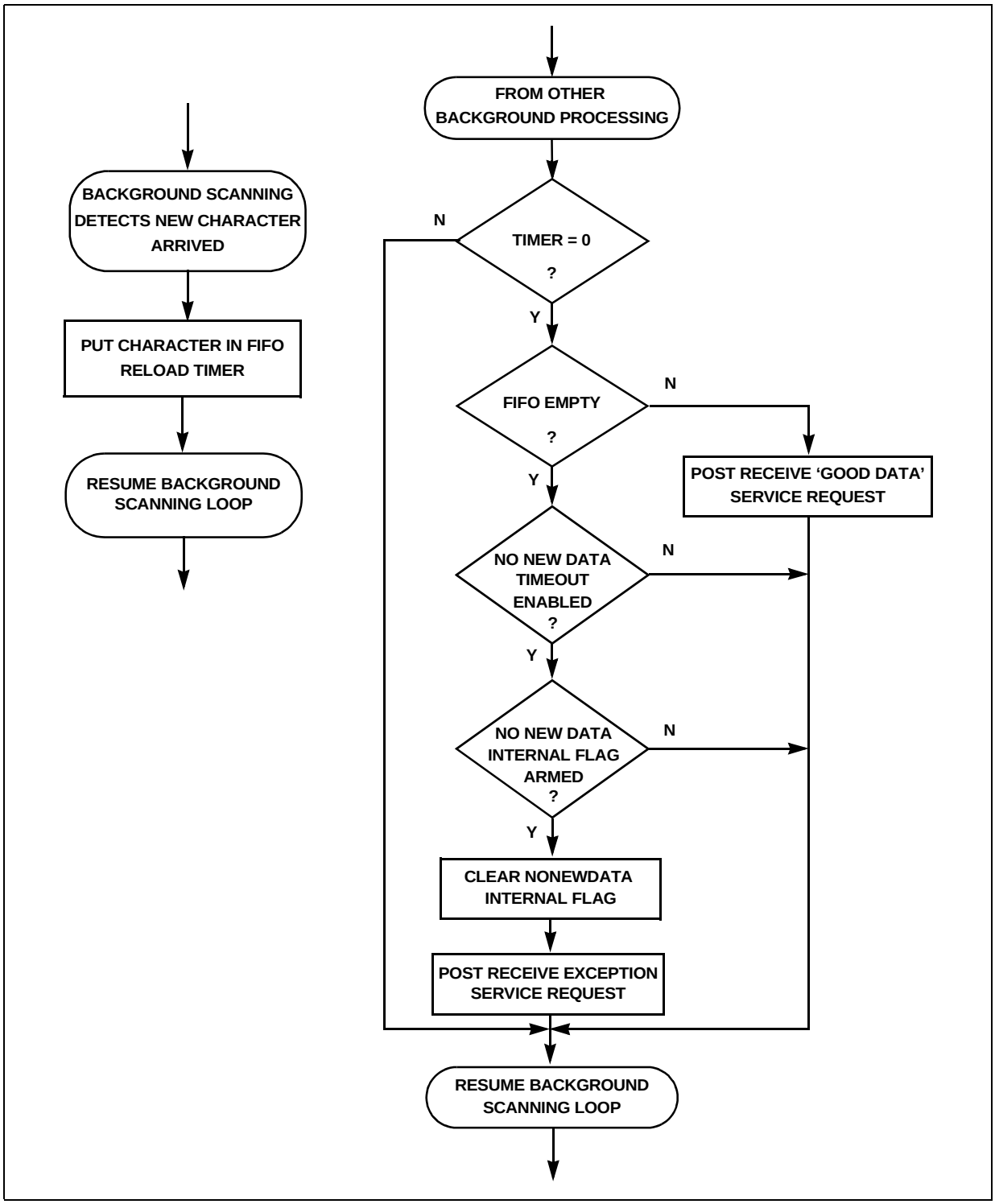
Exception data is sent to the CPU one event at a time. That is, there is a separate service request for each character received with special conditions. If, when an exception condition occurs the receive FIFO contains good data, a good data receive service request is immediately posted upon receipt of the bad data. This happens regardless of the number of characters in the FIFO and the programmed threshold. This allows the CPU to remove the data in the FIFO ahead of the exception data so that the CD1284 can post the service request for the error condition. Once the service-acknowledge procedure for the good data is terminated, a new service request is posted for the exception data.

When the CPU acknowledges the receive exception service request, it first reads the RDSR to determine the status and then to retrieve the data. Reading the data is optional: if the FIFO is not read twice during the service routine, the CD1284 updates the internal FIFO pointers appropriately and discards the second byte.

Note: The CPU need not actually read any data from the FIFO during an exception service acknowledge — the FIFO pointers are correctly updated at the end of the service routine, discarding both the status and the data. In this way, the CPU must at least read the status or it is permanently lost.)

Another special case of exception data handling is received line break conditions. A line break is a character with a start bit, ‘0’ data, and no parity or stop bit. In this case, a null (‘0’) character is placed in the FIFO with the break condition indicated in the accompanying status, and a receive exception service request is posted. However, regardless of the length of the break, only one character is placed in the FIFO. Resumption of normal character reception causes new data to again be placed in the FIFO.

Figure 8. FIFO Timer Processing



5.5.4 Transmitter Operation

Each of the two serial channels on the CD1284 are capable of transmitting characters with a number of programmable characteristics such as length, parity, and baud rate. The channels operate independently and the settings in one have no effect on the operation of the other.

After being reset from either hardware (RESET* input pin) or software (by the master reset command in the CCR), all transmitters are disabled with the TxD output held at a logic ‘1’ condition. This is the ‘off’ or ‘mark’ condition of the asynchronous protocol.

Before any operation of the transmitter can begin, the CPU must program the appropriate parameters in the CORs, TCOR, and TBPR. Once these registers are set, the channel is enabled by issuing a transmit enable command through the CCR, and enabling service requests by setting the appropriate transmit enable request bits in the SRER.

The channel then immediately posts a transmit service request since its FIFO is empty. The CPU responds to the request by loading up to 12 characters into the transmit FIFO through the TDR after it places the CD1284 in the Service-Request Acknowledge mode (see description of service-request/service-acknowledge procedures in Section 5.2.3).

The transmitter does not begin transmitting the characters until the CPU terminates the service routine and writes the EOSRR. Transmission begins by sending a start bit (a logic ‘0’) followed by five to eight data bits (depending on the programmed value), least-significant bit first. The last data bit is followed by the appropriate parity bit, if enabled, and a minimum of one stop bit.

All bit transmission is handled by the transmit bit engine with the MPU sending each bit as requested. If there are still characters in the FIFO, the next one is transmitted immediately after the last stop bit of the previous character. This process continues until all characters in the FIFO are transmitted. At that time the CD1284 posts a service request for more data.

There are actually 14 transmit character holding locations for each channel: 12 in the FIFO, one in the Transmitter Holding register, and one in the Transmitter Shift register. The CD1284 can be programmed on a per-channel basis to request transmit data when one of two conditions exist:

1. When the last character in the FIFO is transferred to the holding register, or
2. When the last data bit of the last character is shifted out of the Transmitter Shift register.

Option number one allows the CPU two character transmit times to reload the FIFO and prevent a transmit data underrun. This is the normal mode of operation. Option number two ensures that the transmitter is empty before reconfiguring the channel. It is likely that transmitter underrun occurs if option number two is selected, unless the CPU is sufficiently fast to respond to a transmit service request and reload the FIFO during transmission of the stop bit(s) of the last character.

If the transmitter underruns, it continues to send stop bits (mark) until more data is placed in the FIFO. Normally, when a string of characters greater than 12 is being transmitted, the software programs the CD1284 transmitter to post a service request when the FIFO is empty. When the last of the data to send is placed in the FIFO, the service request enable is changed so that requests are made after the last character is sent. This notifies the CPU that all the data was transmitted before disabling a channel.

If a channel is disabled without first being emptied, any characters other than the one currently being transmitted are held and the transmitter enters the marking state. If the channel is subsequently reenabled, any remaining data is transmitted.

The transmitter is capable of performing several special functions such as break generation, inter-character delays, and automatic flow control. These functions are discussed in Section 5.6, Section 5.7, and Section 5.8.

As with the receiver, the transmitter has a timer associated with it. This timer generates the timing for embedded transmit commands that send line breaks and inter-character delays. Whenever the MPU detects an embedded transmit command specifying the delay command, this timer is loaded with the value contained in the parameter byte. Then the timer is decremented on each tick of the PPR (prescaler timer) until it reaches zero. At that time, the delay terminates unless the next character in the FIFO is the beginning of another delay command sequence.

5.6 Flow Control

In all data communications applications, data is sent from one system to another by a protocol. Most systems have a method of buffering data for transmission and reception.

In asynchronous protocol, there is no way at the protocol level to determine the length of a data transmission. Therefore, it is not normally possible to designate a buffer area to handle the entire length of the transmission. Also, the hardware receiving the data generally has a limited amount of buffer area — usually a FIFO — and, if the CPU does not unload data fast enough, the buffer or FIFO can overflow. For these reasons, two methods are provided to stop the remote from sending data until there is space to receive data. This is known as flow control.

Flow control can be in-band or out-of-band. In-band flow control uses special characters that can be sent to the CPU to stop data transmission. Out-of-band flow control are signals outside the serial data channel that perform the same function: the RTS* (Request To Send) and CTS* (Clear To Send) signal set, and the DSR* (Data Set Ready) and DTR* (Data Terminal Ready) signals.

The CD1284 supports manual flow control and has built-in capabilities for automatic and/or semi-automatic (depending on direction and options) implementation without CPU intervention.

5.6.1 In-Band Flow Control

In-band flow control is implemented by special characters imbedded in the serial data stream; one to request that transmission stop and one to request that data transmission resume. Any character can be selected, although conventionally, the XON or DC1 (x’11) and XOFF or DC3 (x’13) characters are selected if the ASCII character set is being used.

XOFF designates the character used to stop data transmission. XON determines the character used to resume transmission. Whether these characters are used, the CD1284 allows the two characters to be set to any value appropriate to the system design by the value programmed in SCHR1 and SCHR2 (Special Character register 1 and 2).

SCHR1 defines the XON character and SCHR2 defines the XOFF character. These registers must be initialized by the CPU; the default value loaded during device reset is ‘x’00’.

5.6.2 Receiver In-Band Flow Control

When the CPU senses that the sender requires flow-control due to the receive buffer filling too fast to service, it can request the remote stop transmission by the transmitter sending an XOFF character. This is accomplished by issuing a send special character 2 command through the CCR. The CD1284 then transmits the character programmed in SCHR2.

As previously discussed, the send special character command is preemptive to data currently in the transmit FIFO. The XOFF character is transmitted immediately after the current character and the character in the Transmitter Holding register are sent (a maximum delay of two character times). When the CPU is again ready to start receiving characters, the XON character is sent by another send special character command. At this time, the CD1284 is issued the command to send the character programmed in SCHR1.

Send special character commands override any flow-control by a remote of the CD1284. For example, even if the CD1284 transmitter is shut off by the remote, it can still send flow control characters.

The current state of the flow-control condition is always made available to the CPU through the CCSR. In addition to the enabled/disabled status of the receiver and transmitter, the CCSR displays the flow-control status.

Two bits in the CCSR pertain to receiver flow control, RxFloff and RxFlon. Whenever the CPU issues the send special character 2 (send XOFF) command, the CD1284 sets the RxFloff bit, indicating a request for the remote to stop transmission.

When the CPU issues the send special character 1 (send XON) command, RxFlon is set and RxFloff reset. RxFlon remains set until the first character is received after XON is transmitted. Table 15 shows the bit encoding for RxFloff and RxFlon.

Table 15. CCSR[6:5] Encoding

RxFloff	RxFlon	Encoded Status
0	0	Transmission resumes, the receiver is enabled/disabled, or receiver is in the default reset state.
0	1	XON is sent, but transmission has not restarted.
1	0	XOFF was sent.
1	1	Not used.

RxFloff and RxFlon are cleared whenever the receiver is disabled or enabled, regardless of the state of flow control when the disable/enable occurred.

Note: Regardless of the current state of RxFloff, the CD1284 continues to receive characters. If the remote ignores or is slow to respond to the XOFF character, there an overrun condition can occur.

5.6.2.1 Transmitter In-Band Flow Control

The CD1284 can automatically flow control its own transmitter when it receives the XON and XOFF characters, as programmed in SCHR1 and SCHR2. There are control bits in COR2 and COR3 to enable or disable various aspects of automatic flow control.

Special-character detection must be enabled through the SCD12 bit (COR3[4]) for flow-control characters to be acted upon. When SCD12 is set, the CD1284 scans received characters for a match with one of the special characters programmed in SCHR1–SCHR2.

If enabled in SCD12 and a character matching the contents of SCHR2 is received (the XOFF character), the CD1284 checks that automatic transmit in-band flow control is enabled in COR2[6]. If this function is enabled, the CD1284 stops transmission after the current transmitting character and the character in the Transmitter Holding register, if any, are sent. If enabled, the CD1284 also attempts to match against errored characters. This function is enabled by the CMOE bit (COR5[5]).

COR2[7] enables IXM (Implied XON mode), which determines the character that restarts transmission after a stop by automatic flow control. If IXM (COR2[7]) is '0', only a programmed XON character (SCHR1) can restart the transmitter; all other characters are received and placed in the FIFO. If IXM is reset, any character received restarts data transmission. TxIBE (COR2[6]) must be set to active automatic flow control, otherwise IXM (COR2[7]) has no effect.

As with receiver flow control, the CPU can determine the current state of the transmitter through TXFloff and TxFlon (CCSR[2:1]). When automatic in-band flow control is enabled and the CD1284 receives an XOFF character, TxFloff is set. When an XON character is received, TxFlon is set. Once transmission resumes, TxFlon is cleared. The encoding for TxFloff and TxFlon is shown in Table 16.

Table 16. CCSR[2:1] Encoding

TxFloff	TxFlon	Encoded Status
0	0	Transmission resumes, transmitter is enabled/disabled, or the transmitter is in the default reset state.
0	1	XON was received, but transmission has not restarted.
1	0	XOFF was received, transmission has stopped.
1	1	Not used.

TxFloff and TxFlon are cleared whenever the transmitter is disabled or enabled, regardless of the state of flow control when the disable/enable occurred. This feature can force transmission to resume regardless of remote-initiated flow control.

One final aspect of automatic in-band flow control is FCT (Flow Control Transparency). FCT is enabled/disabled in COR3[5] and determines if remote-initiated flow control is transparent to the CPU. If FCT is not set, in addition to stopping transmission when an XOFF character is received, the CD1284 places the received XOFF character in the receive FIFO and informs the CPU with a receive exception service request. When the XON character is received, it is also sent to the CPU by an exception service request, then restarts data transmission.

If FCT is enabled, received flow control characters control transmission, but are discarded instead of being placed in the FIFO. If the CPU does not require to know when its transmit data has been stopped, this bit can be set to reduce the number of service requests that must be handled.

Table 17 summarizes the control bits in the CORs that enable the various modes of in-band flow control.

Table 17. COR Control Bits

Bit Name	Register	Function
SCD12	COR3	Enables recognition of special characters 1 and 2
FCT	COR3	Enables transparent flow control
TxIBE	COR2	Enables automatic transmitter in-band flow control
IXM	COR2	Enables implied XON mode

5.6.3 Out-of-Band Flow Control

Flow control can also be accomplished through the modem handshake signal pairs RTS/CTS and DSR/DTR. These are called out-of-band because they are external to the data channel. The CD1284 can be programmed to automatically respond to and generate out-of-band flow control through these signals.

5.6.3.1 Receiver Out-of-Band Flow Control

Along with the receiver FIFO threshold that sets the level where the CD1284 posts a service request, another threshold can be set to determine when it automatically asserts/deasserts DTR*. This is the DTR threshold and is enabled in the DTRth[3:0] bits (MCOR1[3:0]). The level can be set for any number of characters from 0 to 12. A threshold of zero disables the function and DTR* is not controlled by the device. If the function and the receiver are enabled, the CD1284 automatically asserts the DTR* output whenever the number of characters in the receive FIFO is less than the programmed number. Once the level reaches the threshold, DTR* is deasserted. DTR* is held in the deasserted state until the CPU removes enough characters from the FIFO to lower the level below the threshold.

For the receiver to operate properly, the DTR threshold must be set to a value equal to, or higher than the receiver service-request threshold. If the levels were reversed, normal character reception could not be completed because DTR* would always be deasserted before the receive FIFO threshold is reached. The CPU would then not get a receive data service request until the receive FIFO timeout is reached. This would result in a serial data transmission performance limitation.

The DTR* output can also be manually controlled through MSVR2[1]. Setting this bit to ‘1’ asserts the DTR* output.

5.6.3.2 Transmitter Out-of-Band Flow Control

Transmitter out-of-band flow control is implemented with three modem control signals: the RTS* output and the CTS* and DSR* inputs. The RTS* output can be programmed to be automatically asserted whenever there is data in the transmit FIFO and the transmitter is cleared to send. CTS* and DSR* can be enabled to automatically control the transmitter.

RTS Automatic Output is enabled in the RtsAO bit (COR2[2]). If RtsAO is set, the CD1284 automatically asserts the RTS* output when there is data in the FIFO to send. When the data is sent and the FIFO is empty, RTS* is deasserted until the CPU places more data in the FIFO. If RTSAO is not set and if required by the remote, the CPU must manually control the RTS* output through MSVR1[0].

The CTS* input can also be monitored by the CD1284 and is a transmitter enable. The functions is enabled by setting CtsAE (COR2[1]). If CtsAE is set, character transmission occurs only when the CTS* input signal is asserted. If the signal is deasserted during active transmission, the current character plus the character in the Transmitter Holding register are transmitted and transmission ceases. Thus, a minimum of one and a maximum of two characters can be transmitted after the control signal is deasserted. Transmission resumes when the signal(s) is reasserted.

The send special character command does not sample the CTS* or DSR* inputs. If the CPU opts to send one of the special characters, the character is transmitted regardless of the state of these inputs. This is preferable as the CPU can still flow control a remote even if it is being flow controlled. If the state of CTS* and DSR* are important, they should be tested through MSVR1[7:6] before the special character send command is issued.

5.6.4 Modem Signals and General-Purpose I/O

Each channel of the CD1284 has four pins that can be used either as modem-control or general-purpose input/output pins. The modem signal names assigned to these four pins provide an easy reference for system designers. In fact, they are all simply general-purpose inputs and outputs (if automatic out-of-band flow-control is not used) are individually controlled in the MSVRs. Since they are general-purpose, system designers can opt to connect the pins any way to suit the application.

DCE, DTE Application

When the system software design opts to use automatic out-of-band flow control, then the signal naming convention no longer holds true in some cases, depending on if the device is used as DCE or DTE. For this case, use these pins within the CD1284, connect them accordingly, and disregard their names. The RTS* and CTS* pins are associated with the transmitter; the DTR* and DSR* pins are associated with the receiver. Table 18 shows the Intel recommended signal hook-up for automatic out-of-band flow control.

Table 18. Out-of-Band Pin Connections

DCE	DTE	CD1284 Pins	Out-of-Band Flow Control
CTS*	–	DTR*	Signal remote to transmit
RTS*	–	–	Not implemented in this direction
–	RTS*	RTS*	Request remote permission to transmit
–	CTS*	CTS*	Enable transmitter

For example, if the CD1284 is designed for DCE and automatic out-of-band flow control, connect DTR* to the remote CTS* input. If the CD1284 is for the DTE side, then connect the CD1284 CTS* output to the remote CTS* input.

Note, if automatic out-of-band flow control is implemented, the activity of DTR* and DSR* do not implement the function assigned to those signal names by the signaling conventions of the CCITT (and other) standards organization. These pin names only apply to these pins if they are under program control and not under automatic CD1284 control. In fact, the defined DTR function enables the modem to go on- and off-line, depending on the state of the pin. If automatic flow

control is used, then DTR* goes inactive when the receive FIFO reaches the programmed threshold, causing the modem to drop the connection (carrier) to the remote, — this is not the correct use of this function.

Table 19. Modem Control Pin Functions

Modem Control Pins	Function
RTS*	Request to send (general-purpose output)
CTS*	Clear to send (general-purpose input)
DTR*	Data terminal ready (carrier detect/general-purpose input/output)
DSR*	Data set ready (general-purpose input)
CD*	Carrier detect (general-purpose input)
RI	Ring indicator (general-purpose input)

Modem pins are implemented as I/O ports accessible by either the CD1284 internal microcode or the host. The modem pins are not connected directly to the transmit or receive hardware. When a user programs the out-of-band modem functions to be active, the CD1284 microcode reads from and writes to these pins. Specifically, when RTS* and CTS* are used for transmit flow control, the CD1284 microcode asserts RTS* and senses CTS*, as required (Table 19). Also, when the receive FIFO is full, DTR* is negated. The host must not reassert DTR* inadvertently.

Note: The host is not ‘locked out’ of accessing these bits; ensure that these bits are not written to when auto out-of-band flow control is enabled as it could cause a system malfunction.

The user can directly control RTS* and DTR* and can probe the state of the CTS*, CD*, and DSR* inputs through the MSVR. Since the host is accessing these pins directly, there is no delay in its ability to detect a level change.

The CD1284 can be programmed to detect level changes and generate service requests when level changes occur. It does this in firmware by reading DTR* and CD* and comparing them to a previously stored value. This function is performed in the main timing loop of the firmware; the maximum time required to detect a level change in worst-case conditions is approximately 2 ms.

When the CD1284 is performing this function, the modem pins are periodically sampled rather than continuously monitored. In this way they have minimal sensitivity to noise, a desirable feature in data communication applications. However, in extremely noisy applications, reread a modem line that caused a modem signal change service request to verify it has changed and is not malfunctioning. This eliminates even the slightest possibility of a noise pulse causing erratic operation.

When the CD1284 is monitoring modem pins to control transmit or receive functions, it does not rely on the previously stored value, but instead checks the pins at the appropriate time. Thus, there is very little delay in this response. For example, before deciding to transmit another character, it examines the CTS* pin at that time. The CD1284 makes this decision when moving characters from the FIFO to the Holding register, not from the Holding register to the Shift register.

Note that the logical sense of the modem bits is inverted; that is, a write of ‘1’ to MSVR1 or MSVR2 causes the output pin to go to nominal zero volts. Likewise, a low-voltage input is sensed as ‘1’.

5.6.4.1 Generating Service Requests with Modem Pins

The CD1284 can generate service requests when any one of the input pins changes state. Either or both edges can be detected by setting bits in MCOR1 and MCOR2. For each pin, the user can individually enable an on-to-off or off-to-on transition detection of the inputs. When the CD1284 detects such a transition, the corresponding bit in the MCR is set. If the corresponding IER bit in the channel is set, the CD1284 asserts the SVCREQM* output.

The user must clear the MCR during the service request service routine before writing to the EOIR. The CD1284 performs this task by reading the modem input signals and comparing the current value with the value read in the last pass through the outer scanning loop. Because this is the lowest-priority event in the CD1284 scanning loop, changes can not be detected unless they are several hundred microseconds long.

For example, the modem input pins can be used to detect the closing of a switch. However, consider the relatively slow speed of response when using modem input pins for this purpose. The CD1284 does not latch the modem input signals.

5.6.4.2 Using Modem Pins as General-Purpose I/O

Since the modem pins can be directly accessed by the host, they can be used as general-purpose I/O pins if they are not needed for flow control or modem interfacing. Simply read from and write to these pins as with any I/O port.

5.7 Receive Special Character Processing

The CD1284 has several ways to send special characters and to process these characters when received. Some special characters have fixed definitions and others are user-defined. Figure 9 on page 64 defines the processing that the CD1284 performs for receive data. This flow chart illustrates the special character handling process.

5.7.1 UNIX® Character Processing

The CD1284 incorporates special character processing of particular benefit in systems designed to run the UNIX® operating system. The processing performs some of the functions normally handled by the ‘line discipline’ part of a serial device driver program. This provides higher overall performance in serial communication than could otherwise be obtained because character manipulation occurs at the hardware level without any CPU interaction. This processing includes CR (carriage return) and NL (new line) substitution, programmable response to errored characters (framing, parity and overrun errors), the LNext function and ISTRIP. Each type of processing is optional and can be enabled/disabled with control bits in the CORs 2, 4, and 5. The following sections describe of each of these functions.

5.7.1.1 Line-Terminating Characters

The CD1284 can be programmed to perform automatic substitution of the CR and NL characters on both received and transmitted data. Received character processing has five unique substitutions based on the value of IGNCR, ICRNL, and INLCR (COR4[7:5]); some combinations cause identical actions.

000	Do nothing – function not enabled
001	Received NL changed to CR
010	Received CR changed to NL
011	Received CR change to NL; NL changed to CR
100	Received CR discarded
101	Received CR discarded; NL changed to CR
110	Received CR discarded
111	Received CR discarded; NL changed to CR

5.7.1.2 Errored Character Processing

The CD1284 can easily manage received characters with errors (such as, parity, framing, and overrun). If none of the special processing functions are enabled, errored characters are delivered to the CPU through a receive exception service request. As defined by the PEH[2:0] bits (COR4[2:0]), these characters can be handled in one of the following ways:

- Parity errors can be ignored — the character is placed in the FIFO as good data and is given to the CPU as any other received good data.
- An errored character can be replaced with a NULL (x'00) character in the FIFO.
- An errored character can be replaced in the FIFO with the 3-byte string x'FF–NULL-character. If this mode is enabled and an actual good x'FF character is received, it is replaced in the FIFO with the two character sequence x'FF-x'FF.
- An errored character can be discarded.

Received breaks are handled differently from other errored characters. They can be processed, based on the settings of the IGNBRK and -BRKINT bits (COR4[4:3]), as:

- Reported as an errored character by a received exception service request.
- Replaced with a good NULL (x'00) character in the FIFO.
- Discarded.

5.7.1.3 LNext

LNext (Literal Next) allows 'escaping' or ignoring any special meaning of special characters and considers them as normal data. The escape character is defined by the value in the LNC register. If the CD1284 receives this character, places it and the next character in the FIFO without further processing. As an example, this allows a flow-control character to be received without it causing actual flow-control activity. LNext can be enabled to operate on characters received with errors (such as, parity, framing, and overrun), otherwise errored characters are handled normally and the next character is not escaped.

5.7.1.4 ISTRIP

ISTRIP is a simple function that, if enabled, resets the most-significant bit (bit 7) of all received good characters. If the character has a parity or framing error, the ISTRIP function does nothing and the character is sent to the CPU as a normal receive exception service request.

5.7.2 Non-UNIX® Receive Special Character Processing

In addition to UNIX special-character processing, the CD1284 provides other special character recognition capabilities. The CD1284 has four registers that define special characters, SCHR1–SCHR4. SCHR1 and SCHR2, are used in flow-control activities and (see Section 5.6). SCHR3 and SCHR4 define two additional special characters that the CD1284 can scan for in the receive data stream. Recognition of special characters 3 and 4 are enabled by the SCD34 bit (COR3[6]). If either of these characters are received, a special character detect (receive exception) service request is sent. Note that if automatic in-band flow control is not enabled, SCHR1 and SCHR2 can still be used as special characters. They are detected and reported as receive exceptions, but they do not cause flow-control activities to be envoked.

The range detect function is another special character function. If this mode is enabled (COR3[7] set), the CD1284 compares all received characters against the values in the SCRL and SCRH registers. If the character received falls between these two values (inclusive), a special character detect service request is posted.

The status shown in the RDSR indicates which of the special character recognition conditions were met and caused the receive exception service request.

Figure 9. CD1284 Receive Character Processing

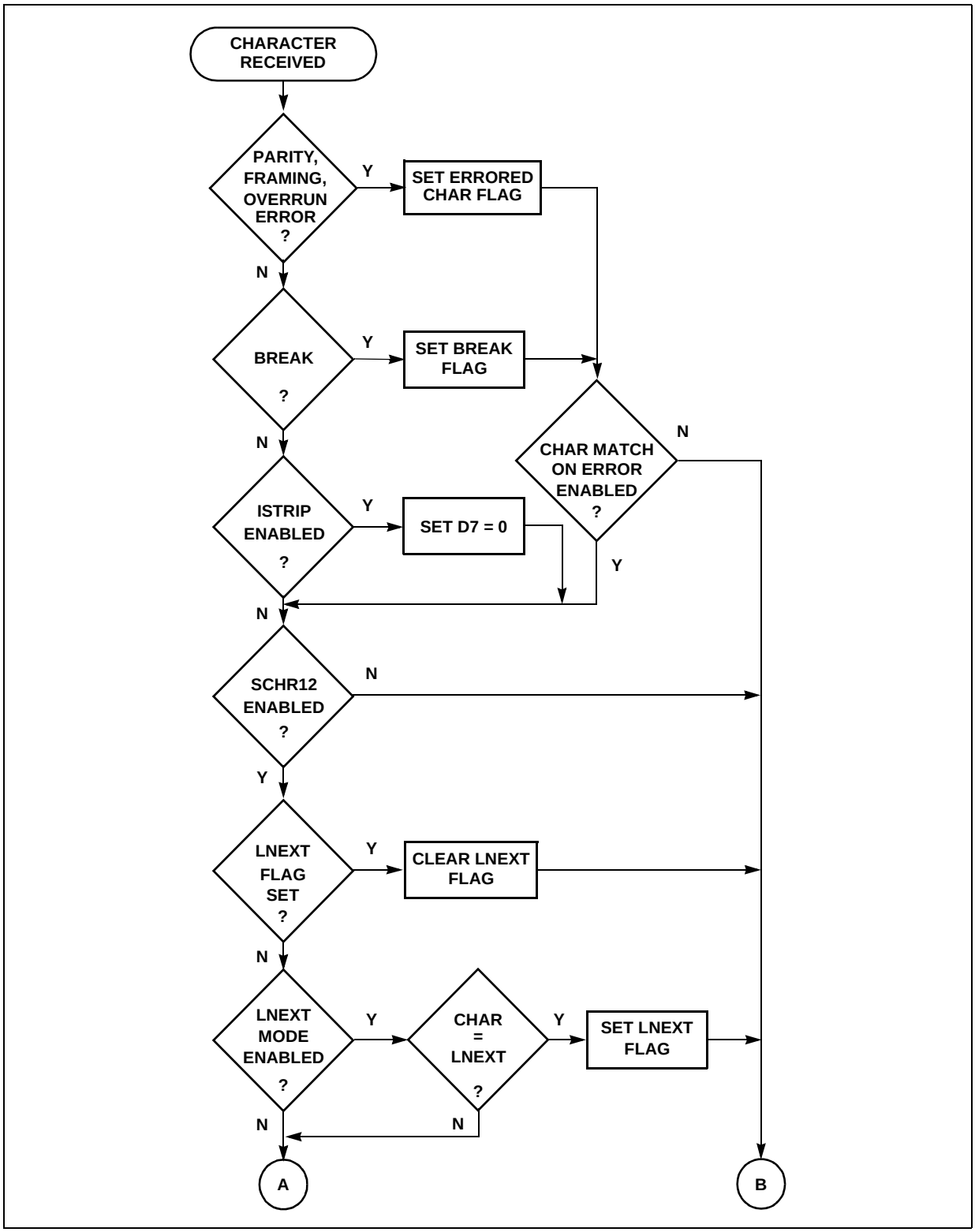


Figure 9. CD1284 Receive Character Processing (Continued)

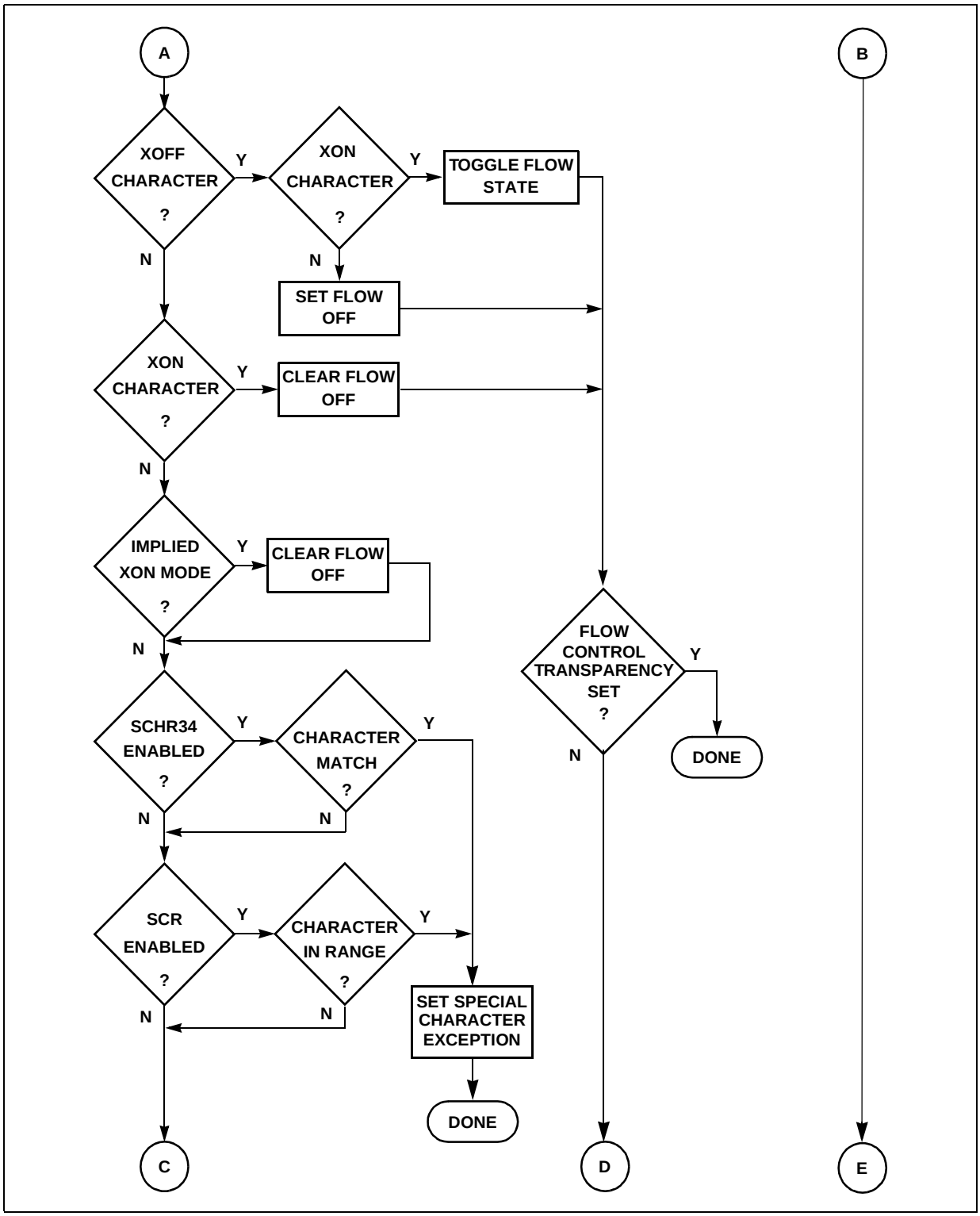
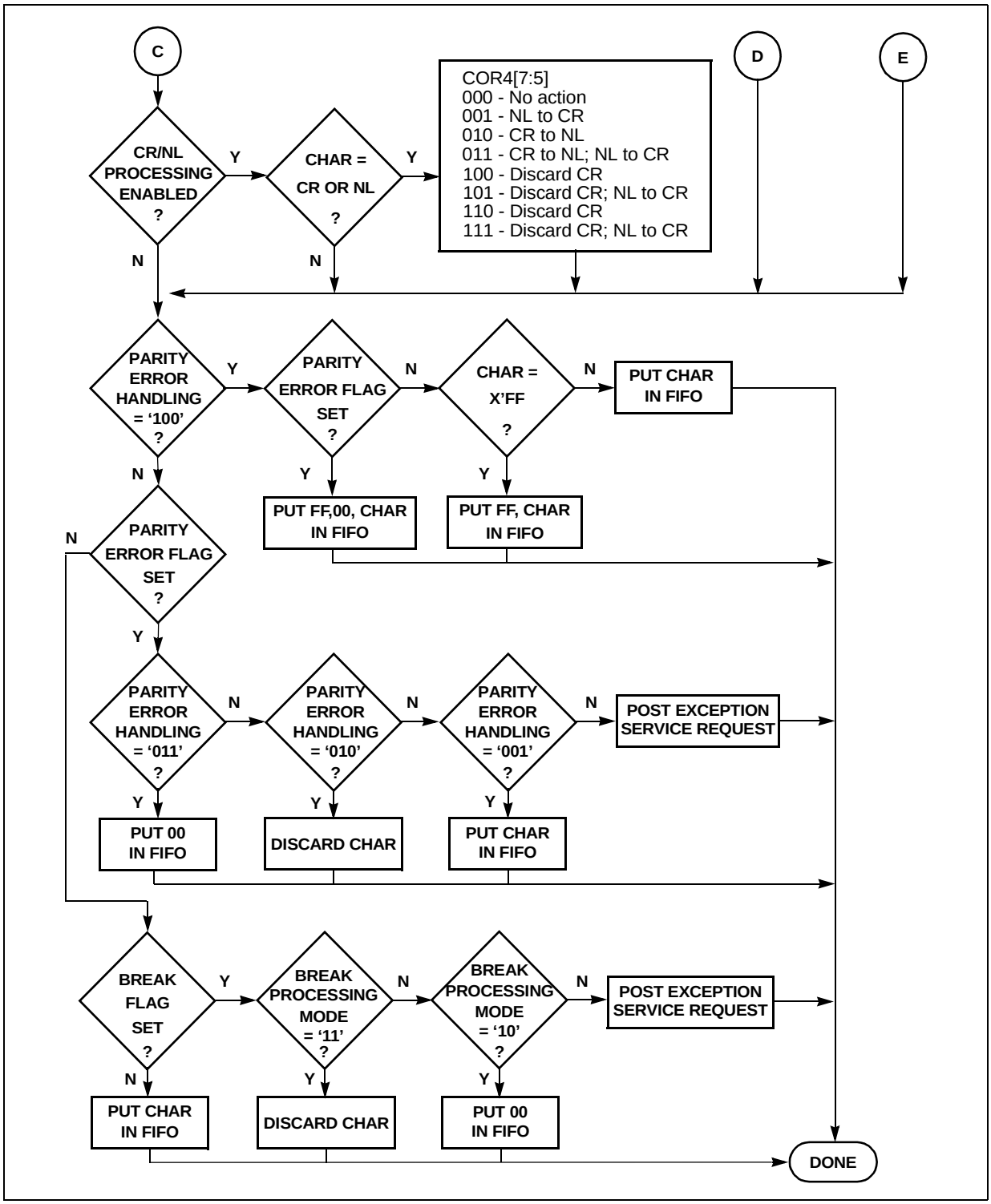


Figure 9. CD1284 Receive Character Processing (Continued)



5.8 Transmit Special Character Processing

The CD1284 also provides some special character handling on the transmit side – embedded transmit commands and direct commands to transmit predefined special characters. Figure 10 on page 70 illustrates the process of special character handling.

5.8.1 Line Terminating Characters

On transmit, there are four possible substitutions based on the setting of two flags, the ONLCR and OCRNL bits (COR5[1:0]):

00	Do nothing — function not enabled
01	Change all <CR> characters to <NL>
10	Change all <NL> characters to <CR> <NL>
11	CR characters changed to NL or NL

When both flags are set (‘11’), only one translation occurs – a CR that changed to NL is not changed to CRNL.

5.8.2 Embedded Transmit Commands

The CD1284 has a special feature that optionally allows specific ‘escape’ character sequences in the transmit data stream to be interpreted as commands. These are called ETCs (embedded transmit commands) and are enabled in COR2[5]. These sequences can insert programmed time delays between characters and generate a line break on the transmit data output.

If enabled, an ETC is detected when the two- or three-character escape sequence is detected in the transmit FIFO. An escape-character sequence is comprised of the special escape character followed by the command character and an optional count for the delay period. The escape character is an all-zero character (null or NUL in the ASCII character set map). Five commands are supported in the ETC command set:

- NUL NUL
- NUL x’81
- NUL x’82 x’xx
- NUL x’83
- NUL x’01–x’3F

NUL NUL – Send One NUL Character

This command sequence allows the NUL character to be sent alone. Thus, this ‘escapes’ the escape when it is desired to send a null character.

NUL x’81 – Send BREAK

This sequence forces the transmitter to enter the line-break condition for at least one character time. Several conditions control the continuation and/or termination of the line break.

- If there is no more data in the FIFO following the send break command, the break continues indefinitely until terminated by a stop break command.
- If there is an insert delay command (see the next command) immediately following the send break command, the break duration is set by the value programmed in the delay command. Any other character in the FIFO immediately following the send break command carries an ‘implied’ end-of-break condition, causing the break to be terminated and the next character to be sent.

NUL x’82 x’xx – Insert Delay

This command causes a delay between the previous character transmitted and the next character to be transmitted. The hex value contained in the third byte of the sequence determines the time of the delay based on the basic time period set by the PPR. The value is treated as an unsigned binary value loaded into an internal counter. The counter decrements once for each tick of the prescale period timer. Thus, if the PPR sets a basic timing period of 10 ms and the value set by the command is 100 (x’64), then a delay of 1 second is generated. Multiple insert delay commands can be placed in the FIFO if time delays longer than that generated by a single delay period are needed.

This command is useful when a delay is required after sending a carriage return. A printer is an example of this type of situation. Often, the carriage return causes the printer to start a print cycle and the sending device must wait for the print to complete before sending the next line of text (unbuffered input). Using the insert delay command allows the delay to be performed automatically without the need for the CPU to time it. The delay command is placed in the FIFO directly following the carriage return and preceding the first data for the next line. The CD1284 automatically executes the delay following the carriage return, then restarts sending characters.

Another useful application of the delay command is as a built-in timer that the CPU uses as an interrupt source causing it to periodically check its internal buffers for data to transmit. This assumes that the channel is not currently transmitting data. When the CPU services the transmit FIFO service request after a delay timeout (as set by the delay value) it can start transmission of a buffer if data is available or resend the insert delay command and wait for the next service request. An internal timer interrupt set by the CPU is now unnecessary to perform this function.

NUL x’83 – Stop BREAK

This command terminates a break in progress regardless of other conditions. This command can be preceded by insert delay commands to set a specific, programmed break period if more than one character time is required. Any character in the FIFO causes the break to terminate. NUL x’83 is required only if it is necessary to stop the break and there is no more data to be sent. A break continues until another character is sent or ESC x’83 is encountered in the FIFO.

NUL x’01–x’3F – Send Repeat Space

This command causes the CD1284 to send repeated space characters. The character following NUL is interpreted as a binary count specifying the number of ASCII space (x’20) characters to send. The count must be in the range of x’01 through x’3F (1–63 decimal).

5.8.3 Send Special Character Command

One command of the CD1284 transmits any one of the four special characters programmed in SCHR1–SCHR4. The command is issued by the CCR[5] set to ‘1’, and the least-significant three bits encoding a selection of one of the four characters. This function is preemptive, meaning that

the selected character is transmitted immediately following the currently transmitting character and any character in the Transmitter Holding register. This preempts any characters in the transmit FIFO. If there are characters in the transmit FIFO, transmission resumes after the special character is sent.

One important use of this command is that it allows the CPU to flow-control a remote without having to wait for the transmit FIFO to empty before the flow control character is placed in it. This is a special case different from the normal transmitter operation of the CD1284, in that the character can be sent without waiting for a transmit service request. The only requirement is that the transmitter *must* be enabled (interrupts need not be enabled).

Figure 10. CD1284 Transmit Character Processing

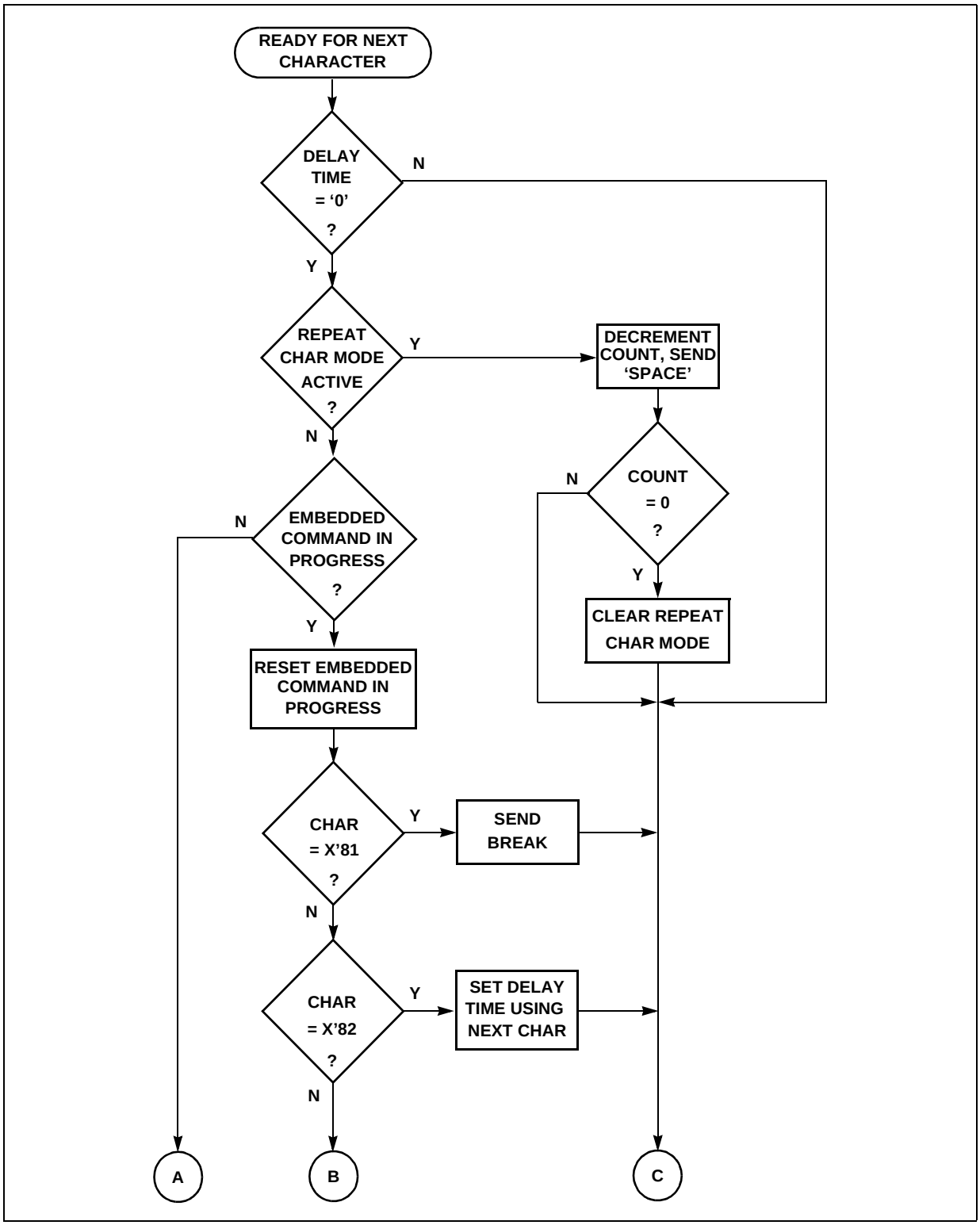


Figure 10. CD1284 Transmit Character Processing (Continued)

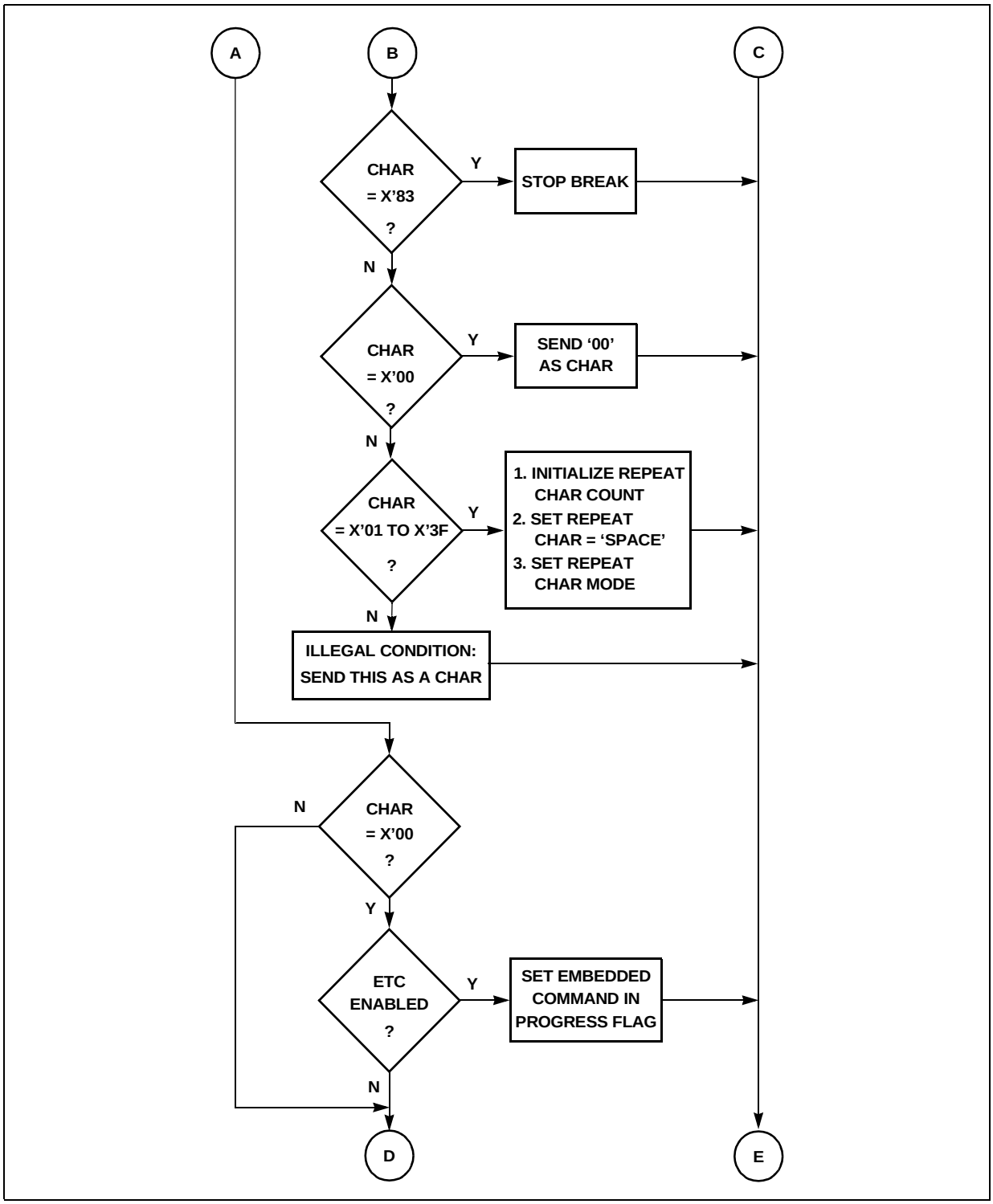
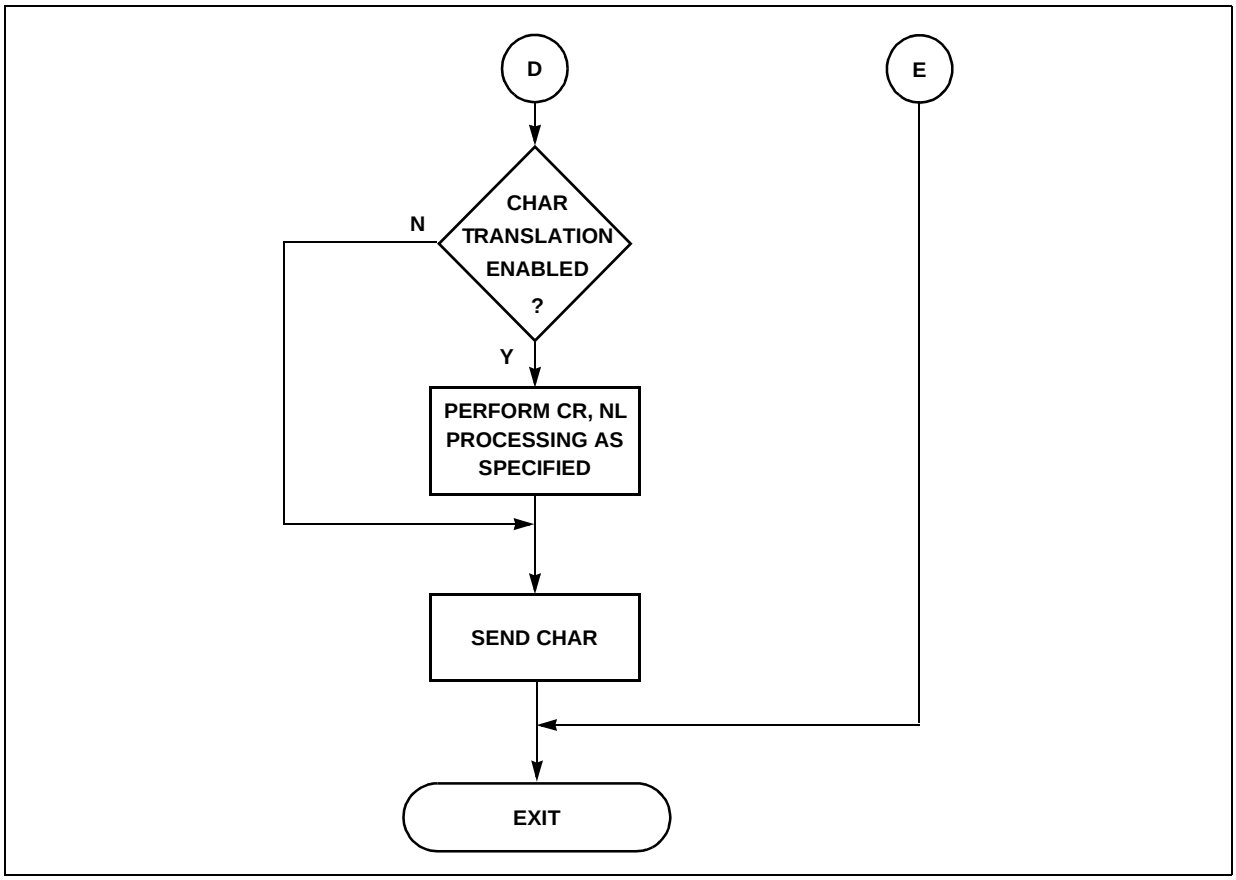


Figure 10. CD1284 Transmit Character Processing (Continued)



5.9 Baud Rate Generation

The CD1284 provides a separate baud rate generator both directions of each channel. Each receive and transmit baud rate generator can be driven from one of five available clock sources. The source being used is selected by the value in the R/TCOR. The selected clock is divided by the value in the R/TBPR to yield the desired bit rate.

Note: R/T is used as a register abbreviation indicating Receive / Transmit followed by the register acronym.

The five clock sources are:

Clk0	System clock ÷ 8, R/TCOR = 0
Clk1	System clock ÷ 32, R/TCOR = 1
Clk2	System clock ÷ 128, R/TCOR = 2
Clk3	System clock ÷ 512, R/TCOR = 3
Clk4	System clock ÷ 2048,

The system clock is the external clock driving the CLK input of the CD1284. Three example baud rate tables are provided at the end of Section 6.7 on page . A sample program for automatically deriving the baud rate clock selection and divisor is also provided in Chapter 6.0.

5.10 Serial Diagnostic Facilities — Loopback

The CD1284 provides the capability to perform internal loopback testing for both local and remote loopback modes. Loopback mode is enabled by the LLM (Local Loopback mode) and RLM (Remote Loopback mode) bits (COR2[4:3]).

In Local Loopback mode, the output of the transmitter bit engine is directly connected to the input of the receiver bit engine; the input and output pins (TxD and RxD) are disconnected. The TxD output is left in the mark condition so that remote equipment does not sense any line activity. Input conditions on the RxD are ignored. All channel parameters and service-request functions are in effect and operate normally. If enabled, special characters in the loopback data are detected and acted upon and UNIX translations occur.

Remote Loopback mode causes the CD1284 to echo any received data back immediately to the transmit output. This is done on a character-by-character basis rather than on a bit-by-bit basis. In other words, characters are echoed once they are completely received and assembled. Received data is not placed in the FIFO, thus no data is sent to the CPU. The received character is retransmitted with parity and stop bit options as defined by COR1. Note, if the transmit baud rate is lower than the receive baud rate, overrun errors and loss of data are likely to occur.

5.11 Parallel Port FIFO and Data Pipeline Overview

The parallel port within the CD1284 implements all modes defined for the ‘slave’ (peripheral) side in the *IEEE STD 1284 Standard Signaling Method for a Bidirectional Parallel Peripheral Interface for Personal Computers*. This specification defines four methods of performing bidirectional data transfers between a computer system and a peripheral device, in addition to the generally accepted unidirectional Centronics®-compatible mode. These modes include Compatibility mode, Reverse-Nibble mode, Reverse-Byte mode, ECP (Extended Capabilities Port) with and without RLE (run length encoding), and the EPP (enhanced parallel port).

The IEEE 1284-compliant parallel port consists of two major functional blocks:

- A data pipeline that moves data between the parallel port and the CPU and includes a FIFO, holding registers, DMA control, interrupt control logic.
- A channel control state machine to perform all control and handshake generation on the parallel port interface side of the device.

5.11.1 IEEE STD 1284 Protocols

The following sections discuss data movement within the pipeline for the various IEEE STD 1284 operating modes. For a complete description of these modes, refer to the IEEE STD 1284 specification; it is beyond the scope of this data book to give complete information on the specification. A copy of the IEEE STD 1284 standard can be obtained from:

IEEE Standards Department

445 Hoes Lane

P.O. Box 1331
Pascataway, NJ 08855-1331
USA

5.11.2 Bus Interface

DMA transfers are the preferred means of transferring data to/from the FIFO. However, it is also possible to transfer data to/from the data pipeline by reading and writing the holding registers directly through the PIO. DMA request and acknowledge handshake signals support transfers to/from the 16-bit-wide DMABUF register. The direction of transfer is determined by the DMAdir bit (PFCR[5]).

In the transmit direction, with DMAbufWe (PFCR[0]) set, the CPU can write 2 bytes at a time directly to the DMABUF register. However, most applications are not concerned with speed on the parallel port in the reverse direction and do not require 16-bit writes to the FIFO. The CPU must avoid writing to these registers when they are already full or reading from them if they are empty. The status bits in the HRSR indicate if the holding registers and the DMA buffer are full or empty. When writing a block of data to the CD1284 (DMAbufWe is set to '1'), the CPU can determine how much data the FIFO can accommodate by reading the PFQR.

Should data become 'trapped' in the DMABUF register in the receive direction because of a failure of the external DMA controller or because the external buffer area is full, it can either remain until the DMA transfer can be resumed or the CPU can read the data directly from the DMA buffer.

Note: The DMA buffer can only be read when DMAREQ* is active because data is not moved into the DMABUF register until DMAREQ* is activated by the threshold logic or a timeout condition.

Once a DMA request is initiated by the CD1284, it is maintained until the last data transfer the FIFO can accommodate occurs, or the CPU either clears DMAen or clears the FIFO and data-transfer logic by setting FIFOres. In the transmit direction, the DMA request is removed by the CD1284 when it determines that the FIFO is nearly full. (If RLEen is set, the pipeline does not fully drain into the FIFO, but the logic does not factor that into the decision to conclude the DMA transfer.)

In the receive direction, the DMA request is removed when there are not at least two more bytes available to transfer or a tagged byte has moved into the data pipeline. In the latter case, an interrupt is generated to the CPU (IntEn must be true) to remove the tagged data from the pipeline.

The quantity of data transferred within a single DMA request can significantly exceed the capacity of the FIFO if RLEen is set, the parallel port is in ECP mode, and compressed data is being transferred. This is because the FIFO always stores the data in compressed form. Since other modes do not support RLE compression, the CPU should only set RLEen when the parallel port interface is in ECP mode.

5.11.3 Parallel Port FIFO

The CD1284 has a dedicated 64-byte FIFO with counters to maintain the fill/empty pointer addresses, logic to manage data transfers, automatic DMA handshake, and status interrupts to the CPU. A simple register interface provides control over setting the direction of the pipeline, initializing/resetting the DMA pointers, setting the DMA threshold, and so on. The FIFO management logic responds to data-transfer requests from the dedicated IEEE 1284 parallel port state machine.

Byte-alignment issues on transfers to/from the FIFO are avoided by having the FIFO byte-oriented with 2-byte word packing/unpacking occurring between the DMABUF register and PFHR1 and PFHR2. The order of byte transfers to/from the DMA buffer is controlled by the BYTESWAP input. If BYTESWAP is high, the upper byte (bits 15:8) is transferred first. If BYTESWAP is low, the lower byte (bits 7:0) is transferred first.

Data transfers to/from the CPU are initiated by a DMA request whenever the quantity of data or space in the FIFO equals or exceeds the threshold value stored in the PFTR. The DMA request is deasserted during the DMA cycle determined by the logic to be the last because of filling/emptying the FIFO or the presence of tagged data in the receive pipeline.

5.11.4 Receive Direction

In the receive direction (DMAdir = 0), the first two bytes of data placed into the FIFO by the parallel port are immediately moved into the data pipeline, PFHR1 and PFHR2 (Figure 11 on page 78). This is done in part to make the tagged status of the data visible to the pipeline control logic. If RLEen is ‘0’, any tagged data from the FIFO must move through the pipeline. However, tagged data cannot be transferred to the CPU by a DMA transfer from the DMABUF register. Therefore, the presence of tagged data in the pipeline causes an interrupt to the CPU. The CPU must then examine the HRSR to determine the pipeline status.

If there is tagged data in one of the holding registers, the CPU must read that register to empty it and clear the tag. If more data is available in the FIFO, data immediately moves forward to fill the pipeline. If the FIFO is empty, the pipeline does not move. If the CPU emptied PFHR2 and PFHR1 is full, the data in PFHR1 moves forward to PFHR2 only if the FIFO is *not* empty.

The pipeline logic keeps the pipeline full in the receive direction. The value in the threshold register is tested against the quantity of data in the FIFO. Therefore, a number of characters equal to the PFTR-threshold value plus two must arrive before a DMA request is made to the CPU to remove the data.

5.11.5 Receiving Compressed Data

RLE compressed-data sequences that consist of a tagged RLE count followed by the compressed data character are stored in the FIFO in compressed form. As data moves from the FIFO into the data pipeline, the tag bit is inspected. If the tagged data is an RLE count (HostAck signal is high) and RLEen is true, the RLE count is loaded into the RLCR instead of PFHR1; the next data character is loaded into PFHR1. Decompression occurs by holding the compressed character in PFHR1 as copies of the character are shifted forward into PFHR2. As each copy of the character is shifted, the RLCR value decrements. When the RLCR reaches zero, the hold on PFHR1 is released and it can shift forward in the pipeline as ordinary data.

Tagged data from the FIFO is recognized as an ECP mode address and shifts into the pipeline where it causes an interrupt to the CPU to remove the tagged data from the pipeline. If RLEen is ‘0’, all tagged data from the FIFO is shifted into the pipeline and produces CPU interrupts.

If an immediate termination occurs between the reception of the RLE count and the corresponding data, then the RLE count is stored in RLCR and the next data byte received in ECP mode is uncompressed into the FIFO (based on the values in RLCR and if RLEen is still set). If the next byte received in ECP mode is a new RLE count, then that value overwrites the old value in RLCR.

5.11.6 Stale Data (Stale, OneChar, and Timeout Status Bits)

Data transfer to the CPU can also be initiated by the ‘stale’ data timer. This timer is reloaded with the value in the SDTPR and restarts each time data is placed into the FIFO from the parallel port. When the timer reaches zero, the status indication stale (visible in the PFSR) is set true unless StaleOff (PACR[5]) is true.

StaleOff keeps the stale status false, even though the SDTCR counter value is zero. Should the stale status become true with at least two characters of data available, a DMA request is made to transfer the data. If the stale is true and there is exactly one character available, the OneChar status bit is set (PFSR[1]) and an interrupt is generated to the CPU to transfer the single residual character.

The Parallel FIFO Status register indicates the Stale and OneChar conditions, and FFmpty. The HRSR (Holding Register Status register) shows that holding register PFHR2 contains the final character. An odd number of bytes can not be transferred by DMA. If a DMA transfer completes with 1 byte of data left, the data is held pending arrival of additional data or the expiration of the stale data timer.

The OneChar status is latched true when the FIFO and DMA buffer are empty and there is one character in the pipeline in PFHR2. While the OneChar status is true, further pipeline operations are inhibited. If additional data arrives in the FIFO, it remains there until the CPU:

1. Services the interrupt caused by the OneChar status, and
2. Reads the data character from PFHR2.

When the CPU reads the single character from PFHR2, any newly arrived data in the FIFO immediately moves forward into the pipeline and a DMA transfer can begin if conditions warrant.

Another latched status condition associated with the stale data timer is the Timeout status bit (PFSR[5]). Timeout is reset by the FIFOres bit (PFCR[7]) and the ClrTO bit (PACR[3]). Timeout, OneChar, and DataErr are pipeline interrupt conditions and, if enabled, generate an interrupt. In the receive direction, the Timeout condition is armed when Stale is ‘0’ and ClrTO and FIFOres are also ‘0’. When Stale becomes ‘1,’ the timeout is triggered, but is not set until a DMA transfer is complete, the FIFO is empty, and there is no more than one character remaining in the pipeline. To clear the timeout condition, set the ClrTO bit. To reenable the timeout function, clear the ClrTO bit.

The CPU can arm the timeout by a write of ‘01h’ directly to the SDTCR. If the timer expires before any data arrives, an interrupt is generated for the timeout condition. If data arrives before the timer expires, the interrupt delays until the data becomes stale.

5.11.7 Transmit Direction

Note: In the transmit direction, the pipeline behaves in one of two ways depending on the RLEen control bit. RLEen should *only* be set by the CPU after the parallel port is in ECP mode, otherwise compression of data occurs, but cannot be supported in data transfers on the parallel port. If RLEen is ‘0’, data written to the DMABUF register by a DMA (DMAen true) or CPU write (DMAbufWe true) is moved through PFHR1 to PFHR2 and immediately transferred into the FIFO (if space is available).

If RLEen is ‘1’, run-length encoding is enabled and comparators among the pipeline stages recognize repeated strings of characters and compress them (Figure 12 on page 80). To allow the comparator-based logic to work, the pipeline registers, PFHR1 and PFHR2, must be kept full. One comparator determines if the characters in PFHR1 and PFHR2 are identical.

Another comparator determines if the next character coming from the DMABUF register and the character in PFHR1 are identical. Compression begins when the pipeline is full (immediately after a DMA or CPU write to the DMA buffer) and both comparators show identical characters in their pipeline stages. This starts the compression process and the character in PFHR1 and the DMA buffer shift forward. The (same) character in PFHR2 is not loaded into the FIFO, but rather the RLCR increments to '1'.

As long as identical additional characters are loaded into the DMA buffer, the RLCR value continues to increment and the data in PFHR2 does not move into the FIFO. When the repeated sequence is finally broken, or the RLCR count reaches 127, the RLCR value transfers into the FIFO, the RLCR zeros, and the character in PFHR2 transfers into the FIFO. Compression resumes when both comparators indicate the presence of a string of at least three identical characters. During intervals between DMA transfers, the last two data characters are held in PFHR1 and PFHR2.

After the entire block transfer is complete, the CPU must either zero RLEen or ensure that both DMAen and DMAbufWe are zeros. When either of these conditions is true, the pipeline is released and data held in PFHR1 and PFHR2 transfers into the FIFO.

The timeout interrupt can be a general timer interrupt in the transmit direction. Unlike the receive case, when DMAdir is true, the timeout status is immediately set when the timeout is triggered by a '0'-to-'1' transition of Stale. To use the timeout interrupt, the CPU must load the desired time delay directly into the SDTCR. When the timer expires, Stale becomes true and the timeout interrupt is generated.

5.12 CD1284 Parallel Port Overview

5.12.1 Terminology

This document uses the terms 'master' and 'slave' for the IEEE STD 1284 specification terms 'host' and 'peripheral' that describe the two sides of a parallel-port interface.

5.12.2 Signal Names

The IEEE STD 1284 specification uses different names for the nine control signals, depending on the current mode of operation (Table 20). The CD1284 uses fixed names for each of its pins. The names were selected to represent the most commonly used names of the various protocols. The CD1284 device operates as a slave only. There are four input-control signals driven by the master-side device, and five output-control signals driven by the slave-side device. The Parallel Data bus (PD[7:0]) is bidirectional.

Table 20. Signal Names (Sheet 1 of 2)

Names	Compatibility	Rev. NB	Rev. BT	ECP	EPP
Inputs					
A_1284	SLCTIN*	A_1284	A_1284	A_1284	nAStrb
HstBsy	AUTOFD*	HstBsy	HstBsy	HstAck	nDStrb
HstClk	STROBE*	HstClk	HstClk	HstClk	nWrite
nInit	INIT*	nInit	nInit	nRevReq	nInit

Table 20. Signal Names (Sheet 2 of 2)

Names	Compatibility	Rev. NB	Rev. BT	ECP	EPP
Outputs					
AkDaRq	PError	AkDaRq	AkDaRq	nAkRev	USER1
PerBsy	BUSY	PerBsy	PerBsy	PerAck	nWait
PerClk	ACK*	PerClk	PerClk	PerClk	Intr
nDatAv	FAULT*	nDatAv	nDatAv	nPerReq	USER2
XFlag	SELECT	XFlag	XFlag	XFlag	USER3

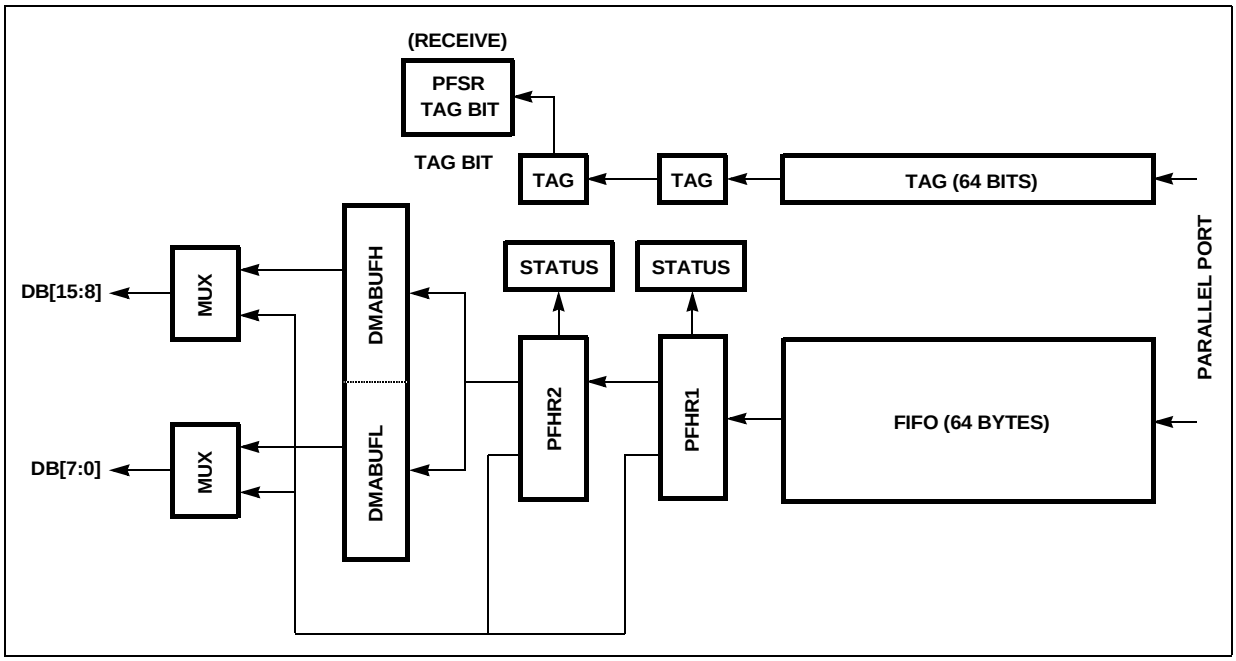
5.12.3 State Machine

The parallel port is controlled by a large synchronous state machine. The state machine is based on the IEEE STD 1284 specification and conforms to all the functional modes (except extensibility link options, none of which are currently — as of the print date of this document — defined).

5.12.4 Configuration

At power-up, the interface begins in Compatibility mode (Centronics mode) ready to accept data from the master. Only the ETxfr bit (PCR[5]) is required to allow transfers in Compatibility mode (parallel port only; datapath section is separate). PCR[7:5] enable transfers and Negotiation and Manual modes.

Figure 11. FIFO Data Path Functional Diagram — Receive



5.12.5 Interrupts

Interrupts are enabled in the PCIER and interrupt status can be read in the PCISR. These two registers have the same format.

5.12.6 Manual Mode

Manual mode allows direct control of the five output control signals and the PD bus. It is not intended for data transfers, but rather for advanced diagnostics. Enter Manual mode by setting the ManMd bit (PCR[7]) when the interface is in Compatibility mode.

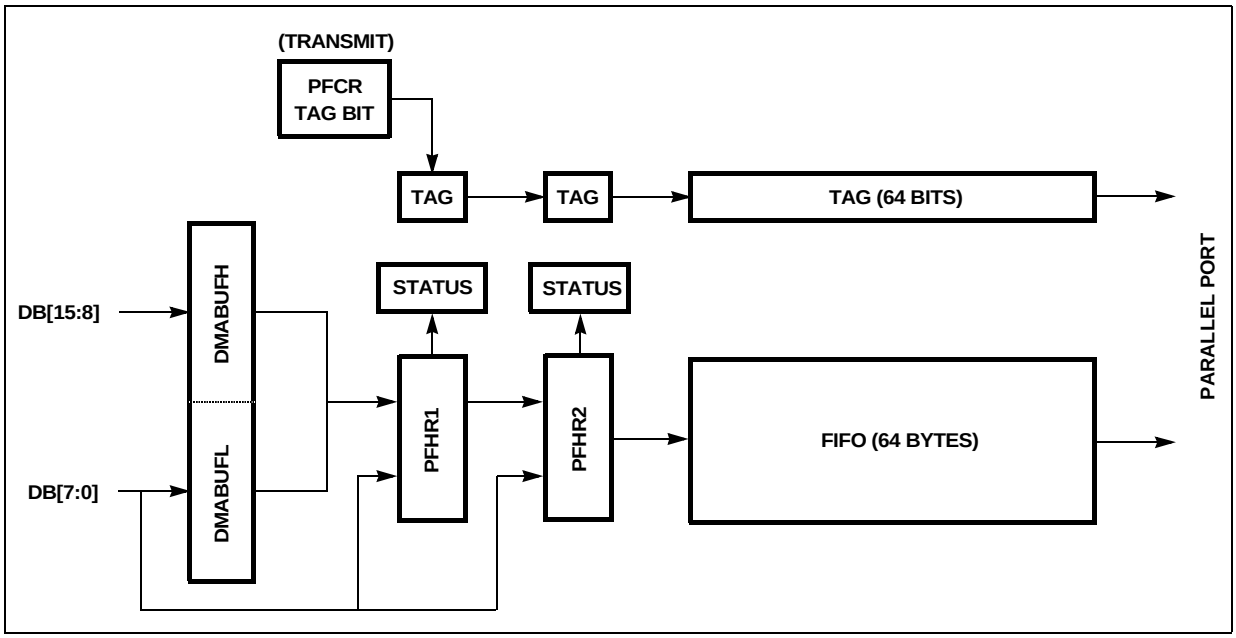
The MMDir bit (PCR[1]) sets the direction of the PD bus: 0 = input; 1 = output. When the MMDir bit is set to ‘1’, data for the PD bus comes from the MDR. The ManOE bit controls the tristate buffer on the PD bus: 0 = floating; 1 = driving. When MMDir is ‘0’, ManOE is ignored, PD[7:0] are inputs, and the data can be read in the MDR.

5.12.7 Control Signals

Output signals are controlled by the OVR. The degree of control depends on the current mode. In Manual mode, all five signals are under user control. In Compatible and EPP modes, only three signals are available; the others are set by the state machine.

IVR, ZDR, ODR, and SSR monitor the four input signals. These four registers have a common format. The IVR always shows the values of the four input pins. The ZDR and ODR allow the user to force interrupts on specific signal transitions. Bits set in ZDR generate an interrupt if the specified signal changes from ‘1’ to ‘0’. Similarly, bits set in ODR cause an interrupt if the specified signal changes from ‘0’ to ‘1’. Setting both bits generates interrupts on either transition. The SSR shows the status of signal changes according to ZDR and ODR. SSR indicates which signal changed. (It is necessary for the user to read IVR to determine how the signal changed.) The signal change interrupt is enabled with the SigCh bit (PCIER[4]).

Figure 12. FIFO Data Path Functional Diagram — Transmit



5.12.8 Parallel Port Interface to the FIFO

The DMAdir bit indicates the current direction (0 = in; 1 = out) of transfers between the FIFO and the DMA logic. Due to a recent negotiation, this can differ from the current parallel-port interface direction. The CPU must change the direction after it receives an interrupt showing a direction change. The FIFOlock bit (PACR[4]) stops the DMA pipeline, useful in diagnostics.

5.12.9 1284 Negotiations

All IEEE STD 1284 protocol negotiations are initiated by the master side. The role of the CD1284 is to accept or reject the attempted negotiation. The NER contains bits to individually enable specific IEEE 1284 modes.

The various IEEE 1284 modes require negotiations on the parallel interface before they can be entered. Until a successful negotiation sequence is complete, the interface remains in Compatibility mode. These negotiations occur in two stages; both stages occur automatically after the device is commanded to begin the negotiation procedure to a particular mode. The first stage determines if the slave is IEEE 1284-compatible. Once determined, the interface continues the process to determine if the mode requested is supported. The result of the requested negotiation appears in the NSR.

For negotiations to occur, the slave must enable the E1284 bit (PCR[6]). Data transfers require that the ETxfr bit (PCR[5]) be set; negotiations can occur without data transfer enabled.

Negotiation Status Register

After any IEEE-1284 negotiation or termination, the current protocol status can be read in the NSR. NegOK and NegFl (bits 7:6) indicate successful and failed attempts. Invalid (bit 4) indicates that the mode terminated from an invalid state. Termination from valid states are reported as successful with NegOK.

A 4-bit code is displayed in the lower portion of the NSR to indicate the results of successful negotiation. This 4-bit code also indicates the mode that the interface was in when an invalid termination was detected, as well as a failed negotiation. Interrupts indicating a successful negotiation into a reverse mode should prompt the CPU to load reverse data into the FIFO.

Special Command Register

The bits in the SCR cause actions on the parallel port. SetPs and ClrPs (bits 3:2) control data movement into the CD1284 from the remote master. In Compatibility mode this function posts error status to the remote. Errors can only be presented to the master by the slave during the active BUSY period. SetPs causes the CD1284 to stop transfers by asynchronously asserting the BUSY signal. To protect against the possibility of data loss, one more byte can be strobed into the CD1284 after BUSY goes active due to the setting of SetPs. When the error status is delivered, ClrPs restores the parallel interface to the normal running state.

EPIrq sends an interrupt pulse in EPP mode. Setting the RevRq bit indicates to the host parallel port that data is available for reverse transfer in either Compatible or ECP mode. These operations are further described in the relevant protocol sections.

5.12.10 Data Transfers

In Compatibility mode, incoming HstClk (STROBE*) pulses activate PerBsy (BUSY), and the data on the PD lines is held in latches. PerBsy protects the data latches by signaling the master it is not ready for more transfers. After the HstClk pulse ends, a pulse is sent on PerClk (ACK*) to acknowledge the receipt of the data into the holding latches. After the data moves from the latches to the FIFO, PerBsy goes low to signal readiness for the next character.

All other data transfer modes require IEEE-1284 negotiations.

5.12.11 Compatible Mode Status

The IEEE 1284 specification requires that the three Compatibility mode status lines (SELECT, FAULT*, and PError) must not be asserted unless PerBsy (BUSY) is high. PerBsy can only be activated in response to a received character, and must remain high until the status condition (for example, paper out) changes.

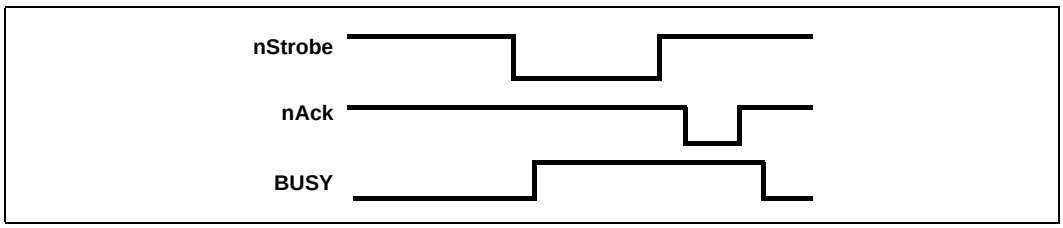
To send these status signals to the master device, set the SetPs bit (SCR[2]) and the appropriate bit in the OVR for each of the status signals. The SetPs bit activates PerBsy, which remains active until ClrPs (SCR[3]) is set. No data is lost in this operation.

5.13 1284 Parallel Protocol Support

5.13.1 Compatibility Mode

Compatibility mode provides backward compatibility with Centronics and PC-compatible printer interfaces. When the host parallel port is in Compatibility mode (with no data transfer in progress), the host can initiate data transfers in Compatibility mode or initiate negotiations to a new operating mode.

Only Busy-while-Strobe and Ack-in-Busy timing is supported in Compatibility mode. Busy-after-Strobe, Ack-after-Busy, and Ack-while-Busy timings are not supported.



5.13.2 Reverse-Nibble and Reverse-Byte Modes

These modes support reverse transfers only, from slave to master. Reverse-Nibble mode is enabled with NER[0]; Reverse-Byte mode is enabled with NER[1]. Reverse-Nibble mode sends 4 bits at a time over four of the peripheral status lines. With software drivers the advantage of this scheme is that any unidirectional PC parallel port can be used for bidirectional data transfers. Reverse-Byte mode requires bidirectional buffers on the PC hardware, but allows substantially faster transfers because it moves one byte at a time.

There is no mechanism in Compatibility mode for the slave to indicate that data is available for reverse transfers. The master must poll the slave by negotiating into a reverse mode and examining the nDatAv signal. During negotiation, RevRq (SCR[0]) instructs the CD1284 to post the availability of data to the master through the nDatAv signal.

5.13.3 ID Request

An ID request is enabled with a combination of NER[6] and one of four other transfer mode bits. ID requests can be made in conjunction with ECP, ECP/RLE, Reverse-Byte, and Reverse-Nibble modes; there is no ID request function defined for EPP mode. The CD1284 can accept an ID request in any mode where it is enabled to do transfers. IDReq is set when an ID request is received in any enabled mode.

5.13.4 ECP Mode

ECP mode allows bidirectional transfers and supports the RLE-compression scheme. The ability to expand RLE data is required of all IEEE-1284, ECP-compliant devices, but the ability to compress data is optional. The CD1284 handles both expansion and compression in the data path section. The parallel port simply passes the inverse of the command signal to/from the FIFO on the ninth tag bit in the FIFO. ECP mode is enabled by NER[2]. RLE mode enabling requires both NER bits 2 and 3.

The handshake is identical for both ECP and RLE modes. The control signals, HstBsy and PerBsy (in the forward and reverse directions, respectively), indicate command and address options. If HstBsy/PerBsy is low, the upper bit of the byte is examined: ‘0’ indicates to interpret the lower 7 bits as an address; ‘1’ indicates to use the lower 7 bits as an RLE repeat count. This count shows the number of times to consecutively repeat that the next data character in the datastream.

The master device is responsible for determining the direction of the transfer. The slave can request a direction change, but the master actually changes the direction. ECP mode always begins in the forward direction, from master to slave. The CPU sets the RevRq bit (SCR[0]) to request reverse transfers. Once the master changes direction, RevRq is automatically cleared and the DirCh interrupt status appears in PCISR (if enabled in the PCIER).

The master device switches the direction of the interface for forward transfers when the slave indicates no more data is available.

5.13.5 EPP Mode

Data transfers use the DMA pipeline and the FIFO. Address transfers are handled out-of-band, not in the FIFO stream. When the slave receives an address write command, it deposits the address into the EAR and asserts an EPPAW interrupt request. When the slave receives a read address command, the contents of the EAR are returned.

5.14 Protocol Timing

The IEEE-1284 specification timing parameter T_P specifies the minimum pulse width and the minimum setup time as 500 ns. The SPR must be loaded with the number of system clock ticks equivalent to 500 ns.

Table 21. System Clock Settings

CLK Freq. (MHz)	Time/Tick (ns)	SPR Value	T_P Width
16	62.5	8	500
20	50	10	500
25	40	13	520

5.15 General-Purpose I/O Port

The CD1284 provides an 8-bit general-purpose port (GP[7:0]) to control or give status of external functions. Each of the eight signals is individually programmable for direction, so the port can be comprised of any number of inputs and outputs. Each port signal is implemented with a standard, bidirectional HCMOS pad and is fully TTL compatible. The port is controlled by two internal registers – GPDIR and GPIO.

Each bit in the GPDIR sets the direction of the corresponding bit in the GPIO; ‘1’ sets the signal as output; ‘0’ sets it as input. When writing to the GPIO, only the bits programmed as outputs are affected by the contents of the data bus. When reading the GPIO, bits programmed as inputs reflect the true state of the condition of the external pin; bits programmed as outputs reflect the state of the last value written to the register and the current state of the output pins.

At reset, all bits in the GPIO are cleared and the signals are programmed as inputs.

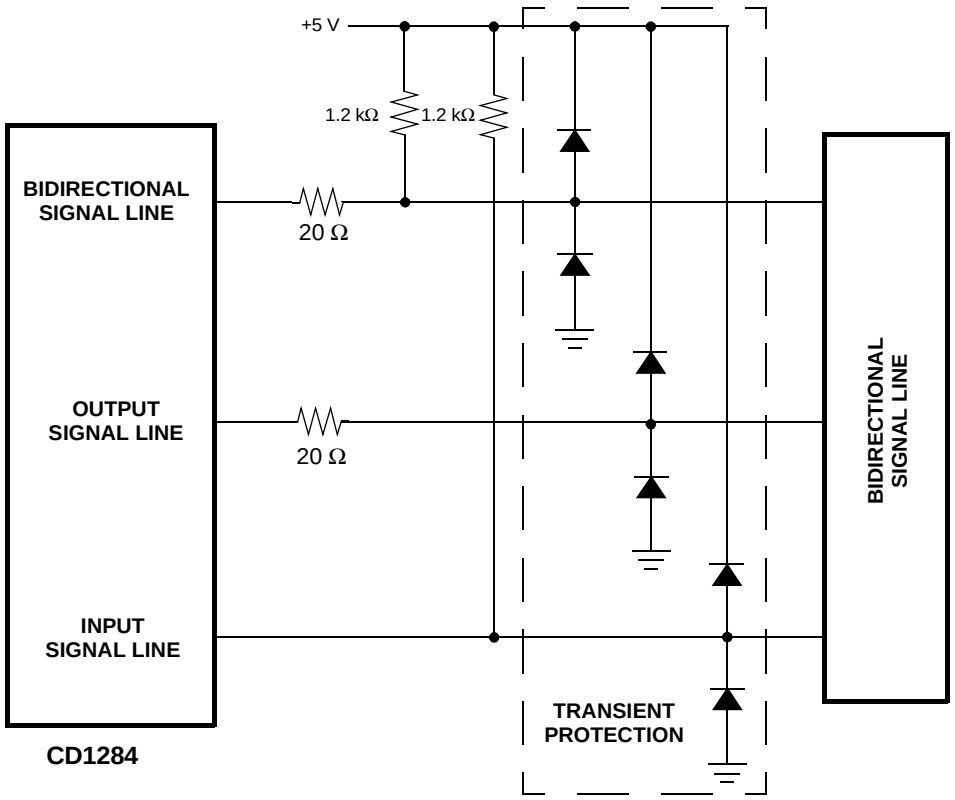
Note: Interrupts are not generated on signal changes within the General-Purpose I/O port; the CPU must periodically poll GPIO to detect changes in external conditions. Therefore, if it is necessary to detect changes, use the port with signals that change with low-duty cycles.

5.16 Parallel Port Interface

The CD1284 parallel port signals are implemented with Level 2 characteristics – as defined in the IEEE STD 1284 specification with the exception of transient protection. The port can be directly connected to the interface cable with the addition of a few external components. The components consist of passive pull-up resistors, series-impedance-matching resistors, and clipping diodes. Additional noise-filtering may be required in an end system. Figure 13 on page 85 shows a typical interface with the components listed above.

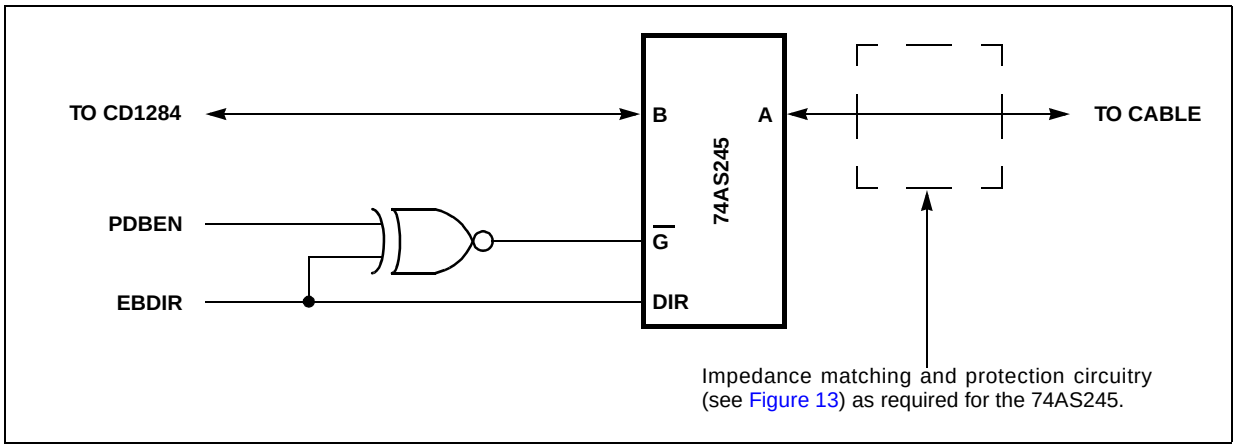
Some system designs may require buffers between the CD1284 and the cable. Systems that require drive cables longer than the specified maximum of 10 m or those that need to protect the CD1284 require inexpensive buffers between it and the cable. The device provides two signal outputs, PDBEN and EBDIR, that can to connect and control buffers (such as, 74AS245 or equivalent). These signals do not allow direct control of the buffer. However, the addition of an XNOR gate provides both an enable control signal and a signal to select the direction of the buffer. PDBEN and EBDIR are outputs from the control state machine that indicate its current state (see Figure 14 on page 86).

Figure 13. Cable Connection



Caution: Transient protection is not implemented inside the CD1284 device, therefore transient voltages may cause damage. Laboratory testing has shown that this type of protection is not necessary under normal conditions. However, damage may occur under harsh conditions or when subjected to unusual abuse. Also note, the protection circuit shown here may cause a powered-up host to supply power to the +5 V (V_{CC}) of the peripheral if it is not powered up. If this is a concern, then another protection circuit must be designed.

Figure 14. External Buffer Control



5.17 Hardware Configurations

The simplicity of the CPU interface to the CD1284 allows the device to be designed into systems that employ popular microprocessors such as the Intel 80x86 family (8086, 80286, 80386, and so on), the Motorola® family (68000, 68010, 68020, and so on), the National Semiconductor® 32x32 family (32CG16, 32332, 32532, 32GX32, and so on), and the AMD® 29000.

5.17.1 Interfacing to an Intel® Microprocessor-Based System

With very little extra logic, the CD1284 can interface to any system based on a processor in the Intel 80x86 family. Figure 15 shows a generalized view of an I/O-mapped interface with an 80286-based system. To provide the proper strobes and controls, the IOR* and IOW* control strobes synthesize the DS* and R/W* signals. DTACK* is used as an input to wait-state-generation logic that holds the processor (if necessary) until the CD1284 has completed the I/O request.

5.17.2 Interfacing to a Motorola® Microprocessor-Based System

Interfacing to a 68000 family device is relatively simple. Bus timing and the interface signal definitions closely match those of the 68000 microprocessor, which allows a direct connection in most cases. With later versions (68020, 68030), some additional logic is required to generate the DSACK0* and DSACK1* functions that replace the DTACK* on earlier devices. The example in Figure 16 on page 88 shows a generalized interface to a 68020 device.

5.17.3 Interfacing to a National Semiconductor® Microprocessor-Based System

The connections between the CD1284 and an NS32000 (32GX320, 32CG16, and so on) embedded controller are also relatively simple. As with the Intel devices, cycles are controlled by the DS*, CS*, and R/W* signals synthesized from the available I/O-control signals. I/O-cycle extensions (wait states) are generated by logic connected to the DTACK* signal. All necessary controls are available to prevent multiple read/write cycles in the CD1284 FIFOs when using memory-mapped I/O.

Figure 17 on page 89 depicts a simplified interface example.

Figure 15. Intel® 80x86 Family Interface

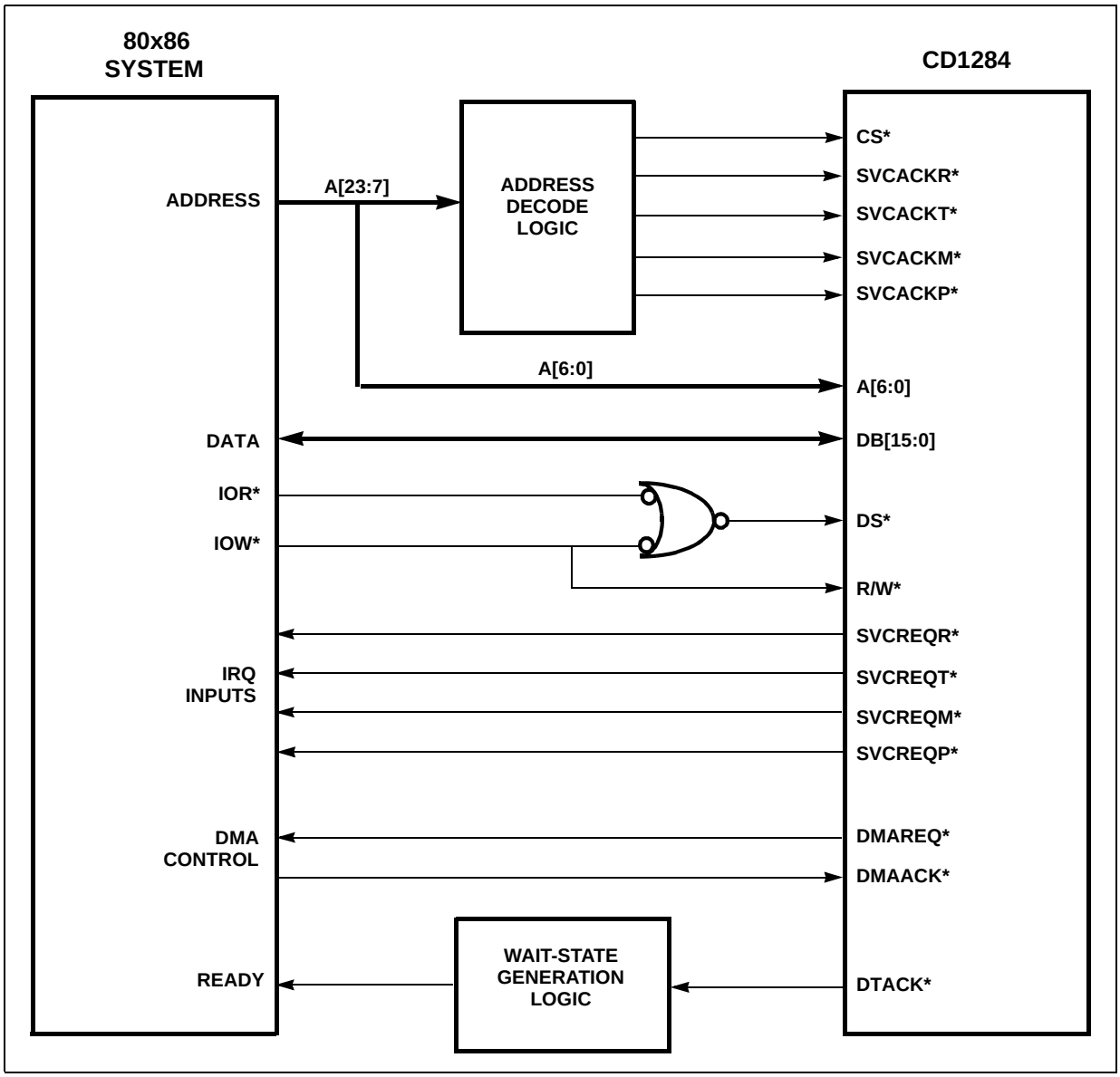


Figure 16. Motorola® 68020 Interface

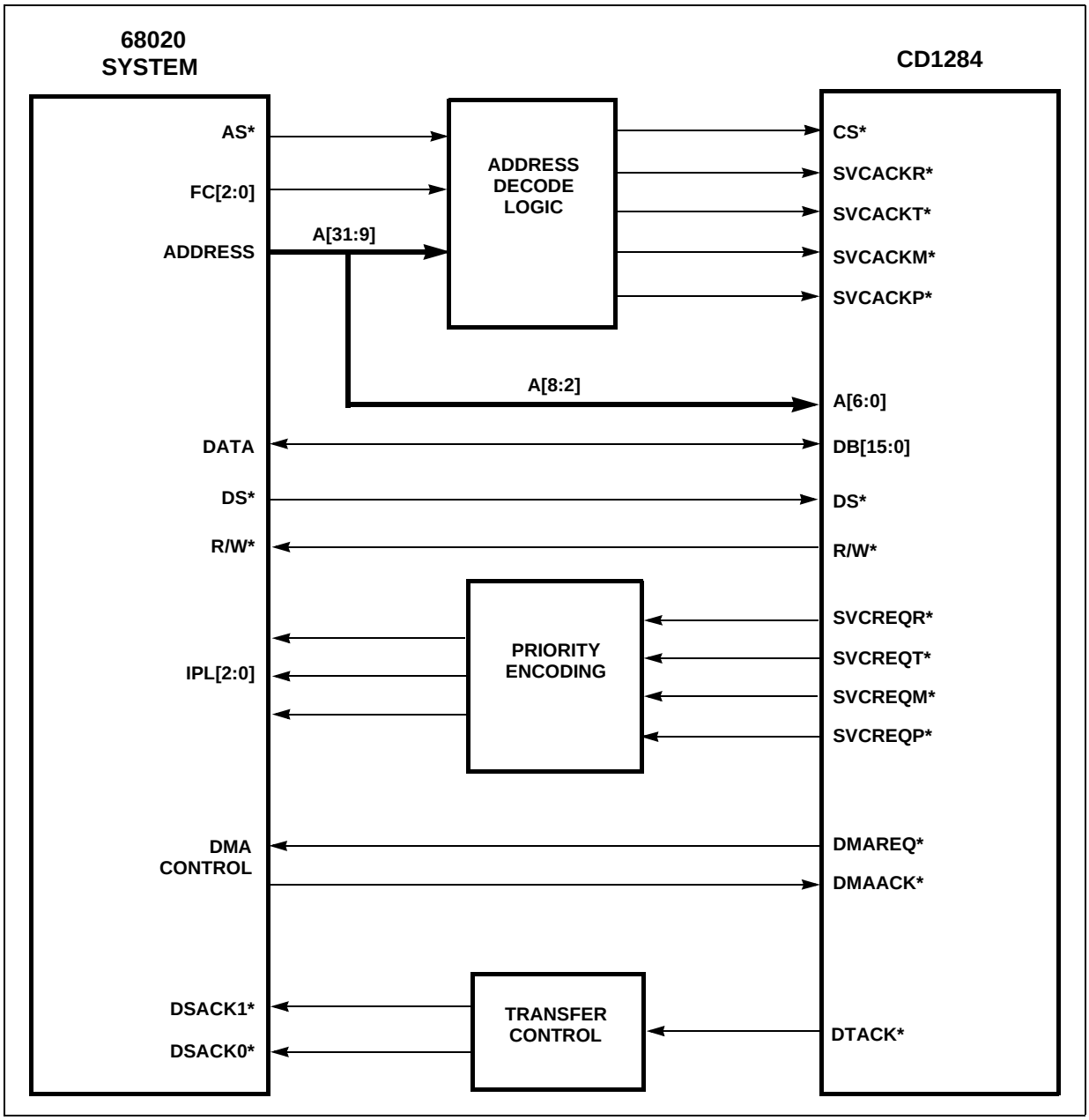
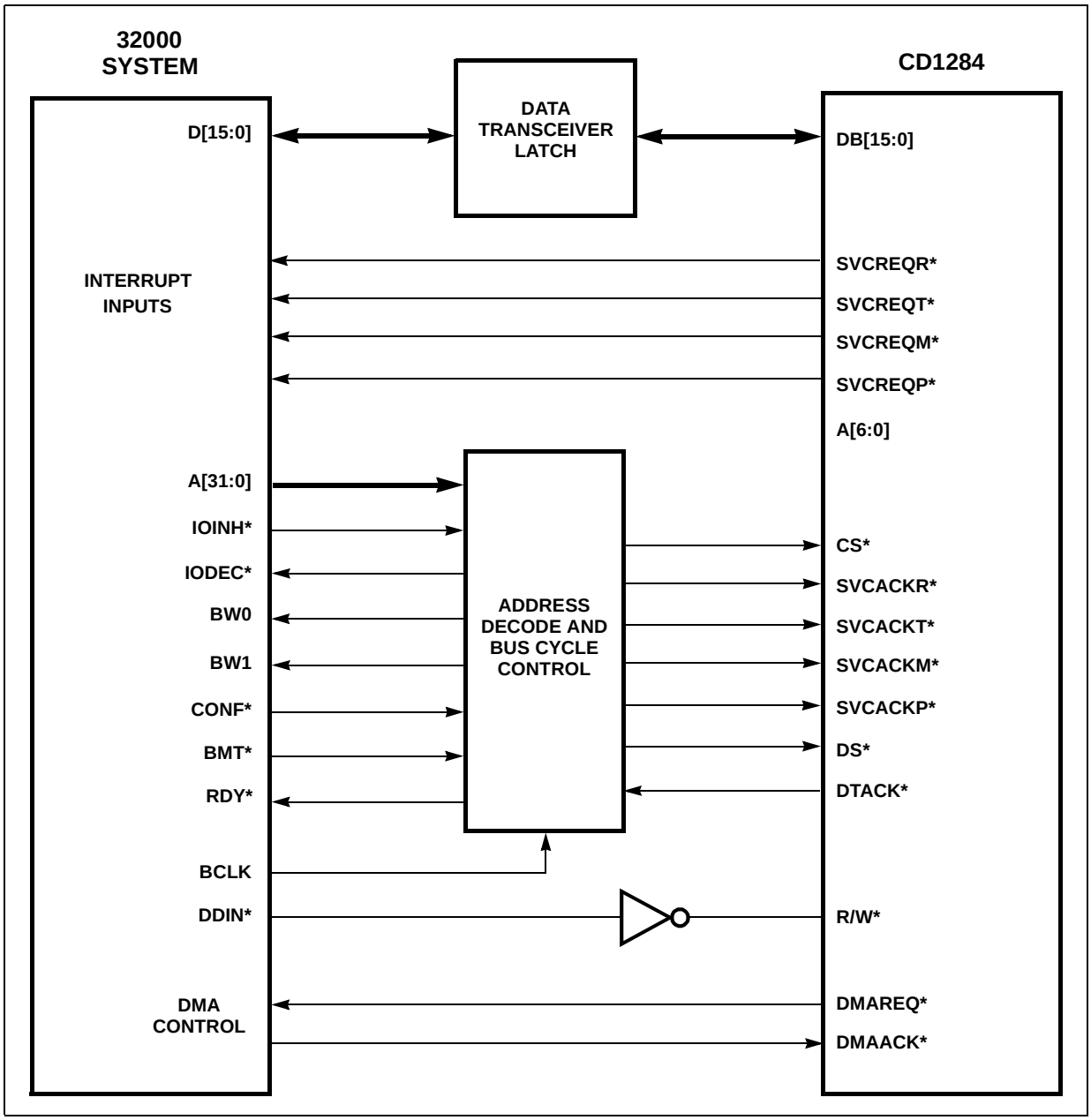


Figure 17. National Semiconductor® 32000 Interface



6.0 Programming

6.1 Overview

As shown in the register summary tables in Chapter 4.0, the CD1284 local CPU interface is made up of a large array of registers. These registers control all aspects of device behavior; some affect overall chip operations, and others affect only one channel. Fortunately, most of the registers are only modified once, during initialization, and rarely modified during normal operation. The purpose of this chapter is to discuss these aspects, as well as the methods of interacting with the CD1284 for channel-service needs.

6.2 Initialization

To properly power-up a CD1284, several procedures must be completed. These include device initialization, programming global functions, and setting channel-specific parameters. In most cases, initialization routines are executed once; during the overall system boot-up. The following sections discuss these steps in detail (see Figure 18 on page 92 for a flow-chart step outline).

6.2.1 Device Reset

The procedures that perform chip reset are normally executed after a power-up, system-wide reset. The hardware reset control signal, RESET* causes the CD1284 to perform its own internal initialization. If desired, the driver software can issue a full chip reset before chip initialization begins. To accomplish this, use the following steps (see Figure 18 for a flow-chart step outline):

1. Wait for CCR (Channel Command register) to contain 0x00.

   The contents of the CCR must be ‘0’ before a command is issued. This is required to ensure that any currently executing command has completed before the new one is started. Since this is probably the first command being written to the CD1284 after power-on initialization, the CCR is likely to be ‘0’, but it is recommended to always check the CCR before writing in a new command.

2. Set the CAR (Channel Access register) to one of the two serial channels (2 or 3).

   This step is required when the parallel channel does not respond to any value written to the CCR address (this register does not exist in the parallel channel).

3. Write hexadecimal 81 (x’81) to the Channel Command Register (CCR).

   This command causes the CD1284 to perform an all-channel and global reset. It causes the internal RISC processor to begin execution from its power-up reset location. The results are the same as if the RESET* input is activated. All internal interface registers are cleared, the FIFOs are flushed, and all channels are disabled.

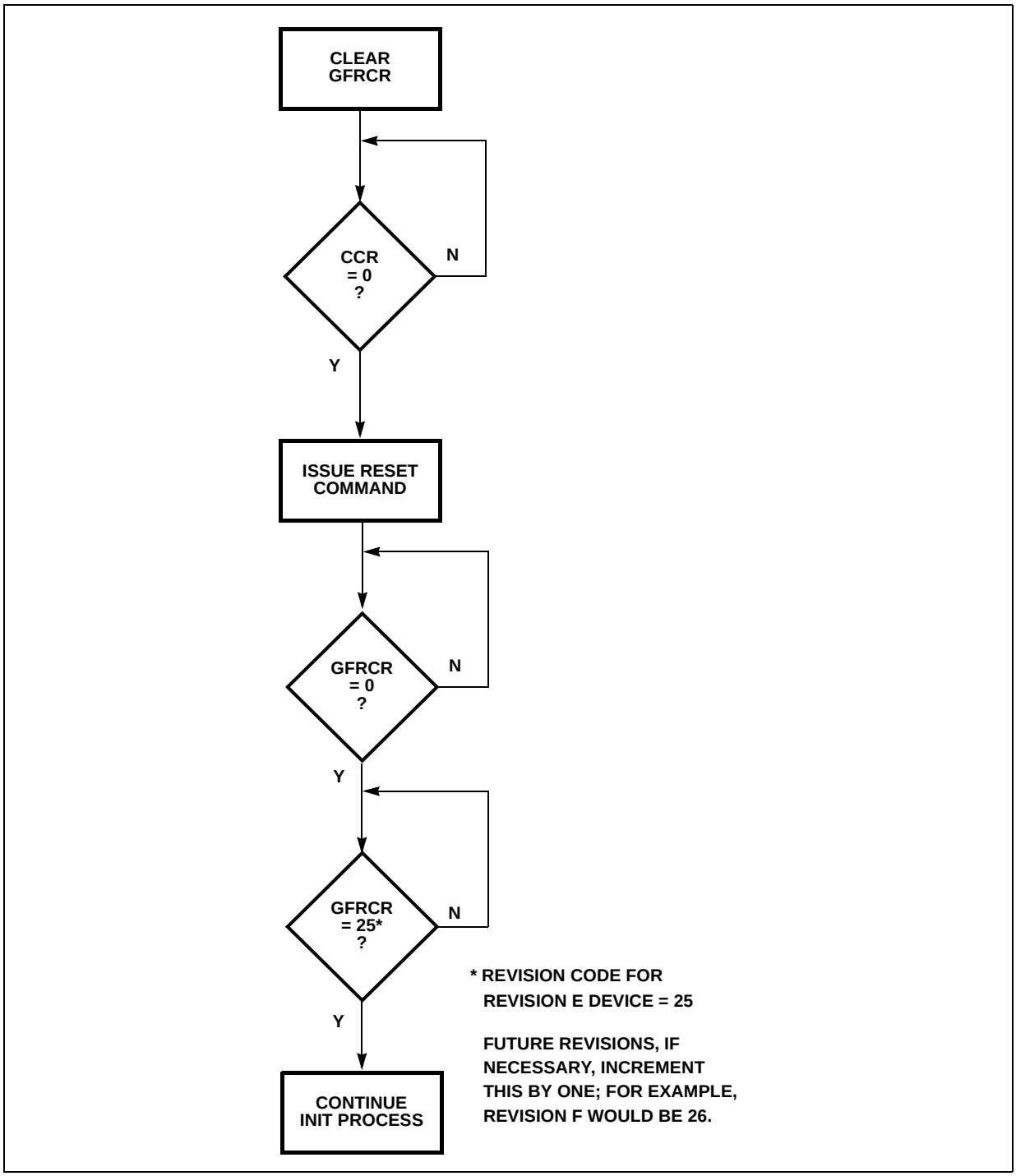
   The full-chip reset command is a special-case CCR operation. Normally, the commands issued to the CCR affect only the channel selected by the CAR. In this case, the setting of the CAR is insignificant, but must be set to channel 2 or 3. Unlike other commands issued to the CCR, the global reset command does not use the clearing of the CCR. Instead, the GFRCR indicates that the command is complete (see below).

4. Wait for the firmware revision code to be written into the GFRCR.

Internal firmware uses this operation to flag completion of the reset procedure. After the reset is issued, the GFRCR is one of the first registers cleared and it is the last one set before normal runtime code execution begins. The initialization routine *must* wait for this register to become non-zero before it begins any other programming of CD1284 registers. If the CPU is sufficiently fast, it could begin testing the GFRCR before the MPU clears it. The assumption could be made that the CD1284 has completed internal initialization when, in fact, it has not even started. To avoid this error, the CPU should look for the GFRCR to change to ‘0.’ It should then look to the current revision code. Alternatively, the CPU can clear the GFRCR just prior to issuing the global reset command and then poll for the correct revision code. This is useful in slow systems that cannot guarantee that the CPU can check the register after it is cleared or before it is loaded with the revision code.

This procedure is also used as part of a diagnostic test suite. The device completes internal initialization within 500 µsec. A timer (software or hardware) detects when the operation is not completed within that time and cues if the device is functional.

Figure 18. Flow Diagram of CD1284 Master Initialization Sequence



6.2.2 Global Function Initialization

Once chip reset has been completed, the next step is to set the Global Operating mode and timer prescale. All other initialization occurs at the channel level.

Set the Prescaler Period Register (PPR)

The PPR sets the master time ‘tick’ for the CD1284. It is a binary value that sets a constant by which the system clock is divided (after a fixed prescale of 512) to produce the internal clock for the on-chip timers (This does not include baud rate generators). This clock is used for receiver FIFO time-out generation and delay timing for the insert delay command in the embedded transmit command set. For example, to generate a timer clock of 1 ms, the value is computed as:

$$\left(\frac{25MHz}{512}\right) \times 1ms = 48.828$$

The value 49 is loaded into the PPR. This value, selects an approximate 1-kHz clock as the source for the RTPR (Receiver Time-out Period registers) of each channel. Those registers are loaded with an appropriate value divisor that generate the desired character time-out periods. This value, 49, is the recommended minimum value that is placed in the PPR for a clock frequency of 25 MHz. Values that generate a time period of less than 1 ms adversely affect the performance of the MPU, and thus, overall serial data performance.

6.2.3 Serial Channel Initialization

At this point, the basic operation of the CD1284 serial channels are set up. The internal register states are cleared and basic timer operations are initialized. The next step is programming the operating modes of each channel. This includes setting the values for the interrupt vectors, the receive and transmit baud rates, number of bits per character, number of stop bits, parity, special characters, if any, and so on. Each channel can have a completely unique set of operating characteristics or they can all be the same.

Serial channel initialization is application-dependent. The operating modes of one channel have no effect on the operation of others.

The following code, is a typical initialization sequence for setting up a single serial channel:

```
9600 baud, send and receive
8 bits per character, 1 stop bit,No parity
Automatic In-Band (Xon/Xoff) flow control
Transparent flow control
Special character detect enabled
Eight character receive FIFO threshold
Receiver and transmitter enabled for interrupt operation
Enable ISTRIP on incoming characters
```

A clear way of showing this initialization sequence is by a ‘C’ program fragment; the code shown below, is compatible with Borland® Turbo C™:

```
/* Init channel. Channel number is included in call. Register names and addresses
are defined in the header file (not shown). This routine does not include parallel
channel initialization. */
init_channel(chan)
char         chan
{
```

```
        outportb(CAR, chan);   /* set channel number in CAR */
        outportb(RTPR, 0x14);  /* set channel time-out value (20ms) */
        outportb(TCOR, 0x01);  /* constants for 25 MHz clock – clock option*/
        outportb(TBPR, 0x51);  /*          – baud rate period */
        outportb(RCOR, 0x01);  /* constants for 25 MHz clock – clock option*/
        outportb(RBPR, 0x51);  /*          – baud rate period */
        outportb(COR1, 0x03);  /* no parity, 1 stop bit, 8 bit chars */
        outportb(COR2, 0x40);  /* auto. in-band flow control */
        outportb(COR3, 0x38);  /* transp. flow-control, special char 1 & 2 detect,
fifo thresh = 8 */
        while (inportb(CCR) != 0)/* make sure that CCR is zero before issuing
commands */
             ;
        outportb(CCR, 0x4E);   /* issue COR changed command for COR1, 2, 3 */
        outportb(COR5, 0x80);  /* enable ISTRIP */
        outportb(SRER, 0x14);  /* enable receive and transmit interrupts */
        while (inportb(CCR) != 0)/* make sure that CCR is zero before issuing
commands */
             ;
        outportb(CCR, 0x1A);   /* issue receiver and transmitter enable command to
CCR */
}
```

6.3 Serial Poll Mode Examples

The CD1284 provides a set of seven registers dedicated to Poll-mode operation, described in Chapter 5.0. This section shows one of many ways that these registers are used to detect and service requests from any of the channels receiver, transmitter, or modem signal change functions.

The primary registers involved in polling are: SVRR, RIR, TIR, MIR, and CAR. The supplementary registers are: RIVR, TIVR, and MIVR. Of the latter three registers, only RIVR is actually used. RIVR provides the service request status for ‘good’ data or exception data. The TIVR and MIVR provide redundant information and are rarely used. Other registers related to service requests (TDR, RDSR, MISR, and so on) perform the same functions as in hardware-acknowledged service requests. The parallel channel uses a slightly different register manipulation procedure and is shown separately. The top-level polling routine is the same regardless of the type of request serviced.

Once again, C code fragments describe the functions. As with other coding examples, it is assumed that register addresses are defined elsewhere, such as in a header file. The routines cannot be considered complete. The routines cannot be considered complete; some pieces are dependent on the system software design and the code presented is only an example. The pieces do, however, show methods used to implement the poll mode service request/service acknowledge sequence.

6.3.1 Polling Routine Examples

6.3.1.1 Scanning Loop

```
/* Poll-mode code fragments routinely check for any servicing requests and branches
to the appropriate service routine. The code prioritizes service requests as receive,
transmit, modem and parallel, in that order. System design dictates the actual
priorities required. Note that the routine ignores the state of the DMA active bit.
*/
poll( )
{
        char        status;
        char        rx_stat = tx_stat = md_stat = 0, par_stat = 0;

        if (status = inportb(SVRR) & 0x0F) {/* Mask off DMA status */
```

```
            switch (status) {
                  case 1:               /* all values that include a receive
request */
                  case 3:
                  case 5:
                  case 7:
                  case 0xF:
                        rx_stat = service_rec( );
                        return(rx_stat);
                        break;
                  case 2:               /* all values that include transmit but
not receive */
                  case 6:
                  case 0xA:
                  case 0xE:
                        tx_stat = service_txm( );
                        return(tx_stat);
                        break;
                  case 4:               /* modem service request */
                  case 0xC:
                        md_stat = service_mdm( );
                        return(md_stat);
                        break;
                  case: 8:              /* parallel port service request */
                        par_stat = service_par();
                        return(par_stat);
                        break;
                  default:              /* can't happen */
                        break;
            }
      }
}
```

Once the code above locates an active request posted in the SVRR, it calls the appropriate subroutine to service the request. The service routines follow.

6.3.1.2 Serial Receive Service

```
/* The receive service acknowledge cycle begins by reading the RIR. This register
contains the necessary information to switch the CD1284 into the correct service
acknowledge context. The RIR is saved for use at the end of the routine and then
copied into the CAR. The act of copying the RIR into the CAR forces the context
switch. The channel number requesting service is extracted from the RIR. The RIVR
register indicates whether the request is for good data or exception data and is
used to correctly handle the request. At the end of the service, the upper two bits
in the RIR are cleared causing the switch out of the service acknowledge context. */

service_rec( )
{
      char  serv_type, save_rir, save_car, channel, status, char;
      int   char_count, i;

      save_rir = inportb(RIR);          /* retrieve and save receive interrupt
value */
      channel = save_rir & 0x03;        /* extract channel number from the RIR*/
      save_car = inportb(CAR);          /* save CAR for restore */
      outportb(CAR, save_rir);          /* switch CD1284 to service ack. context
*/
      serv_type = inportb(RIVR) & 0x07; /* read vector register; get type (good/
exception)*/
      switch (serv_type) {
            case 3:                     /* good data service */
                  char_count = inportb(RDCR);           /* get number of
characters in FIFO */
                  for ( i = 1; i <= char_count; i++) {/* - read that number of
chars */
                        char = inportb(RDSR);  /* read char from FIFO */
```

```
                    /* Code here would put the character in a buffer of some sort for
each
                     * channel. That code would be dependent on system software design
                     * so it won't be shown here. */

                    }
                    outportb(RIR, save_rir & 0x3f);/* terminate service ack.
sequence */
                    outportb(CAR, save_car);    /* restore original CAR* /
                    return(0);
                    break;
             case 7:                    /* exception data service request */
                    status = inportb(RDSR);/* by definition, only one char; get
status */
                    outportb(RIR, save_rir & 0x3f);/* terminate service ack.
sequence */
                    outportb(CAR, save_car);    /* restore original CAR */
                    return(status);       /* just return the error type */
                    break;
       }
}
```

6.3.1.3 Serial Transmit Service

```
/* The transmit service acknowledge routine follows very nearly the same steps that
the receive service routine follows. This time, the TIR is used to force the switch
to a transmit service for the requesting channel. */

service_txm( )
{
       char        save_tir, save_car, channel;
       int         char_count, i;

       save_tir = inportb(TIR);          /* retrieve and save transmit interrupt
value */
       channel = save_tir & 0x03;        /* extract channel number from the TIR*/
       save_car = inportb(CAR);          /* save CAR for restore */
       outportb(CAR, save_tir);          /* switch CD1284 to service ack. context
*/

             /* Buffer management code would set-up pointers to the next 12
              * characters (maximum) to be sent on this channel. Again, buffer
              * layout is system design dependent and won't be shown here.
             */

       for ( i = 0; i < char_count; i++) {/* transmit FIFO can take 12 characters
*/
             outportb(TDR, *next_char++);

             /* it is assumed that char_count and next_char is set up by buffer code
*/

       }

       outportb(TIR, save_tir & 0x3f);/* terminate service ack. sequence */
       outportb(CAR, save_car);    /* restore original CAR */
       return(0);
}
```

6.3.1.4 Modem Service

```
/* Code to handle modem signal change service request can be simple or complex
depending on whether port control is handled directly in the service routine or
simply noted with status returned. The following routine services the request and
returns the status of which signals changed with the channel number OR'ed into the
```

```
least-significant two bits; the main driver software must perform the necessary
functions. As with the receive and transmit routines, the Interrupt register, this
time the MIR, is used to force the CD1284 into the service context. */

service_mdm( )
{
        char        save_mir, channel, save_car, mdm_status;

        save_mir = inportb(MIR);          /* retrieve and save modem interrupt value
*/
        channel = save_mir & 0x03;        /* extract channel number from the MIR*/
        save_car = inportb(CAR);          /* save CAR for restore */
        outportb(CAR, save_mir);          /* switch CD1284 to service ack. context
*/
        mdm_status = inportb(MISR);       /* get status of which modem signals
changed */
        outportb(MIR, save_mir & 0x3f)    /* terminate the service ack. sequence */
        outportb(CAR, save_car);          /* restore CAR */
        return(mdm_status | channel);
}
```

6.4 Hardware-Activated Service Examples

In nearly all respects, the way that the CPU interacts with the CD1284 during hardware-activated service acknowledge is the same as software-activated methods. The main difference is that the SVCACK* input signals perform the context switch automatically, relieving that duty from the CPU. The result is the same: the CAR is set to point to the correct channel and the device is placed in the proper internal mode to service the request.

When the SVCACK* input is activated, a read cycle is performed. The CD1284 places the contents of the appropriate Interrupt Vector register (RIVR, TIVR, MIVR) of the channel requesting service on the data bus. The CPU uses the information provided to determine the type of service and the ID number of the device being accessed in the case of daisy-chained multiple CD1284s.

At the end of the service routine, the CPU writes a dummy value to the EOSRR. This causes the switch out of the service acknowledge context and restores the environment to what it was before the service began. Again, the parallel port service is slightly different and it is shown separately.

The following code fragments show the differences between this type of service acknowledge and the types shown above for the software-activated context switch. Only the beginning and ending steps are shown; the code between is very similar to the previous examples. These routines can be executed as the result of a hardware interrupt or by software polling as in the previous examples. For the purpose of this discussion, the method of arriving at the proper service routine is not important.

6.4.1 Serial Receive Service

```
/* The receive service acknowledge cycle begins by executing a service acknowledge
cycle, which activates the SVCACKR* input. The data obtained as a result of this
‘read’ cycle is the contents of the RIVR register of the channel making the service
request. The service routine decodes the vector in the least significant three bits
to determine if the data is ‘good’ or ‘bad’ (exception). The context switch is done
automatically when the SVCACKR* signal is activated and the CAR does not need to be
loaded. The routine reads the RICR to determine the requesting channel number. If
this is a multiple–CD1284 system using daisy-chaining, the routine extracts the chip
ID from the upper five bits of the RIVR. */

service_rec( )
{
```

```
        char        serv_type, vector, channel, status, char;
        int         char_count, i;

        vector = inportb(SVCACKR);        /* gen. ack and get vector (read LIVR) */
        channel = inportb(RICR) >> 2;     /* extract channel number from the RICR*/
        serv_type = vector & 0x07;        /* mask RIVR to get type (good/
exception)*/
        switch (serv_type) {
              case 3:                     /* good data service */
                    char_count = inportb(RDCR); /* get number of characters in FIFO
*/
                    for ( i = 1; i <= char_count; i++) {/* - read that number of
chars */
                          char = inportb(RDSR);  /* read char from FIFO */

                    /* Code here would put the character in a buffer of some sort for
each
                     * channel. That code would be dependent on system software design
                     * so it won't be shown here; this code just shows how to
manipulate the
                     * CD1284 registers to implement the poll mode service
acknowledge. */

                    }
                    break;
              case 7:                     /* exception data service request */
                    status = inportb(RDSR);/* by definition, only one char; get
status */
                    break;
        }
        outportb(EOSRR, 0x00);            /* write dummy value to EOSRR to terminate
*/
}
```

6.4.2 Serial Transmit Service

```
/* The transmit service acknowledge routine follows very nearly the same steps that
the receive service routine follows. The SVCACKT* input is activated to start the
service cycle, reading the contents of the TIVR, and the TICR is read to get the
channel number. */

service_txm( )
{
        char        vector, channel;
        int         char_count, i;

        vector = inportb(SVCACKT);        /* retrieve and save transmit interrupt
value */
        channel = inportb(TICR) >> 2;     /* extract channel number from the RICR*/

              /* Buffer management code would set-up pointers to the next 12
               * characters (maximum) to be sent on this channel. Again, buffer
               * layout is system design dependent and won't be shown here.
              */

        for ( i = 0; i < char_count; i++) {/* transmit FIFO can take 12 characters
*/
              outportb(TDR, *next_char++);

              /* it is assumed that char_count and next_char is set up by buffer code
*/

        }
        outportb(EOSRR, 0x00);            /* write dummy value to EOSRR to terminate
*/
}
```

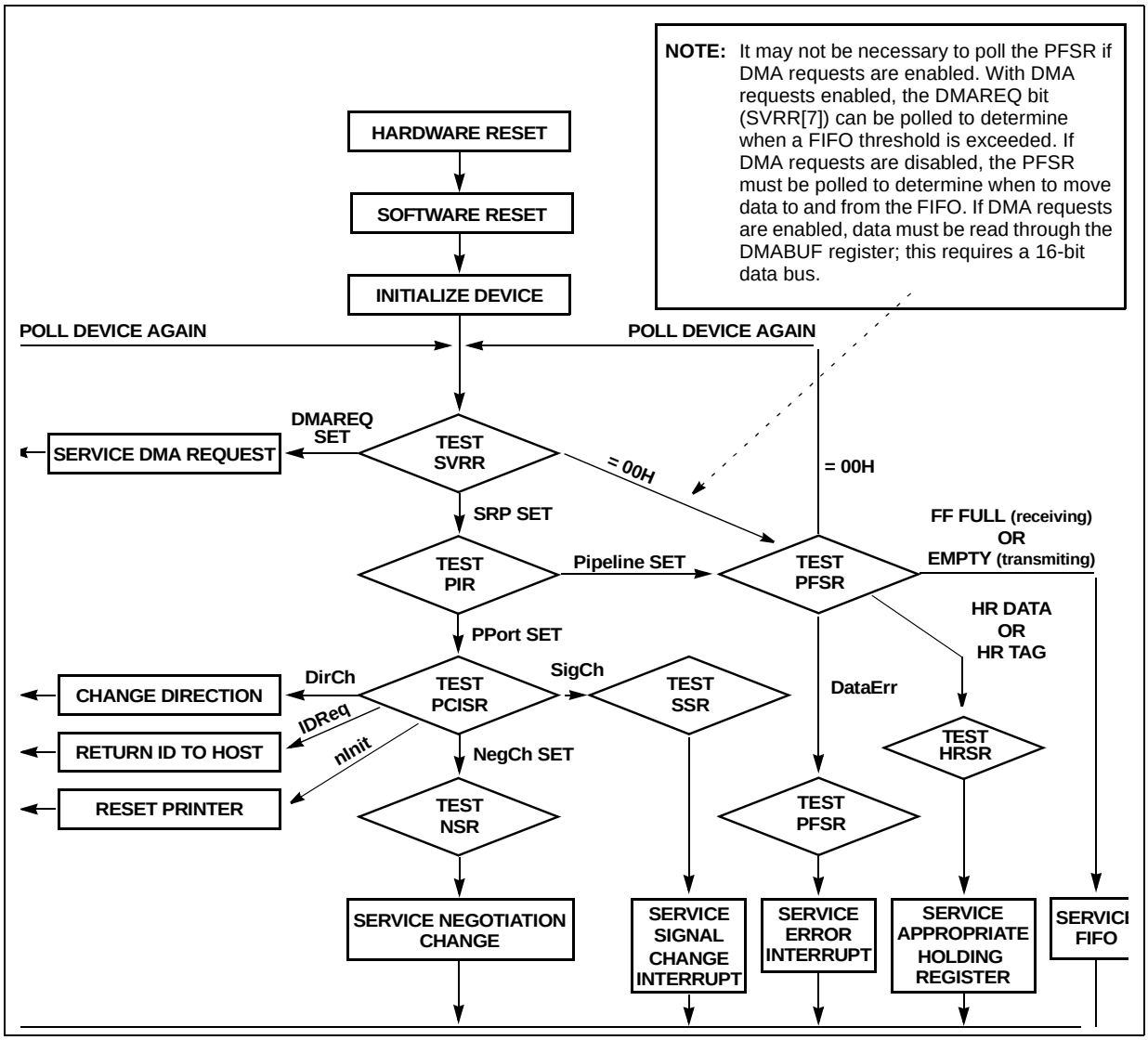
6.4.3 Modem Service

```
/* The following routine services the modem change service request. Context switch
is set up by activating the SVCACKM* input, reading the MIVR. The routine reads the
MISR register to determine which modem signal(s) changed. Channel status is an
externally defined variable that this routine updates. */

service_mdm( )
{
        char        vector, channel;

        vector = inportb(SVCACKM);        /* retrieve and save transmit interrupt
value */
        channel = inportb(MICR) >> 2;     /* extract channel number from the RICR*/
        mdm_status[channel] = inportb(MISR);/* get status of which modem signals
changed */
        outportb(EOSRR, 0x00);            /* write dummy value to EOSRR to terminate
*/
}
```

6.5 Parallel Channel Service Routines

In most respects, the parallel channel functions in the same way as the two serial channels, but the Poll mode operation is different. Its functions can be performed in a couple of ways. The MPU is only involved with the parallel channel in performing interrupt generation services. All other channel operations are completely separate. Since the MPU is involved in the interrupt structure, aspects of its behavior must be taken into account.

Figure 19. Polling Flow Chart



6.5.1 Software-Activated Service Examples (Poll)

The scanning loop for Poll-mode operation is shown in Section 6.3. Software activation of the context switch is performed in the same manner, but termination of the service is done in two ways. The first method is similar to the serial channel method and the second method can work well in certain systems, but requires extra steps.

The first method follows the same basic procedure as the serial channels, but the termination sequence requires only that the upper bit (PPIreq) of the PIR is cleared by the CPU. Since Fair Share is not implemented on the parallel channel, there is no ‘unfair’ bit in the PIR; the ‘busy’ status is maintained by the MPU differently and is not maintained in the PIR.

The routine below shows one way of implementing the poll-mode service activation using the first method.

```
service_par( )
{
       char       save_pir, save_car, livr_val;

       save_pir = inportb(PIR);          /* retrieve and save parallel interrupt
value */
       save_car = inportb(CAR);          /* save CAR for restore */
       outportb(CAR, save_pir);          /* switch CD1284 to service ack. context
*/
       livr_val = inportb(LIVR) & 0x07;
       switch (livr_val) {
            case 4:                      /* just the parallel channel state-machine
request is active */
                  service_par_chan();
                  break;
            case 5:                      /* just the data path pipeline request is
active */
                  service_pipeline();
                  break;
            case 6:                      /* both requests are active */
                  service_par_chan();
                  service_pipeline();
                  break;
            default:
                  break;
       }
       outportb(PIR, save_pir & 0x00);   /* terminate service ack. sequence by
clearing bit 7 */
       outportb(CAR, save_car);          /* restore original CAR*/
       return(0);
}
```

It is not necessary for the CPU on the parallel channel to actually copy the contents of the PIR into the CAR. Since it is known that the parallel channel is always channel 0, the CPU may switch the context by simply writing a x’00 into the CAR after first saving the previous state of the CAR, if desired. At the end of the service, the interrupt context can be returned by toggling the IntEn bit in the PFCR within the data pipeline. Hardware in that block of logic detects the toggle operation and clears the PPireq bit itself. The CPU can restore the CAR, if desired, and exit the routine. Just as it would in the other poll-mode case, once the MPU has detected the clearing of the parallel interrupt source bits (PPort and Pipeline) and the PPIreq bit. It cleans up the PIR and LIVR.

```
service_par( )
{
       char       save_car, livr_val;

       save_car = inportb(CAR);          /* save CAR for restore (if desired) */
       outportb(CAR, 0x00);              /* switch CD1284 to service ack. context
*/
       livr_val = inportb(LIVR) & 0x07;  /* get the vector (Not from the PIVR) */
       switch (livr_val) {
            case 4:                      /* just the parallel channel state-machine
request is active */
                  service_par_chan();
                  break;
            case 5:                      /* just the data path pipeline request is
active */
                  service_pipeline();
                  break;
            case 6:                      /* both requests are active */
                  service_par_chan();
                  service_pipeline();
                  break;
            default:
```

```
                break;
        }
        outportb(PFCR, inportb(PFCR & 0xEF);/* clear IntEn (first step of 'toggle'
operation */
        outportb(PFCR, inportb(PFCR | 0x10);/* set IntEn (second step of 'toggle'
operation */
        outportb(CAR, save_car);          /* restore original CAR (if desired) */
        return(0);
}
```

6.5.2 Hardware-Activated Service Examples

Hardware-activated context switching is nearly identical to the serial case; during the service acknowledge cycle, the SVCACKP* input is active and the CD1284 drives the parallel channel vector on the data bus. At the same time, the MPU pushes the current state of the device on the context stack and sets the context for channel 0. The vector comes from the PIVR, which is a reflection of the LIVR. The vector supplied indicates the source of the request in the IT2-IT0 bits. There is no equivalent to the Interrupting Channel register (TICR, RICR, MICR) since, by definition, the interrupt is from channel 0. Once the context switch occurs, the CPU can proceed to service the source of the request.

The CPU must decode the ITx bits to determine the blocks that require service. Each section of the parallel channel has an Interrupt Status register to indicate what conditions, if any, in that block require service. These are the PFSR in the data path and the PCISR in the channel control state machine.

At the end of the service routine, the CPU must perform the same dummy write operation to the EOSRR as for the serial channels. This informs the device that the parallel service is complete. The write operation to the EOSRR generates a high-priority interrupt to the MPU to cause it to pop the context stack and restores the device environment to what it was at the start of the interrupt service.

6.6 Baud Rate Derivation

```
/* This is a simple code example which shows a way to derive the proper values for
the RCOR/TCOR and RBPR/TBPR register pairs for any baud rate. Routine is called with
the desired baud rate and master clock; global variables cor and bpr are set by the
routine. */

int          brp, cor;

compute_baud(clock, baud_rate)
double             clock;
double             baud_rate;
{

        double           cor_values[ ] = {8.0, 32.0, 128.0, 512.0, 2048.0, -1.0};
        int   i;

        for ( i = 0; cor_values[i] != -1; i++ )
        {
                   brp = (int) ((( clock / baud_rate) / cor_values[i]) + 0.5);
                   if (brp < 0xFF)
              {
                         cor = i;
                         bpr = brp;
                         break;
              }
        }
```

```
        return(0);
}
```

6.7 Baud Rate Tables

Table 22 through Table 26 indicate the values to be loaded into the RCOR/RBPR and TCOR/TBPR to set the designated baud rate when using five standard frequency crystals. Table 22 uses a 25-MHz frequency; Table 23 uses a 20.2752-MHz frequency, which yields near-perfect bit rates. Table 24 uses a 20-MHz frequency and shows error rates that are larger although still well within the limits set by the various standards covering asynchronous communications. Table 25 also uses another standard communications base frequency (18.432 MHz) that yields divisors with nearly zero errors overall. However, since this frequency is below 20 MHz, performance at the higher baud rates (76.8K and above) may be slightly lower and rates above 76.8K are *not* recommended. Table 26 shows divisors for the lowest recommended operating frequency, 16 MHz.

Note: It is not necessary that both the receiver and transmitter of a channel be programmed to the same baud rate; the CD1284 can send and receive at different rates on the same channel.

Table 22. Baud Rate Constants — CLK = 25 MHz

Baud Rate	R/TCOR[1]	R/TBPR (Hex)	Error
110	4	6F	0.02%
150	4	51	0.47%
300	3	A3	0.15%
600	3	51	0.47%
1200	2	A3	0.15%
2400	2	51	0.47%
4800	1	A3	0.15%
9600	1	51	0.47%
19200	0	A3	0.15%
38400	0	51	0.47%
56000	0	38	0.35%
57600	0	36	0.47%
64000	0	31	0.35%
76800	0	29	0.76%
115200	0	1B	0.47%
128000	0	18	1.70%
150000	0	15	0.80%

NOTE:
1. In this and the following tables, R/T is used as a register abbreviation indicating Receive/Transmit, followed by the register acronym.

Table 23. Baud Rate Constants — CLK = 20.2752 MHz

Baud Rate	RCOR/TCOR	RBPR/TBPR (Hex)	Error
110	4	5A	0.00%
150	4	42	0.00%
300	3	84	0.00%
600	3	42	0.00%
1200	2	84	0.00%
2400	2	42	0.00%
4800	1	84	0.00%
9600	1	42	0.00%
19200	0	84	0.00%
38400	0	42	0.00%
56000	0	2D	0.57%
57600	0	2C	0.00%
64000	0	28	1.00%
76800	0	21	0.00%
115200	0	16	0.00% (Not recommended at this CLK)
128000	0	14	1.01% (Not recommended at this CLK)
150000	0	11	0.62% (Not recommended at this CLK)

Table 24. Baud Rate Constants — CLK = 20.00 MHz (Sheet 1 of 2)

Baud Rate	RCOR/TCOR (Hex)	RBPR/TBPR	Error
110	4	59	0.25%
150	4	41	0.16%
300	3	82	0.16%
600	3	41	0.16%
1200	2	82	0.16%
2400	2	41	0.16%
4800	1	82	0.16%
9600	1	41	0.16%
19200	0	82	0.16%
38400	0	41	0.16%
56000	0	2D	0.79%
57600	0	2B	0.94%
64000	0	27	0.16%

Table 24. Baud Rate Constants — CLK = 20.00 MHz (Sheet 2 of 2)

Baud Rate	RCOR/TCOR (Hex)	RBPR/TBPR	Error
76800	0	21	1.36%
115200	0	16	1.36% (Not recommended at this CLK)
128000	0	14	2.40% (Not recommended at this CLK)

Table 25. Baud Rate Constants — CLK = 18.432 MHz

Baud Rate	RCOR/TCOR	RBPR/TBPR (Hex)	Error
110	4	52	0.22%
150	3	F0	0.00%
300	3	78	0.00%
600	2	F0	0.00%
1200	2	78	0.00%
1800	2	50	0.00%
2400	1	F0	0.00%
4800	1	78	0.00%
9600	0	F0	0.00%
19200	0	78	0.00%
38400	0	3C	0.00%
56000	0	29	0.35%
57600	0	28	0.00%
64000	0	24	0.00%
76800	0	1E	0.00%
115200	0	14	0.00% (Not recommended at this CLK)
128000	0	12	0.00% (Not recommended at this CLK)

Table 26. Baud Rate Constants — CLK = 16 MHz (Sheet 1 of 2)

Baud Rate	RCOR/TCOR (Hex)	RBPR/TBPR	Error
110	4	47	0.03%
150	3	D0	0.16%
300	3	68	0.16%
600	2	D0	0.16%
1200	2	68	0.16%
1800	2	45	0.16%
2400	1	D0	0.16%

Table 26. Baud Rate Constants — CLK = 16 MHz (Sheet 2 of 2)

Baud Rate	RCOR/TCOR (Hex)	RBPR/TBPR	Error
4800	1	68	0.16%
9600	0	D0	0.16%
19200	0	68	0.16%
38400	0	34	0.16%
56000	0	24	0.80%
57600	0	23	0.80%
64000	0	1F	0.80%
76800	0	1A	0.16% (Not recommended at this CLK)
115200	0	11	2.080% (Not recommended at this CLK)

6.8 ASCII Code Tables

6.8.1 Hexadecimal — Character

00	NUL	01	SOH	02	STX	03	ETX	04	EOT	05	ENQ	06	ACK	07	BEL
08	BS	09	HT	0A	NL	0B	VT	0C	NP	0D	CR	0E	SO	0F	SI
10	DLE	11	DC1	12	DC2	13	DC3	14	DC4	15	NAK	16	SYN	17	ETB
18	CAN	19	EM	1A	SUB	1B	ESC	1C	FS	1D	GS	1E	RS	1F	US
20	SP	21	!	22	“	23	#	24	$	25	%	26	&	27	‘
28	(	29	)	2A	*	2B	+	2C	,	2D	-	2E	.	2F	/
30	0	31	1	32	2	33	3	34	4	35	5	36	6	37	7
38	8	39	9	3A	:	3B	;	3C	<	3D	=	3E	>	3F	?
40	@	41	A	42	B	43	C	44	D	45	E	46	F	47	G
48	H	49	I	4A	J	4B	K	4C	L	4D	M	4E	N	4F	O
50	P	51	Q	52	R	53	S	54	T	55	U	56	V	57	W
58	X	59	Y	5A	Z	5B	[	5C	\	5D	]	5E	^	5F	_
60	~	61	a	62	b	63	c	64	d	65	e	66	f	67	g
68	h	69	i	6A	j	6B	k	6C	l	6D	m	6E	n	6F	o
70	p	71	q	72	r	73	s	74	t	75	u	76	v	77	w
78	x	79	y	7A	z	7B	{	7C	|	7D	}	7E	_	7F	DEL

6.8.2 Decimal — Character

0	NUL	1	SOH	2	STX	3	ETX	4	EOT	5	ENQ	6	ACK	7	BEL
8	BS	9	HT	10	NL	11	VT	12	13	13	CR	14	SO	15	SI
16	DLE	17	DC1	18	DC2	19	DC3	20	DC4	21	NAK	22	SYN	23	ETB
24	CAN	25	EM	26	SUB	27	ESC	28	FS	29	GS	30	RS	31	US
32	SP	33	!	34	“	35	#	36	$	37	%	38	&	39	‘
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_
96	~	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	|	125	}	126	_	127	DEL

7.0 Detailed Register Descriptions

This section presents a complete and detailed description of each register. Registers have two formats: 1) full eight bits, where the entire content defines a single function; 2) the register is a collection of bits, grouped singly or in multiples, defining a function. In the second format, the descriptions divide the register into its component parts and describe the bits individually. The order of register presentation corresponds to the register summary tables in Chapter 4.0.

7.1 Global Registers

7.1.1 Channel Access Register

Register Name: CAR						8-Bit Hex Address: 68	
Register Description: Channel Access						Default Value: XX	
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Poll	Poll	Poll	Poll	Poll	0	C1	C0

The CAR provides access to individual channels within the CD1284. The least-significant two bits of the register select one of the four channels. Before any operation that affects a channel, this register must be loaded so that channel registers are available to the host. Bit 2 must always be ‘0’. Bits 7:3 are not used except during Poll-mode operation (see Section 6.3 for details).

C1	C0	Channel Selected
0	0	Channel 0
0	1	Not used
1	0	Channel 2
1	1	Channel 3

7.1.2 Global Firmware Revision Code Register

Register Name: GFRCR						8-Bit Hex Address: 4F	
Register Description: Global Firmware Revision Code						Default Value: 25	
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Firmware Revision Code							

The GFRCR serves two purposes in the CD1284. First, it displays the revision number of the firmware in the chip. When a revision to the CD1284 is required, the revision number of the firmware is incremented by one. The revision code is 24 (hex) for the Revision D device, and 25 (hex) for the Revision E device.

Secondly, a system programmer can use this register to indicate when the internal processor completes reset procedures. This is done by a power-on reset (by the RESET* input) or a software global reset (by the reset command in the CCR). Immediately after the reset operation begins, the internal CPU clears the register. When complete, and the CD1284 is ready to accept host accesses, the register is loaded with the revision code.

7.1.3 General-Purpose I/O Direction Register

Register Name: GPDIR — 8-Bit Hex Address: 71
Register Description: General-Purpose I/O Direction — Default Value: 00
Access: Read/Write

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Dir 7	Dir 6	Dir 5	Dir 4	Dir 3	Dir 2	Dir 1	Dir 0

7.1.4 General-Purpose I/O Register

Register Name: GPIO — 8-Bit Hex Address: 70
Register Description: General-Purpose I/O — Default Value: 00
Access: Read/Write

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Data 7	Data 6	Data 5	Data 4	Data 3	Data 2	Data 1	Data 0

This pair of registers enables access and control of the general-purpose I/O port. The general-purpose I/O port provides a byte-wide general purpose set of signals that are individually direction programmable.

The GPIO register accesses the data port on pins 53–60 (G[7:0]) with Data 0 accessing GP[0], etc. The corresponding bit in the GPDIR register controls the direction of the associated signal; ‘1’ programs the signal as output and ‘0’ programs it as input. When writing to the GPIO register, ‘1’s and ‘0’s are reflected in their true states on the pins that are programmed as outputs. When reading from the GPIO register, bits programmed as inputs reflect the true state of the signal condition on those bits; bits programmed as output reflect the previously set state.

7.1.5 Modem Interrupting Channel Register

Register Name: MICR — 8-Bit Hex Address: 46
Register Description: Modem Interrupting Channel — Default Value: 00
Access: Read/Write

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
X	X	X	X	C1	C0	X	X

The MICR, RISR, and TICR indicate the serial channel number that is currently being serviced by an active acknowledge cycle (whether polled or interrupt). Bits 3:2 (C1 and C0) are only valid during the context of a channel service routine; at any other time, their state is undefined. Host system software uses these registers to determine the number of the channel that originated the particular service request (receive, transmit, or modem). The format of these registers is the same and the description is valid for each. The upper four bits and lower two bits are user-defined and can be set to any value desired. When the register is read, these bits are presented as defined by the user; C1 and C0 are set by the CD1284 to reflect the proper channel number.

Bit	Description
7:4	User defined.
3:2	**Channel X:** When these bits are set to the values shown below, the channel number is defined. C1 / C0 / Channel Number: 0 / 0 / Channel 0 0 / 1 / Undefined 1 / 0 / Channel 2 1 / 1 / Channel 3
1:0	User defined.

7.1.6 Modem Interrupt Register

Register Name: MIR
Register Description: Modem Interrupt
Access: Read/Write
8-Bit Hex Address: 69
Default Value: 08

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
MdIreq	Mdbusy	Mdunfair	0	1	0	ch[1]	ch[0]

The MIR, PIR, and TIR are used during Poll-mode operation of the CD1284. All three registers provide the same type of information for each of the three service requests. The functions of RxIreq, TxIreq, and MdIreq have identical meanings, as do the group Rxbusy, Txbusy, and Mdbusy and the group Rxunfair, Txunfair, and Mdunfair. The least-significant two bits indicate the number of the channel requesting service. Bits 4:2 are used internally by the CD1284 to set the context of the service-acknowledge cycle. See the description of Poll-mode operations in Chapter 5.0 for complete details.

Bit	Description
7	RxIreq, TxIreq, and MdIreq: These bits are set by the internal processor when service is required by a channel. The bits are a direct reflection of the inverse state of the SVCREQ* pins and they are the active-high output of the latch that drives the SVCREQ* pins. The bits can be scanned by the host to detect an active service request. These bits are cleared by the internal processor at the beginning of the service-acknowledge cycle (hardware-service acknowledge) or by the host software when the Poll-mode cycle is terminated.
6	Rxbusy, Txbusy, and Mdbusy: These bits are set by the internal processor and they remain set until the end of the service-acknowledge cycle is indicated by either a write to the EOSRR (hardware-service acknowledge), or cleared by the host software when the Poll-mode cycle is terminated. These bits signal the current state of the service-acknowledge cycle. When cleared, the internal processor knows that it can assert another service request of this type.

Bit	Description
5	Rxunfair, Txunfair, and Mdunfair: These bits are used by the internal processor to implement the Fair Share service request function. If this bit is set, the CD1284 does not assert another service request of this type until the bit is cleared by a pulse on the external SVCACK* pin. The unfair bits are forced to '0', disabling the Fair Share mechanism, by setting the Unfair bit in the PACR. These bits are not used in Poll mode.
4:2	These bits define the context of the current service-acknowledge cycle during Poll mode and are fixed by hardware within the CD1284. These bits must be replicated exactly when the register is copied to the CAR and is activating a service-acknowledge cycle. See the discussion of Poll-mode operation in Section 5.3 for a more detailed description.
1:0	**ch[1:0]:** These two bits encode the channel number of the requesting channel. During Poll-mode operation when the RIR, TIR, and MIR are copied into the CAR to start the service routine, ch[1:0] set the channel number that is serviced.

7.1.7 Parallel Interrupt Register

Register Name: PIR
Register Description: Parallel Interrupt
Access: Read/Write
8-Bit Hex Address: 61
Default Value: 00

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
PPIreq	PPort	Pipeline	0	0	0	0	0

The PIR is a modified version of the other interrupt registers (RIR, TIR, and MIR) that incorporates the unique differences between interrupt structures of the two major blocks of the CD1284. The Ireq bit (bit 7) has the identical function as the Ireq bits in the TIR, RIR, and MIR.

Bit	Description
7	**PPIreq:** The internal processor sets this bit to generate the external service request output. It is a direct reflection of the inverse state of the SVCREQP* pin; it is the active-high output of the latch that drives SVCREQP*. This bit can be scanned by the host to detect an active service request. The bit is cleared by the internal logic at the beginning of the hardware service-acknowledge cycle or by toggling the IntEn bit (PFCR[4]).
6:5	**PPort and Pipeline:** These two bits indicate which of the two functional blocks of the parallel port are requesting service. PPort set indicates that the parallel channel control state machine is the cause of the request; Pipeline set indicates that the data pipeline is requesting service. Both bits set indicates that both blocks are requesting service simultaneously.
4:0	**Reserved:** These bits always return '0' when read by the host. Do not modify.

7.1.8 Prescaler Period Register

Register Name: PPR
Register Description: Prescaler Period
Access: Read/Write
8-Bit Hex Address: 7E
Default Value: FF

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
8-bit Binary Value							

The PPR sets the divisor that generates the time period for CD1284 timer operations. It can be set to any value between 0 and 255 (x'FF). The PPR is clocked by the system clock prescaled (divided) by 512.

Note: This value does not have any effect on baud rate generation.

The time period generated by this register drives the receive timer and activates the 'no new data' and 'receive data timeout' interrupts. See the receiver operation discussion in Chapter 5.0 for a description of receiver timer functions.

7.1.9 Receive Interrupting Channel Register

Register Name: RICR Register Description: Receive Interrupting Channel Access: Read/Write						8-Bit Hex Address: 44 Default Value: 00	
7	6	5	4	3	2	1	0
X	X	X	X	C1	C0	X	X

See Section 7.1.5 on page 109, the description of the MICR, for details on the RICR.

7.1.10 Receive Interrupt Register

Register Name: RIR Register Description: Receive Interrupt Access: Read/Write						8-Bit Hex Address: 6B Default Value: 18	
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
RxIreq	Rxbusy	Rxunfair	1	1	0	ch[1]	ch[0]

See Section 7.1.6 on page 110, the description of the MIR, for details on the RIR.

7.1.11 Service Request Register

Register Name: SVRR Register Description: Service Request Access: Read only						8-Bit Hex Address: 67 Default Value: 00	
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
DMAREQ	ExtM	ExtT	ExtR	SRP	SRM	SRT	SRR

The SVRR reflects the inverse of the state of the service request pins (SVCREQR*, SVCREQT*, and SVCREQM*). Its primary use is in polled systems, and it allows system software to determine what, if any, service requests are pending.

Bit	Description
7	**DMA Request Status:** '1' indicates request pending.
6	**ExtM:** Reflects the current state of the external SVCREQM* signal.
5	**ExtT:** Reflects the current state of the external SVCREQT* signal.
4	**ExtR:** Reflects the current state of the external SVCREQR* signal.
3	**Service Request Parallel:** '1' indicates request pending.

Bit	Description
2	**Service Request Modem:** '1' indicates request pending.
1	**Service Request Transmit:** '1' indicates request pending.
0	**Service Request Receive:** '1' indicates request pending.

7.1.12 Transmit Interrupting Channel Register

Register Name: TICR Register Description: Transmit Interrupting Channel Access: Read/Write						8-Bit Hex Address: 45 Default Value: 00	
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
X	X	X	X	C1	C0	X	X

See Section 7.1.5 on page 109, the description of the MICR, for details on the TICR.

7.1.13 Transmit Interrupt Register

Register Name: TIR Register Description: Transmit Interrupt Access: Read/Write						8-Bit Hex Address: 6A Default Value: 10	
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
TxIreq	Txbusy	Txunfair	1	0	0	ch[1]	ch[0]

See Section 7.1.6 on page 110, the description of the MIR, for details on the TIR.

7.2 Virtual Registers

The CD1284 has two operational contexts:

Normal: Allows host access to most registers and any channel

Service-acknowledge: Allows host access to some registers specific to the channel requesting service.

This special set of registers is called Virtual because they are only available to host access and valid during this service-acknowledge context. At all other times, their contents are undefined and must not be written to by host software.

The use of Virtual registers and context switching allows the CD1284 to maintain all channel-specific information. To access the registers pertinent to the channel being serviced, it is not necessary for the host to make any changes to the device registers.

The service-acknowledge context can be entered in two ways: 1) by activating one of the SVCACK* input pins (hardware-activated); 2) by the host software when the contents of any one of TIR, RIR, MIR, or PIR are copied into the CAR during a Poll-mode acknowledge cycle. Chapter 5.0 discusses the differences between these two modes.

Virtual Registers — Serial

7.2.1 Modem Interrupt Status Register

Register Name: MISR							8-Bit Hex Address: 4C
Register Description: Modem Interrupt Status							Default Value: 00
Access: Read only							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
DSRch	CTSch	RIch	CDch	0	0	0	0

The MISR provides the status regarding a modem service request. If the modem-signal change detections (zero-to-one or one-to-zero transition) are enabled in MCOR1 or MCOR2, the change causes a service request and the changed signal is flagged in this register.

Bit	Description
7	**Data Set Ready Change:** An enabled transition on the Data Set Ready signal causes this bit to be set and a modem service request posted.
6	**Clear To Send Change:** An enabled transition on the Clear To Send signal causes this bit to be set and a modem service request posted.
5	**Ring Indicator Change:** An enabled transition on the Ring Indicator signal causes this bit to be set and a modem service request posted.
4	**Carrier Detect Change:** An enabled transition on the Carrier Detect signal causes this bit to be set and a modem service request posted.
3:0	These read-only bits always return ‘0’.

7.2.2 Modem Interrupt Vector Register

Register Name: MIVR							8-Bit Hex Address: 41
Register Description: Modem Interrupt Vector							Default Value: 00
Access: Read only							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
X	X	X	X	X	IT2	IT1	IT0

The value in this register is placed on the data bus, DB[7:0], when SVCACKM* is activated in response to an active SVCREQM*. See Section 7.4.6 on page 128 for more details on the LIVR.

IT2	IT1	IT0	Description
0	0	0	No modem interrupts.
0	0	1	Group 1: Modem signal change service request.
0	1	0	Invalid.
I	I	I	
1	1	1	

7.2.3 Parallel Interrupt Vector Register

Register Name: PIVR						8-Bit Hex Address: 40	
Register Description: Parallel Interrupt Vector						Default Value: 00	
Access: Read only							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
X	X	X	X	X	IT2	IT1	IT0

The value in this register is placed on the data bus, DB[7:0], when SVCACKP* is activated in response to an active SVCREQP*. See Section 7.4.6 on page 128 for more details on the LIVR.

IT2	IT1	IT0	Description
0	0	0	No parallel interrupt source is active.
0	0	1	Group 1: Modem signal change service request.
0	1	0	Invalid.
I	I	I	
0	1	1	
1	0	0	The parallel port state machine requests service.
1	0	1	The parallel port data pipeline requests service.
1	1	0	Both the parallel port state machine and the parallel port data pipeline request service.
1	1	1	Invalid.

7.2.4 Receive Data/Status Registers

Register Name: RDSR						8-Bit Hex Address: 62	
Register Description: Receive Data						Default Value: 00	
Access: Read only							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Received Character							

Register Name: RDSR						8-Bit Hex Address: 62	
Register Description: Receive Status						Default Value: 00	
Access: Read only							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Timeout	SC Det2	SC Det1	SC Det0	Break	PE	FE	OE

The Receive Data/Status register serves two purposes. During a serial receive-service acknowledge for good data, the RDSR provides access to the receive FIFO. The number of characters available in the FIFO is indicated by the RDCR, and is described in Section 7.5. Any number of characters, up to the value in the RDCR, can be read from the FIFO. All internal FIFO pointers are updated by the on-chip processor.

During a serial receive exception service acknowledge, the RDSR provides both the received character and the status that caused the exception condition. By definition, a receive exception service request has only one character available (multiple receive exceptions produce multiple service requests). The first read from the RDSR provides the exception status, and the second read

provides the character. It is not necessary to read either of these values. If the service acknowledge is terminated without reading the exception status and data from the RDSR, the internal processor updates the FIFO pointers as if the status/data were read. The same is true when only the status is read. Overrun errors are an exception to this (see table below).

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7</td><td>Timeout: If the service request enable for timeout is set, this bit indicates that no data has been received within the receive timeout period set by the RTPR after the last character was removed.</td></tr>
<tr><td>6:4</td><td>Special Character Detect: These three bits are encoded as follows:
<table>
<tr><th>SCDet2</th><th>SCDet1</th><th>SCDet0</th><th>Status</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>None detected.</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Special character 1 matched.</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Special character 2 matched.</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Special character 3 matched.</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Special character 4 matched.</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>Not used.</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>End-of-break detected.</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Range detect.</td></tr>
</table>
NOTE: No special character matching is performed if either a parity (PE) or framing (FE) error occur unless CMOE is enabled by COR5[5].</td></tr>
<tr><td>3</td><td>Break: Indicates that a break was detected.</td></tr>
<tr><td>2</td><td>Parity Error: Indicates that a character was received with parity other than that programmed in COR1.</td></tr>
<tr><td>1</td><td>Framing Error: Indicates that the character was received with a bad stop bit.</td></tr>
<tr><td>0</td><td>Overrun Error: This bit is set if new data is received, but there is no space available in the FIFO and Holding register. In this case, the character data is lost, and the overrun flag is applied to the last good data received before the overrun occurred. Thus, the character read on the subsequent read from the RDSR is good data and should not be discarded.</td></tr>
</table>

7.2.5 Receive Interrupt Vector Register

Register Name: RIVR — 8-Bit Hex Address: 43
Register Description: Receive Interrupt Vector — Default Value: 00
Access: Read only

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
X	X	X	X	X	IT2	IT1	IT0

The value in this register is placed on the data bus, DB[7:0], when SVCACKR* is activated in response to an active SVCREQR*. See Section 7.4.6 on page 128 for more details on the LIVR.

<table>
<tr><th>IT2</th><th>IT1</th><th>IT0</th><th>Description</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>No receive interrupt active.</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td rowspan="2">Invalid.</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
</table>

IT2	IT1	IT0	Description
0	1	1	Group 3: Received good data service request.
1	0	0	Invalid.
I	I	I	
1	1	0	
1	1	1	Group 3: Received exception data service request.

7.2.6 Transmit Data Register

Register Name: TDR						8-Bit Hex Address: 63	
Register Description: Transmit Data						Default Value: 00	
Access: Write only							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Transmit Character							

The transmit data register is the port for the host to write to the transmit FIFO. When a channel is being serviced for a transmit service request, the host can write up to 12 characters to this register. The transmit data register should only be written during the context of a transmit-service acknowledge. A write of data to this location at any other time yields unpredictable results.

7.2.7 Transmit Interrupt Vector Register

Register Name: TIVR						8-Bit Hex Address: 42	
Register Description: Transmit Interrupt Vector						Default Value: 00	
Access: Read only							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
X	X	X	X	X	IT2	IT1	IT0

The value in this register is placed on the data bus, DB[7:0], when SVCACKT* is activated in response to an active SVCREQT*. See Section 7.4.6 on page 128 for more details on the LIVR.

IT2	IT1	IT0	Description
0	0	0	No transmit interrupt active.
0	0	1	Invalid.
0	1	0	Group 2: Transmit data service request.
1	1	1	Invalid.
I	I	I	
1	1	1	

Virtual Registers — All

7.2.8 End of Service Request Register

Register Name: EOSRR							8-Bit Hex Address: 60
Register Description: End of Service Request							Default Value: XX
Access: Write only							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
X	X	X	X	X	X	X	X

The EOSRR is a dummy location used to signal the end of a hardware service-acknowledge procedure invoked by the activation of SVCACK*. The data pattern written is a ‘don’t care’ value. A write to this location causes the CD1284 to perform its internal switch out of the service-acknowledge context. This register is only used during a hardware-activated service acknowledge and must not be written during Poll-mode operation.

7.3 Channel Registers

Each of the four channels has a set of registers that control aspects of its operation. In the following register descriptions the register contents and offsets apply to any of the channels; the channel being accessed at any given time is controlled by the CAR. This is true even during a service-acknowledge context; the CAR points to the channel to be serviced, whether it was loaded by the host (during Poll-mode operation) or by the CD1284 itself (during a hardware-activated service acknowledge).

7.3.1 Channel Command Register

Register Name: CCR							8-Bit Hex Address: 05
Register Description: Channel Command							Default Value: 00
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Res Chan	COR Chg	Send SC	Chan Ctl	D3	D2	D1	D0

The CCR issues commands directly to the on-chip processor to control or change some channel and, in one case, global functions of the channel selected by the CAR. The upper four bits indicate which of four command types is being issued and the lower four bits are parameters to those commands. No more than one bit is ever set in the command type field. When the command is executed by the CD1284, it zeros out the CCR. Therefore, two consecutive commands must wait for the CCR to clear after the first is issued, before the second command is issued.

Note: The CCR is valid only for serial channels 2 and 3. Commands issued to the CCR location of the parallel channel (channel 0) or channel 1 are ignored by the MPU and have no effect on device operation. If the host needs to issue a full device reset, it must select either channel 2 or channel 3 before issuing the command.

7.3.1.1 Format 1 — Reset Channel Command

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Res Chan	0	0	0	0	0	FTF	Type

When bit 7 is set, one of three types of reset operations are initiated, based on the value of the least-significant two bits. Bit 0 sets the type of reset, either channel-only or full-chip, and bit 1 causes the FIFO of the selected channel to be flushed.

The two types of reset selected by bit 0 cause very different results. When bit 0 is ‘0’, the reset command effects only the selected channel. Resetting a channel disables both the receiver and transmitter, and all FIFOs are flushed (cleared). If bit 0 is ‘1’, a full-chip reset is initiated. This reset has the same results as a hardware reset caused by activation of RESET*: all channels are disabled, all FIFOs are flushed, and all control registers set to their power-on reset state.

The completion of the reset operation can be detected the same way as though a power-on or hardware reset had occurred: the GFRCR changes from zero to the value of the firmware revision. Note that at the start of the reset operation, the GFRCR is cleared, but it can take some time for this to occur. Host software should wait for the GFRCR to go to zero, and then wait for it to go non-zero to indicate that the reset operation is complete. The host can clear the GFRCR before issuing the reset command and then wait for it to become non-zero.

The FTF (flush serial transmit FIFO) command, bit 1, causes the serial transmit FIFO of the selected channel to be cleared and pointers reset to the empty state. Any data in the FIFO is lost.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7</td><td>This bit must always be ‘1’.</td></tr>
<tr><td>6:2</td><td>These bits must always be ‘0’.</td></tr>
<tr><td>1:0</td><td>These bits are encoded as:
<table>
<tr><th>FTF</th><th>Type</th><th>Function</th></tr>
<tr><td>0</td><td>0</td><td>Reset current channel.</td></tr>
<tr><td>0</td><td>1</td><td>Full CD1284 reset.</td></tr>
<tr><td>1</td><td>0</td><td>Flush serial transmit FIFO of current channel</td></tr>
<tr><td>1</td><td>1</td><td>Not used.</td></tr>
</table></td></tr>
</table>

7.3.1.2 Format 2 — Channel Option Register Change Command

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	COR Chg	0	0	COR3	COR2	COR1	0

Bit 6 – combined with any bits 3:1 – informs the MPU that a change occurred in one of the Channel Option registers, COR1, COR2, and/or COR3, respectively. It is permissible to indicate that more than one COR has changed.

This command exists so that changes in the CORs are noted by the MPU, allowing it to update its internal working register, since it keeps copies of the CORs in its own shadow registers.

Bit	Description
7	This bit must always be ‘0’.
6	This bit must always be ‘1’.
5:4	These bits must always be ‘0’.
3:1	These three bits are encoded as: COR3 / COR2 / COR1 / Encoding 0 / 0 / 0 / Not used. 0 / 0 / 1 / COR1 changed. 0 / 1 / 0 / COR2 changed. 0 / 1 / 1 / COR1 and COR2 changed. 1 / 0 / 0 / COR3 changed. 1 / 0 / 1 / COR3 and COR1 changed. 1 / 1 / 0 / COR3 and COR2 changed. 1 / 1 / 1 / COR1, COR2, and COR3 changed.
0	This bit must always be ‘0’.

7.3.1.3 Format 3 — Send Special Character Command

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
0	0	Send SC	0	0	SSPC2	SSPC1	SSPC0

This command causes one of the pre-programmed characters in the special character registers (SCHR1, SCHR2, SCHR3, and SCHR4) to be sent preemptively (applies to the serial channels only). The character sent is selected by the settings of bits 2 through 0. ‘Preemptively’ means that the special character is sent immediately following the character in the Transmitter Holding register; it does not wait until the FIFO empties. Once the special character is sent, transmission of any characters remaining in the FIFO proceeds normally.

Bit	Description
7:6	Must be ‘0’.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>5</td><td>Must be ‘1’.</td></tr>
<tr><td>4:3</td><td>Must be ‘0’.</td></tr>
<tr><td>2:0</td><td>These bits are encoded as:
<table>
<tr><th>SSPC2</th><th>SSPC1</th><th>SSPC0</th><th>Encoding</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Not used.</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Send special character 1.</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Send special character 2.</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Send special character 3.</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Send special character 4.</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td rowspan="3">Not used.</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table></td></tr>
</table>

7.3.1.4 Format 4 — Channel Control Command

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
0	0	0	Chan Ctl	XMT EN	XMT DIS	RCV EN	RCV DIS

This command activates or deactivates the serial transmitter and/or receiver of the selected channel, based on the values in bits 3 through 0. This command is issued when a channel is being started for the first time. Once a channel is in use, it can be started and stopped using this command. It is more efficient however, to use the appropriate SRER bit in the IER. Multiple control commands can be issued at the same time; for example, both the transmitter and receiver can be enabled by simultaneously setting both the XMT EN and RCV EN bits.

Issuing an enable/disable command does not affect any register programming of the selected channel. It does however, affect the state of transmit flow-control. Issuing a disable or enable command to a channel whose transmitter has been flow-controlled by a remote (see the TxIBE bit in COR2), restarts transmission and clears the TxFloff bit (CCSR[2]). This ability is provided so that the host can override remote-generated flow control.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7:5</td><td>Must be ‘0’.</td></tr>
<tr><td>4</td><td>Must be ‘1’.</td></tr>
<tr><td>3:0</td><td>Select channel enable/disable activity:
<table>
<tr><th>XMT EN</th><th>XMT DIS</th><th>RCV EN</th><th>RCV DIS</th><th>Encoding</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>Disable receiver.</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>Enable receiver.</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>Disable transmitter.</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>Enable transmitter.</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>Disable transmitter and receiver.</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>Disable transmitter; enable receiver.</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>Enable transmitter; disable receiver.</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td><td>Enable transmitter and receiver.</td></tr>
</table></td></tr>
</table>

7.3.2 Channel Control Status Register

Register Name: CCSR — 8-Bit Hex Address: 0B
Register Description: Channel Control Status — Default Value: 00
Access: Read only

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
RxEN	RxFloff	RxFlon	0	TxEN	TxFloff	TxFlon	0

The CCSR provides current receiver/transmitter status of the selected channel.

Bit	Description
7	**Receiver Enabled:** This bit is set when the receiver is enabled and cleared when it is disabled.
6	**Receiver Flow Off:** This bit indicates that the receiver has requested the remote to stop transmitting through the use of a send XOFF character by a send special character 2 command in the CCR. The bit is cleared when a send special character 1 (XON) command is issued; the channel is either enabled or disabled, or the channel is reset.
5	**Receiver Flow On:** When a send special character 1 (XON) command is issued by the CCR, this bit is set. This bit is cleared when one of three events has occurred, 1) the first non-flow control character is received, 2) the receiver is either enabled or disabled, 3) or the channel is reset.
4	**Reserved:** This bit returns ‘0’ when read.
3	**Transmitter Enabled:** This bit is set when the transmitter is enabled and cleared when it is disabled.

Bit	Description
2	**Transmitter Flow Off:** This bit indicates that the CD1284 has been requested to stop transmission by the remote (received in-band flow control character XOFF). The bit is cleared when the CD1284 requests to restart transmission (receives an XON character); the channel is either enabled or disabled, or the channel is reset.
1	**Transmitter Flow On:** This bit is set when the CD1284 requests to restart transmission (received an XON character). It is reset when transmission begins, when the channel is either enabled or disabled, or when the channel is reset.
0	**Reserved:** This bit returns '0' when read.

7.4 Channel Registers — Parallel Pipeline

The following five Channel Option registers control many aspects of CD1284 serial channel operation and enable special character processing features. COR4 and COR5 specifically enable the UNIX line discipline character handling functions.

7.4.1 Channel Option Register 1

Register Name: COR1 — 8-Bit Hex Address: 08
Register Description: Channel Option Register 1 — Default Value: 00
Access: Read/Write

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Parity	ParM1	ParM0	Ignore	Stop1	Stop0	ChL1	ChL0

Bit	Description
7	**Parity Type:** This bit selects the type of parity that is generated and checked if parity is enabled. '1' selects odd parity and '0' selects even parity.
6:5	**Parity Mode 1 and Parity Mode 0:** These bits define the parity operation for both the transmitter and receiver. The encoding is: **ParM1** / **ParM0** / **Function** 0 / 0 / No parity. 0 / 1 / Force parity (odd parity = force 1, even parity = force 0). 1 / 0 / Normal parity. 1 / 1 / Not used.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>4</td><td>Ignore Parity: If this bit is set, the CD1284 ignores the parity on all incoming characters, thus no receive exception service requests are generated if the parity is in error. If the bit is cleared, parity is evaluated.</td></tr>
<tr><td>3:2</td><td>Stop Bit Length: These two bits set the length, in bit times, of the Stop bit for each character.
<table>
<tr><th>Stop1</th><th>Stop0</th><th>Number of Stop Bits</th></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1.5</td></tr>
<tr><td>1</td><td>0</td><td>2</td></tr>
<tr><td>1</td><td>1</td><td>Not used.</td></tr>
</table></td></tr>
<tr><td>1:0</td><td>Character Length: ChL1 and ChL0 select the length of each character, in number of bits. The CD1284 receives and transmits the same length character, on a given channel, in the range of five to eight bits.
<table>
<tr><th>ChL1</th><th>ChL0</th><th>Character Length</th></tr>
<tr><td>0</td><td>0</td><td>5 bits</td></tr>
<tr><td>0</td><td>1</td><td>6 bits</td></tr>
<tr><td>1</td><td>0</td><td>7 bits</td></tr>
<tr><td>1</td><td>1</td><td>8 bits</td></tr>
</table></td></tr>
</table>

7.4.2 Channel Option Register 2

Register Description: COR2 — 8-Bit Hex Address: 09
Register Description: Channel Option Register 2 — Default Value: 00
Access: Read/Write

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
IXM	TxIBE	ETC	LLM	RLM	RtsAO	CtsAE	DsrAE

Bit	Description
7	**Implied XON mode:** This bit enables the automatic resumption of character transmission upon the reception of any character. This bit only has meaning if the transmitter is in Automatic In-band Flow-control mode as programmed by the TxIBE control bit. When this bit is reset and TxIBE is enabled, the reception of any character restarts character transmission.
6	**Enable Automatic In-band Transmit Flow Control:** This bit allows the CD1284 to examine error-free incoming characters looking for an XOFF character (as programmed by SCHR2), if the special character match function is enabled (COR3[4]). If a match occurs, transmission ceases after the current characters in the Transmitter Shift register and Transmitter Holding register are sent. Transmission resumes when an XON character (or any character, depending on the value of the IXM bit) is received or if a channel enable command is issued by the CCR.
5	**Embedded Transmit Command Enable:** If the ETC bit is set, the CD1284 examines characters in the transmit FIFO. If an embedded command is detected, it is processed. See the embedded transmit command description in Chapter 5.0 for details of valid commands.

Bit	Description
4	**Local Loopback Mode:** This bit enables local loopback of the channel. This mode is generally used during system diagnostics. If this bit is set, the transmitter is internally 'looped' back to the receiver. The TxD pin is set to the marking state. Data sent is immediately received by the receiver. No data appears on the TxD pin; data on the RxD pin is ignored.
3	**Remote Loopback Mode:** Remote loopback allows a remote system to test its serial data stream. If this function is enabled, the CD1284 internally connects its receiver to the transmitter. Any data received is immediately echoed back. This mode is enabled by setting RLM, and disabled by clearing RLM.
2	**Request To Send Automatic Output:** The CD1284 can automatically assert RTS when a channel is enabled (by transmit/receive enable command in the CCR) and there is data in the FIFO. When the channel is disabled or there is no more data to send (that is, in the FIFO or Holding and Shift registers), RTS* is negated. Setting RtsAO enables the function.
1	**Clear To Send Automatic Enable:** This bit enables the CTS* input to control transmitter operation. If CtsAE is set and CTS* is not asserted, character transmission does not proceed.
0	**Data Set Ready Automatic Enable:** This bit allows the DSR* input to control receiver operation. Setting DsrAE enables the function. When enabled and DSR* is deasserted, the CD1284 discards all received characters.

7.4.3 Channel Option Register 3

Register Name: COR3 — 8-Bit Hex Address: 0A
Register Description: Channel Option Register 3 — Default Value: 00
Access: Read/Write

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
SCDRNG	SCD34	FCT	SCD12	RxTh3	RxTh2	RxTh1	RxTh0

Note: The threshold for the parallel channel (channel 0) are set by the PFTR.

Bit	Description
7	**Special Character Detect Range:** This bit enables range checking on received characters. If the character falls between a lower range, set by the value stored in the SCRL register, and an upper range, set by the value stored in the SCRH register – inclusive, a receive exception service request is posted with the status indicating a range detect (RDSR bits SCDet2–SCDet0 = 111).
6	**Enable Special Character Detect on SCHR4 and SCHR3:** This bit controls whether or not the CD1284 performs a comparison on received characters against the values stored in SCHR4 and SCHR3. The comparison is enabled by this bit being '1'.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>5</td><td>Flow Control Transparency: This bit enables/disables the transparent response to flow control characters received by the CD1284. If set, received XON and XOFF characters are not placed in the FIFO for the host. If in-band flow control is enabled, the characters are acted upon. If this bit is not set, flow control characters are acted upon, placed in the receive FIFO, and the host is notified by a receive exception service request.</td></tr>
<tr><td>4</td><td>Enable Special Character Detect on SCHR2 and SCHR1: This bit controls whether or not the CD1284 compares received characters with the values stored in SCHR2 and SCHR1. ‘1’ enables compare. This bit must be set to enable automatic in-band flow control.</td></tr>
<tr><td>3:0</td><td>Serial Receive FIFO Threshold
<table>
<tr><th>RxTh3</th><th>RxTh2</th><th>RxTh1</th><th>RxTh0</th><th>Receiver FIFO Threshold</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>Not used.</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1 character</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>2 characters</td></tr>
<tr><td>•</td><td>•</td><td>•</td><td>•</td><td></td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td><td>11 characters</td></tr>
</table></td></tr>
</table>

7.4.4 Channel Option Register 4

Register Name: COR4 — 8-Bit Hex Address: 1E
Register Description: Channel Option Register 4 — Default Value: 00
Access: Read/Write

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
IGNCR	ICRNL	INLCR	IGNBRK	–BRKINT	PEH[2]	PEH[1]	PEH[0]

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7:5</td><td>Carriage Return (CR) and New Line (NL) Processing: These three bits define the way that the CD1284 processes received CR and NL characters (x'0D and x'0A). The following table shows the actions performed:
<table>
<tr><th>IGNCR</th><th>ICRNL</th><th>INLCR</th><th>Action</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>No action.</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Received NL changed to CR.</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Received CR changed to NL.</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Received CR changed to NL; NL changed to CR.</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Received CR discarded.</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>Received CR discarded; NL changed to CR.</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Received CR discarded.</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Received CR discarded; NL changed to CR.</td></tr>
</table></td></tr>
<tr><td>4:3</td><td>Break Processing: The CD1284 can handle received break characters in three ways:
<table>
<tr><th>IGNBRK</th><th>–BRKINT</th><th>Break Action</th></tr>
<tr><td>0</td><td>0</td><td>Received break generates an exception service request. End-of-Break also generates an exception service request if EBD is enabled in COR5.</td></tr>
<tr><td>0</td><td>1</td><td>Received break treated as a good NULL character.</td></tr>
<tr><td>1</td><td>0</td><td>Not used.</td></tr>
<tr><td>1</td><td>1</td><td>Received break discarded.</td></tr>
</table></td></tr>
<tr><td>2:0</td><td>Parity (P), Framing (F), and Overrun (O) Error Special Processing: As with break characters, the CD1284 can treat error characters in several different ways, if enabled:
<table>
<tr><th>PEH[2]</th><th>PEH[1]</th><th>PEH[0]</th><th>Action</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Received P/F/O error characters treated as exception data.</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Received P/F/O error characters treated as good data.</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Received P/F/O error characters discarded.</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Received P/F/O error characters replaced with good NULL characters.</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Received P/F/O error characters are replaced with the two character sequence x'FF-NULL-character. Good x'FF characters are replaced with the two character sequence x'FF-x'FF.</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td rowspan="3">Not used.</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table></td></tr>
</table>

7.4.5 Channel Option Register 5

Register Name: COR5						8-Bit Hex Address: 1F	
Register Description: Channel Option Register 5						Default Value: 00	
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
ISTRIP	LNE	CMOE	0	0	EBD	ONLCR	OCRNL

Bit	Description
7	**ISTRIP:** This bit enables stripping of the most-significant bit (bit 7) on all received characters. '1' enables the function.
6	**LNext Enable:** When this bit is set, characters following an LNext character (as programmed by the LNC register) are not processed as a special character.
5	**Character Matching on Error:** If this bit is set, character matching occurs on both good and error characters. If the bit is cleared, matching occurs on good characters only.
4:3	These bits must always be '0'.
2	**End of Break Detect:** If this bit is set, the CD1284 after detecting and reporting a line-break condition, searches for the end of a break and reports it by an exception service request with the End of Break status in the RDSR (see RDSR description Section 7.2.4 on page 115).
1:0	Carriage Return (CR) and New Line (NL) Processing – Transmit: These two bits define any actions taken on characters in the transmit data stream.

ONLCR	OCRNL	Action
0	0	No action.
0	1	Transmit CR changed to NL.
1	0	Transmit NL changed to CRNL.
1	1	Transmit CR changed to NL; NL changed to CRNL.

7.4.6 Local Interrupt Vector Register

Register Name: LIVR						8-Bit Hex Address: 18	
Register Description: Local Interrupt Vector						Default Value: 00	
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
X	X	X	X	X	IT2	IT1	IT0

The LIVR is used only during hardware-activated service-acknowledge cycles. Host software loads desired information into the most-significant five bits; the least-significant three bits are not used. When the CD1284 is setting up a service request, it overlays the five most-significant bits of the LIVR into appropriate interrupt vector register (RIVR, TIVR, PIVR, and MIVR) and sets the least-significant three bits as required for the service request vector type. (See RIVR, TIVR, PIVR, and MIVR descriptions). Refer to Section 7.7.5 on page 138 for a more detailed description of this register.

7.4.7 LNext Character Register

Register Name: LNC							8-Bit Hex Address: 24
Register Description: LNext Character							Default Value: 00
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
LNext Character							

This register defines the LNext character. If the LNext function is enabled (COR5[6]), the CD1284 examines received characters and compare them against this value. If a match occurs, this character and the following are placed in the FIFO without any special processing. In effect, the LNext function causes the CD1284 to ignore characters with special meaning, such as flow-control characters. There are two exceptions. If the character following the LNext character is either a break or an error character, LNext is placed in the FIFO, and the following character are treated as it normally would be for these error conditions.

7.5 Modem Change Option Registers

The CD1284 has two registers that control its response to changes on the modem input pins. It can be programmed to respond to the low-to-high transition, the high-to-low transition or both. In addition, the threshold at which the DTR signal is negated can be set by the DTRth3–DTRth0 bits in MCOR1.

7.5.1 Modem Change Option Register 1

Register Name: MCOR1							8-Bit Hex Address: 15
Register Description: Modem Change Option Register 1							Default Value: 00
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
DSRzd	CTSzd	RIzd	CDzd	DTRth3	DTRth2	DTRth1	DTRth0

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7:4</td><td>DSRzd, CTSzd, RIzd and CDzd: Each of these bits controls its corresponding input pin. If the bit is set, the function is enabled and transitions from one-to-zero (zeros detect) generate an SVCREQM* service request.</td></tr>
<tr><td>3:0</td><td>DTRth3 through DTRth0: These bits form a binary value to determine when the DTR output is negated (based on the number of characters in the receive FIFO). When the FIFO holds more characters than this value, DTR is negated, informing the remote to stop transmission. This value must be set to a value numerically larger than the value set for the receive FIFO threshold in COR3.
<table>
<tr><th>DTRth3</th><th>DTRth2</th><th>DTRth1</th><th>DTRth0</th><th>Number of Characters in FIFO</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>Automatic DTR mode disabled.</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1 character</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>2 characters</td></tr>
<tr><td>•</td><td>•</td><td>•</td><td>•</td><td></td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td><td>11 characters</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td><td>12 characters</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td rowspan="3">Not used.</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td></tr>
</table>
</td></tr>
</table>

7.5.2 Modem Change Option Register 2

Register Name: MCOR2 — 8-Bit Hex Address: 16
Register Description: Modem Change Option Register 2 — Default Value: 00
Access: Read/Write

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
DSRod	CTSod	RIod	CDod	0	0	0	0

Bit	Description
7:4	**DSRod, CTSod, RIod, CDod:** Each of these bits controls its corresponding input pin. If the bit is set, the function is enabled and transitions from ‘0’-to-‘1’ (ones detect) generate an SVCREQM* service request.
3:0	These bits are not used and must be ‘0’.

7.5.3 Modem Signal Value Register 1

Register Name: MSVR1 — 8-Bit Hex Address: 6C
Register Description: Modem Signal Value Register 1 — Default Value: XX
Access: Read/Write

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
DSR	CTS	RI	CD	0	0	0	RTS

7.5.4 Modem Signal Value Register 2

Register Name: MSVR2							8-Bit Hex Address: 6D
Register Description: Modem Signal Value Register 2							Default Value: XX
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
DSR	CTS	RI	CD	0	0	DTR	0

MSVR1 and MSVR2 provide information regarding the state of the modem input pins (DSR*, CTS*, RI*, and CD*) and allows control of the modem output pins (DTR* and RTS*). A write to any of the input bits has no effect. With the exception of the least-significant two bits, the registers reflect identical data. The two are provided as a convenience for control of the modem output pins. It is not necessary for host software to keep a copy of the current state of either when controlling the other. The actual signal level on the output is the inverse of the value placed in this register. For example, setting the DTR bit causes the DTR output to become active-low. The state of the modem input pins is also the inverse of the value in the corresponding bit in the registers.

7.5.5 Receive Baud Rate Period Register

Register Name: RBPR							8-Bit Hex Address: 78
Register Description: Receive Baud Rate Period							Default Value: 41
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Binary Divisor Value							

This register holds the baud rate divisor for the receiver. It is used in conjunction with the RCOR. This provides the clock, which is divided by this value. The time period produced must equal the value for one bit time of the receive data.

7.5.6 Receive Clock Option Register

Register Name: RCOR							8-Bit Hex Address: 7C
Register Description: Receive Clock Option							Default Value: 01
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
X	X	X	X	X	ClkSel2	ClkSel1	ClkSel0

The RCOR selects the clock source, which drives the RBPR. The value in ClkSel2–ClkSel0 selects one of five possible clocks generated from the master clock (CLK).

ClkSel2	ClkSel1	ClkSel0	Clock Selected
0	0	0	Clk0 (CLK ÷ 8)
0	0	1	Clk1 (CLK ÷ 32)
0	1	0	Clk2 (CLK ÷ 128)
0	1	1	Clk3 (CLK ÷ 512)
1	0	0	Clk4 (CLK ÷ 2048)

ClkSel2	ClkSel1	ClkSel0	Clock Selected
1	0	1	Not used.
1	1	0	
1	1	1	

7.5.7 Received Data Count Register

Register Name: RDCR — 8-Bit Hex Address: 0E
Register Description: Received Data Count — Default Value: 00
Access: Read only

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
0	0	0	0	CT3	CT2	CT1	CT0

The RDCR indicates the number of good characters currently in the serial received data FIFO. Host software can use this value as a loop counter when taking characters out of the FIFO. The value in this register is only valid during the context of a service request acknowledge. At other times, it may or may not give a true indication of the number of characters in the FIFO.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7:4</td><td>These bits must always be ‘0’.</td></tr>
<tr><td>3:0</td><td>Character Count 3:0: The encoding for these bits is:
<table>
<tr><th>CT3</th><th>CT2</th><th>CT1</th><th>CT0</th><th>Number of characters in FIFO</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>Not used.</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1 character</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>2 characters</td></tr>
<tr><td>•</td><td>•</td><td>•</td><td>•</td><td></td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td><td>11 characters</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td><td>12 characters</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td rowspan="3">Not used.</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td></tr>
</table></td></tr>
</table>

7.5.8 Receive Timeout Period Register

Register Name: RTPR							8-Bit Hex Address: 21
Register Description: Receive Timeout Period							Default Value: 00
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Binary Count Value							

The RTPR determines the time period used for the NNDT (no new data timeout) and the ‘no new data’ timeout. The timeout counter is loaded from this register whenever a new character is placed in – or the last character is removed from – the receive FIFO. The counter decrements on each tick of the prescaler counter (PPR). A service request is generated if the count reaches zero and:

- Either an NNDT if the FIFO is empty and the NNDT is enabled, or
- A Good Data service request is generated if there is data in the FIFO

In either case the timeout period has expired before the FIFO reaches the programmed threshold.

7.6 Special Character Registers

The four special character registers, SCHR1–SCHR4, contain the character patterns used for various character matching and flow-control functions. Each 8-bit character is right justified, that is, comparison occurs from right to left, and all bits are compared. Any unused bits must be ‘0’. SCHR1 and SCHR2 serve the additional function of defining the XON and XOFF characters, respectively, used for in-band flow control.

7.6.1 Special Character Register 1

Register Name: SCHR1							8-Bit Hex Address: 1A
Register Description: Special Character Register 1							Default Value: 00
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Special Character 1							

SCHR1 defines the XON character.

7.6.2 Special Character Register 2

Register Name: SCHR2							8-Bit Hex Address: 1B
Register Description: Special Character Register 2							Default Value: 00
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Special Character 2							

SCHR2 defines the XOFF character.

7.6.3 Special Character Register 3

Register Name: SCHR3							8-Bit Hex Address: 1C
Register Description: Special Character Register 3							Default Value: 00
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Special Character 3							

7.6.4 Special Character Register 4

Register Name: SCHR4							8-Bit Hex Address: 1D
Register Description: Special Character Register 4							Default Value: 00
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Special Character 4							

7.6.5 Received Character Range Detection

If enabled (by bit 7 of COR3), the CD1284 checks received characters to see if they fall within a range of values. SCRL and SCRH set the range and the checking occurs inclusive of the values programmed into these registers. If a received character is determined to be within the range, a special character detect exception service request is posted. When set to ‘111’, RDSR[6:4] indicate a range detect. Note that this range checking is performed in addition to the normal special character detection on SCHR4–SCHR1.

7.6.6 Special Character Range — High

Register Name: SCRH							8-Bit Hex Address: 23
Register Description: Special Character Range, high							Default Value: 00
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Character Range — high							

SCRH sets the upper inclusive value for range detection.

7.6.7 Special Character Range — Low

Register Name: SCRL							8-Bit Hex Address: 22
Register Description: Special Character Range, low							Default Value: 00
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Character Range — low							

SCRL sets the lower inclusive value for range detection.

7.6.8 Serial Service Request Enable Register

Register Name: SRER							8-Bit Hex Address: 06
Register Description: Serial Service Request Enable							Default Value: 00
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
MdmChg	0	0	RxData	0	TxRdy	TxEmpty	NNDT

This register enables the conditions that cause the CD1284, to post a service request by the SVRR and the SVCREQ* output pins, and applies to the serial channels only. Each of the individual enable bits control one type of service request.

Bit	Description
7	**Modem Change:** This bit enables the Modem Change service request. When this bit is ‘1’, any selected modem signal change conditions (as programmed by MCOR1 and MCOR2) cause a modem service request to be posted.
6:5	These bits must always be ‘0’.
4	**Receive Data Enable:** This bit enables the posting of receive service requests when characters have been received and either the FIFO reaches the programmed threshold (set by COR3) or the receive timeout period has expired.
3	This bit must always be ‘0’.
2:1	**Transmitter Ready and Transmitter Empty:** The transmitter can be enabled to post service requests on one of two conditions: either the FIFO is empty or the Transmitter Shift register is empty. TxRdy enables the service request on the condition that the FIFO is empty. In this case, there are still two characters available for transmission before the transmitter underruns (one in the Shift register and one in the Holding register). TxEmpty enables the service request on the condition that the Shift register is empty. The transmitter underruns due to the latency experienced between the time the service request is posted and the time the host can load the FIFO. Under normal operating conditions, TxEmpty is set and TxRdy reset when there is no more data to transmit and the host requires notification that the last character was sent before it can disable the transmitter.
0	**No New Data Timeout Enable:** This bit activates the optional exception service request when all data is removed from the FIFO and no new data has arrived after a preprogrammed delay period set by the value in the RTPR. The LIVR (or RIVR) indicates a receive exception in the IT2–IT0 vector bits. There is no data associated with this exception service request. RDSR[7] indicates that the service request is for an NNDT condition.

7.6.9 Transmit Baud Rate Period Register

Register Name: TBPR							8-Bit Hex Address: 72
Register Description: Transmit Baud Rate Period							Default Value: 41
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Binary Divisor Value							

This register holds the baud rate divisor for the transmitter and is used in conjunction with the TCOR. This provides the clock, which is divided by this value. The time period produced must equal the value for one bit time of the transmit data.

7.6.10 Transmit Clock Option Register

Register Name: TCOR							8-Bit Hex Address: 76
Register Description: Transmit Clock Option							Default Value: 01
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
X	X	X	X	X	ClkSel2	ClkSel1	ClkSel0

The TCOR selects the clock source which drives the TBPR. The value in ClkSel[2:0] selects one of five possible clocks generated from the master clock (CLK).

ClkSel2	ClkSel1	ClkSel0	Clock Selected
0	0	0	Clk0 (CLK ÷ 8)
0	0	1	Clk1 (CLK ÷ 32)
0	1	0	Clk2 (CLK ÷ 128)
0	1	1	Clk3 (CLK ÷ 512)
1	0	0	Clk4 (CLK ÷ 2048)
1	0	1	Not used.
1	1	0	
1	1	1	

7.7 Channel Registers — Parallel Pipeline

7.7.1 Data Error Register

Register Name: DER							8-Bit Hex Address: 33
Register Description: Data Error							Default Value: 00
Access: Read only							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
DMAwrerr	DMArderr	Bufwrerr	Bufrderr	HR1wrerr	HR1rderr	HR2wrerr	HR2rderr

The bits in this read-only register indicate read/write errors involving the DMABUF register and the Data Pipeline registers. The DataErr bit (PFSR[0]) is the logical OR of these eight Error Status bits.

Reading this register has no effect on the error status. A write to this register clears all the bits, which cannot be written by the user. Host software should clear this register (write x'00) after completing an error service-acknowledge procedure. This bit is provided primarily as an aid to driver software development. Data errors should never occur under normal circumstances.

This register is cleared during device reset.

Bit	Description
7	**DMA Write Error:** This bit is set if the DMA control logic has written to the DMA buffer when it already contains data. It indicates that an invalid DMA transfer cycle occurred (a DMAACK* without a corresponding DMAREQ*).
6	**DMA Read Error:** As with bit 7, this bit indicates that DMA logic has performed a read from the DMA buffer when there was no data in it. It indicates that an invalid DMA transfer cycle occurred.
5	**Buffer Write Error:** This bit indicates that a system write to the DMA buffer occurred while it still contained data.
4	**Buffer Read Error:** This bit indicates that a system read from the DMA buffer occurred while it was empty.
3	**Holding Register 1 Write Error:** This bit indicates that a system write to PFHR1 occurred while it still contained data.
2	**Holding Register 1 Read Error:** This bit indicates that a system read from PFHR1 occurred while it was empty.
1	**Holding Register 2 Write Error:** This bit indicates that a system write to PFHR2 occurred while it still contained data.
0	**Holding Register 2 Read Error:** This bit indicates that a system read from PFHR2 occurred while it was empty.

7.7.2 DMA Buffer Data Register — High

Register Name: DMABUFH — 8-Bit Hex Address: 30
Register Description: DMA Buffer Data Register, high — Default Value: 00
Access: Read/Write

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
15	14	13	12	11	10	9	8

7.7.3 DMA Buffer Data Register — Low

Register Name: DMABUFL — 8-Bit Hex Address: 30
Register Description: DMA Buffer Data Register, low — Default Value: 00
Access: Read/Write

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
7	6	5	4	3	2	1	0

This 16-bit data register is used to buffer DMA data transfers to and from the CD1284. Under normal operating conditions, this register is only accessed during a DMA data transfer cycle. If the DMAbufWe (PFCR[0]) is set to ‘1’ and DMAdir (PFCR[5]) is set to ‘1’, data may be transferred from the host to the FIFO by directly writing to the DMABUF. The data automatically moves forward into the FIFO through the Data Pipeline Holding registers. The user must ensure that the FIFO has sufficient free space to accept the data before writing into the DMABUF.

The BYTESWAP pin determines the order of byte transfer from this register into the data pipeline. If BYTESWAP is set to ‘1’, data transferred on DB[15:8] is the first byte transferred into the data pipeline and DB[7:0] is transferred second. If BYTESWAP is set to ‘0’ this sequence is reversed. The same applies during data read during DMA transfers: if BYTESWAP is set to ‘1’, data from the data pipeline moves to the upper byte of DMABUF, the next byte moves into the lower byte. Again, if BYTESWAP is set to ‘0’, this sequence is reversed.

These resisters can be read through DMA acknowledge or PIO cycles, however, the DMABUF registers can only be read when the DMAREQ* signal is active. If DMAREQ* is inactive, the DMABUF registers will be empty. DMAfull (HRSR[3]) indicates if the DMABUF register is empty when DMAREQ* is active.

7.7.4 Firmware Revision Code Holding Register Status Register

Register Name: HRSR — 8-Bit Hex Address: 34
Register Description: Holding Register Status — Default Value: 04
Access: Read only

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
HR1full	HR1tag	HR2full	HR2tag	DMAfull	DMAmpty	DMAact	Ctnot0

The HRSR is a read-only register that indicates current data pipeline status. This register is not directly set to any particular value by a device reset, but reflects the current state of bits in other registers.

Bit	Description
7:6	**Holding Register 1 Full and Holding Register 1 Tagged:** These two bits indicate status of PFHR1. Bit 7 indicates that the register contains data; bit 6 indicates that the data is tagged. Bits 7 and 6 can be set simultaneously.
5:4	**Holding Register 2 Full and Holding Register 2 Tagged:** These two bits indicate status of PFHR2. Bit 5 indicates that the register contains data; bit 4 indicates that the data is tagged. Bits 5 and 4 can be set simultaneously.
3:2	**DMA Buffer Full and DMA Buffer Empty:** These two bits indicate status of the DMA transfer buffer (DMA buffer). Bit 3 indicates that the register contains data; bit 2 indicates that it is empty.
1	**DMA Active:** When this bit is set, it indicates that the DMA handshake is active and a DMA service has been requested but is not yet complete (DMAREQ* active – waiting for DMAACK*).
0	**Count Not Zero:** This bit indicates that the RLE counter is not zero, thus run-length encoding/decoding is in progress.

7.7.5 Local Interrupt Vector Register

Register Name: LIVR — 8-Bit Hex Address: 18
Register Description: Local Interrupt Vector — Default Value: 00
Access: Read/Write

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
User-Defined Bits					IT2	IT1	IT0

This read/write register can be initialized to any desired value and, when read in the normal context (that is, not a service acknowledge context), the same value will be returned. The upper 5 bits are copied into the appropriate vector register (MIVR, PIVR, TIVR, or RIVR) when the corresponding SVCACK* signal is activated *and* an SVCREQ* of the same type is active. During this hardware-activated service acknowledge read cycle, the appropriate vector register (MIVR, PIVR, TIVR, or RIVR) is driven onto the data bus, DB[7:0].

Bits	Description
7:3	**User-defined Interrupt Vector:** Host software can use these five bits for any purpose appropriate to the application. In some cases, these bits might define the rest of a complete interrupt response vector (Motorola-type systems). In the case of daisy-chain systems made up of multiple CD1284s, these bits define the device number in the chain.
2:0	**Interrupt Vector Type Code:** These bits are read/writable in the normal context. These bits are ‘don’t cares’.

7.7.6 Parallel Auxiliary Control Register

Register Name: PACR
Register Description: Parallel Auxiliary Control
Access: Read/Write
8-Bit Hex Address: 3F
Default Value: 00

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
ShrtTen	ShrtStal	StaleOff	FIFOlock	ClearTO	0	AsyncDMA	Unfair

This register provides some special functions for the parallel data path and interrupt generation circuitry. The upper two bits change the basic timing of the timers associated with the data pipeline. Bit 5 can disable the stale data timer. Bit 0 overrides the Fair Share functions of the device (serial and parallel channels).

Bit	Description
7	**ShrtTen:** This function shortens the Prescaler count cycle that generates the internal 10-µs (based on a 25-MHz system clock) clock for the stale data counter. This bit is cleared by RESET*. If set, the 10-µs ‘ticks’ of the counter are generated every two CLKs; the normal period is one ‘tick’ every 250 CLKs.
6	**ShrtStal:** This function shortens the period of the stale data timer. The stale data timer includes a divide-by-10 prescaler; setting this bit bypasses the prescaler function thus causing the stale data timer to count on each 10-µs clock ‘tick’. If both ShrtTen and ShrtStal are set, the stale data timer counts on every other CLK.
5	**StaleOff:** If set, this bit masks off the Stale Status bit. The inverse of this bit is AND’ed with the stale state condition of the parallel channel to produce the stale status and disables OneChar and Stale as interrupt sources. StaleOff is provided primarily for test and development purposes if slow movement of data into the parallel port causes Stale and OneChar to always appear true.
4	**FIFOlock:** The FIFOlock bit causes the FIFO to stop accepting data from the parallel channel state machine. This action makes the FIFO appear full to the parallel port, thus causing it to enter the ‘busy’ state. This function is primarily intended for use in system testing to cause a timeout on the 1284 bus. Setting this bit in ECP Forward mode may cause a stall condition event 35 because event 36 does not occur until FIFOlock is cleared. The ECP mode host transfer recovery handshake sequence (from event 35 stall) is supported and the byte transit discarded as required by the specification. This bit does not provide an effective means to flow control the host.
3	**Clear Timeout:** This bit is a reset bit for the timeout status latch logic. When toggled by software, the timeout status in the PFSR is cleared; it may be left set to disable the Timeout status function. Note that if this bit is left set, the OneChar interrupt condition will never become true because the OneChar interrupt logic uses the timeout status to determine when the FIFO has become stale.

Bit	Description
2	**Reserved:** Must be '0.'
1	**AsyncDMA:** AsyncDMA causes the device to synchronize the DMAACK* signal to the internal clock (rising clock edge). This capability provides an asynchronous DMA interface for systems that cannot meet the set-up times required by the synchronous DMA logic. Refer to Chapter 8.0 for specific timing relationships between CLK and DMAACK* when AsyncDMA is enabled.
0	**Unfair:** This bit overrides the Fair Share function of the device. If this bit is set, the device posts service requests even if the service request is already asserted by an external device. The override is in effect for channels 2 and 3; Fair Share is not functional on the parallel service request. For applications where the three serial channel service request outputs are wire-OR'ed together, set Unfair so that an interrupt of one type does not prevent posting one of the other types (receive, transmit, and modem).

7.7.7 Parallel Channel Reset Register

Register Name: PCRR
Register Description: Parallel Channel Reset
Access: Read/Write
8-Bit Hex Address: 6C
Default Value: 00

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
0	0	0	0	0	0	0	PChReset

This register exists only in the Channel 0 register set and is in the equivalent address location as the MSVR register of the serial channels.

Bit	Description
7:1	**Reserved:** Must be '0'
0	**PChReset:** Setting this bit asserts the equivalent of a hardware power-on reset to the parallel channel, channel 0. If set by the host, it must be cleared to resume normal parallel channel operation. This hardware reset affects only the parallel channel and has no affect on other functions of the device.

7.7.8 Parallel FIFO Control Register

Register Name: PFCR
Register Description: Parallel FIFO Control
Access: Read/Write
8-Bit Hex Address: 31
Default Value: 00

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
FIFOres	DMAen	DMAdir	IntEn	RLEen	setTAG	ErrEn	DMAbufWe

This register controls overall function of the parallel FIFO. These functions include resetting (flushing) the FIFO, enabling DMA transfers, enabling host interrupts, run-length encoding, and so on. The host sets these bits according to the mode of operation required.

After hard reset (RESET* or a CCR command of x'81 in one of the two serial channels), this register is cleared to all zeros.

Bit	Description
7	**FIFO Reset:** This bit must be set together with the correct value of DMAdir to properly initialize the data pipeline and FIFO registers for data transfer or when a new data transfer direction is desired. Any data remaining in the FIFO is discarded. The FIFO remains in reset mode until this bit is cleared with a second register write operation.
6	**DMA Enable:** This bit must be set for DMA requests to move data to or from the FIFO to be made. When DMAen = 1, The PFQR quantity value is compared with the PFTR user-programmed threshold value. In Receive mode, if the threshold is equalled or exceeded, DMAREQ* is asserted and causes DMA data transfers of whole (2-byte) words from the FIFO by the data pipeline. In Transmit mode, if the amount of data in the FIFO is equal to or less than the threshold, DMAREQ* is asserted causing DMA data transfers of whole (2-byte) words to the FIFO by the data pipeline.
5	**DMA Direction:** This bit sets the direction of transfer between the parallel FIFO and system memory. If DMAdir = 1, the direction is transmit (system memory to the parallel FIFO); if it is '0', the direction is receive. The desired DMAdir value must be set together with FIFOres when initializing the FIFO logic for data transfer. Once a DMAdir value is set and the FIFOres is complete, that DMAdir selection must be maintained during any other changes to the control bits of the PFCR. ***Note:*** This bit sets the direction of the channel, even when DMA is not enabled. The proper direction must be set regardless of the DMAen bit.
4	**Interrupt Enable:** This is the master interrupt enable for the parallel channel. This bit must be set for any interrupts generated by the data pipeline, parallel port, or error status. In Poll-mode operation, host software toggles this bit to signal the completion of the service-acknowledge cycle. Toggling this bit updates the state of SVCREQP* and the PIR according to the current state of PCISR, DERR, and PFSR. For this reason, PCISR, DERR, and PFSR should be read and cleared at the end of the service routine to ensure that no requests were skipped. This is because an edge-sensitive interrupt controller may not detect a request active when the program returns from the service routine.
3	**RLE Enable:** The state of this bit enables RLE encoding/decoding for the direction defined by DMAdir. The RLEen bit effects the flow of data through the data pipeline in the transmit direction. Data flow into the FIFO is managed in such a way that PFHR1 and PFHR2 are kept full to permit evaluation of data sequences for possible compression. The effect is that following any data transfer while RLEen is set, the final 2 bytes written to the DMABUF register are kept in PFHR1 and PFHR2. To allow these bytes to be moved into the FIFO or to make room in PFHR1 for a tagged data transfer, RLEen must be '0' and both DMAen and DMAbufWe must be '0'.
2	**Set TAG:** This bit specifies that the next character written to the parallel channel by the PFHR1 register is to be tagged as an ECP or EPP special character (for a detailed explanation of the special handling of these characters, see Section 5.13). The setTAG bit is cleared by a write to PFHR1 thus, this bit must be set each time a tagged character is to be written.
1	**Error Interrupt Enable:** This bit enables a non-zero DataErr status to cause an interrupt if IntEn is also set.
0	**DMA Buffer Write Enable:** This bit must be set to enable host writes to the DMABUF register. It also enables the FIFO data pipeline to empty the DMABUF register when written to by the host system. In this case, the system writes to the DMA buffer (without DMA transfers) providing a low-performance alternative to DMA transfers.

7.7.9 Parallel FIFO Empty Pointer Register

Register Name: PFEP — 8-Bit Hex Address: 39
Register Description: Parallel FIFO Empty Pointer — Default Value: 00
Access: Read/Write

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
0	0	6-bit binary FIFO Pointer Value					

This register holds the internal empty location pointer of the FIFO. It identifies the location in the FIFO from which the next byte of data transfers from the FIFO.

The PFEP is cleared by a device or FIFO reset.

7.7.10 Parallel FIFO Fill Pointer Register

Register Name: PFFP							8-Bit Hex Address: 38
Register Description: Parallel FIFO Fill Pointer							Default Value: 00
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
0	0	6-bit binary FIFO Pointer Value					

This register holds the internal fill location pointer of the FIFO. It identifies the location in the FIFO to receive the next data byte from the pipeline.

The PFFP is cleared by a device or FIFO reset.

7.7.11 Parallel FIFO Holding Register 1

Register Name: PFHR1							8-Bit Hex Address: 35
Register Description: Parallel FIFO Holding Register 1							Default Value: 00
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
8-bit Character Data							

7.7.12 Parallel FIFO Holding Register 2

Register Name: PFHR2							8-Bit Hex Address: 36
Register Description: Parallel FIFO Holding Register 2							Default Value: 00
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
8-bit Character Data							

These two 1-byte registers provide a data pipeline between the FIFO and DMA buffer. Data always flows into PFHR1 first, then to PFHR2, and finally, either to the FIFO or the DMABUF register. The flow is to the FIFO if DMAdir is ‘1’ and, from the FIFO if DMAdir is ‘0’. The pipeline and the holding registers support ‘tagged’ data for complete support of ECP Parallel Port mode. Tagged data is either an address or a run-length code.

If RLEen (PFCR[3]) is set, in the receive direction, run-length codes are captured in the RLCR for decompression of received data. ECP address codes are recognized and pass into the PFHR1–PFHR2 pipeline. The presence of an ECP address interrupts DMA flow and causes an interrupt to the host so it can remove the tagged data from the pipeline by reading either PFHR2 or PFHR1.

In the transmit direction, the host can introduce ECP address (tagged) data or run-length codes for precompressed data by setting the SetTAG bit (PFCR[2]) and writing the byte to be tagged to PFHR1. For each tagged data transfer, the SetTAG bit must be set prior to writing to PFHR1. To perform a tagged data transfer, the automatic DMA function must be disabled prior to the transfer (set DMAen = 0). This can be done at the same time that SetTAG is set to ‘1’.

These registers are cleared by a device or FIFO reset and marked as empty in HRSR. Any tagged status is also cleared.

7.7.13 Parallel FIFO Quantity Register

Register Name: PFQR							8-Bit Hex Address: 3A
Register Description: Parallel FIFO Quantity							Default Value: 00
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Data or Space Available in FIFO — Max 0x40							

This register maintains the quantity (or count) of either data bytes or space available in the parallel FIFO. In the receive direction (DMAdir = 0), PFQR counts data characters in the FIFO. In the transmit direction (DMAdir = 1), PFQR counts space available in the FIFO for additional characters to transmit. FIFOres, together with the value of DMAdir, initialize PFQR to either x'00 (receive) or x'40 (transmit).

In either case, the PFQR indicates only the quantity of data or space available in the FIFO, and does not include the data pipeline registers.

7.7.14 Parallel FIFO Status Register

Register Name: PFSR							8-Bit Hex Address: 32
Register Description: Parallel FIFO Status							Default Value: 40
Access: Read only							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
FFfull	FFempty	Timeout	HRtag	HRdata	Stale	OneChar	DataErr

This read-only register provides the current FIFO and data pipeline status. Host software should examine these bits in response to pipeline interrupts or polling operations.

This register is not directly cleared by reset, but the individual bits reflect the status of other registers.

Bit	Description
7	**Parallel FIFO is Full:** If this bit is set, it indicates that the parallel FIFO is full.
6	**Parallel FIFO is Empty:** If this bit is set, the parallel FIFO is empty.
5	**Timeout:** This bit is set when Stale goes from false to true. In the receive direction, Timeout is delayed until the FIFO is empty and all DMA cycles are complete (PFHR2 may or may not be full). Timeout is a pipeline interrupt condition and must be cleared manually by the CPU. This is done by toggling ClrTO (PACR[3]) or by a FIFO reset in PFCR.
4	**Holding Register Tag:** This bit indicates that a tagged character is in either PFHR1, PFHR2, or both. If enabled, this bit being set causes a host interrupt to be generated. The host should examine the HRSR to determine the exact cause(s) of this bit being set.
3	**Holding Register Data:** If this bit is set, it indicates that either PFHR1, PFHR2, or both contain data.

Bit	Description (Continued)
2	**Stale:** This bit is set when the stale data timer expires (see the description of SDTPR). If a single byte remains in the data pipeline when this bit is set, a host interrupt is generated, the OneChar bit is set, and new data entering the FIFO does not move into PFHR1 until PFHR2 empties. If two or more bytes remain in the pipeline when this bit is set, a host interrupt is *not* generated, however, a DMA request is generated if enabled.
1	**One Character:** In the receive direction, this bit set indicates that the FIFO is empty and stale, and one character remains in PFHR2. This condition occurs if an odd number of bytes is transferred by the parallel interface. Since DMA cycles only move even numbers of bytes (words) and odd transfers leave one byte remaining, host software must remove this character outside of DMA transfer cycles.
0	**Data Error:** If this bit is set, it indicates that one or more of the bits in the DER are set.

7.7.15 Parallel FIFO Threshold Register

Register Name: PFTR — 8-Bit Hex Address: 3B
Register Description: Parallel FIFO Threshold — Default Value: 00
Access: Read/Write

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
0	DMA Transfer Threshold						

This register sets the FIFO threshold for initiating DMA requests for data transfer. The value is expressed in bytes. Whenever DMAen is true, regular comparisons are made between the PFQR and the PFTR. If the value in the PFQR is greater than or equal to the threshold, the DMA request logic becomes active and remains active until the FIFO is essentially filled or emptied. An odd character or space in the FIFO can remain.

In the receive direction, the Holding register pipeline (PFHR1 and PFHR2) are kept filled, so that tagged data (for example, ECP mode addresses) can be detected and passed to the host by an interrupt. For example, if the FIFO and data pipeline are initialized for receive, and 40 hex bytes are placed into the FIFO from the parallel port, the first two of those bytes automatically are placed in the Pipeline registers. If the PFTR were programmed to x'40 bytes, x'42 bytes must arrive to trigger a DMA transfer.

PFTR is cleared by device reset; it is not cleared by FIFOres.

7.7.16 Run Length Count Register

Register Name: RLCR — 8-Bit Hex Address: 37
Register Description: Run Length Count — Default Value: 00
Access: Read/Write

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
0	7-bit Unsigned Binary Count						

This register works with the Holding registers (PFHR1 and PFHR2) to perform run-length encoding and decoding when RLEen is set (PFCR[3]). The parallel port must be in ECP mode; in other modes, run-length encoding does not occur.

In the transmit direction, strings of three or more identical characters are recognized and compressed. The running count of identical characters is kept in the RLCR. Once the sequence is broken by a different character or the end of the transmit burst transfer, the count and a single copy of the duplicated character are put in the FIFO.

In the receive direction, run-length codes can be received from the remote device. These codes are recognized ‘on the fly’ as data flows from the FIFO through the holding register pipeline. A run-length code is diverted to the RLCR. The subsequent character from the FIFO is duplicated (held in PFHR1) while the RLCR decrements. Once the RLCR reaches ‘0’, normal pipeline data movement resumes. If run-length codes are being received by the parallel port but RLEen is not set, the codes enter PFHR1 and PFHR2 as tagged data and cause interrupts to the host. The host must read the tagged Holding register directly to remove the character from the pipeline and clear the tag.

This register is cleared by a device or FIFO reset.

7.7.17 Stale Data Timer Count Register

Register Name: SDTCR — 8-Bit Hex Address: 3D
Register Description: Stale Data Timer Count — Default Value: 00
Access: Read/Write

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
8-bit Stale Data Timer Count							

This register determines the period that signals stale data in the FIFO. The timer is used only in the receive direction. Each time a new character is placed in the FIFO from the parallel port, the SDTCR is reloaded from the SDTPR, and down-counting begins at the ‘tick’ rate. If the counter reaches ‘0’, the Stale bit (PFSR[2]) is set. If the amount of data available is greater than or equal to one word, a DMA request is made to move all remaining whole words to the host with a DMA transfer. Once the DMA transfer is complete, a single remaining character causes an interrupt to the host to remove the character by reading PFHR2.

This register is cleared by a device or FIFO reset. Clearing it causes the Stale bit (PFSR[2]) to become true.

7.7.18 Stale Data Timer Period Register

Register Name: SDTPR — 8-Bit Hex Address: 3C
Register Description: Stale Data Timer Period — Default Value: 00
Access: Read/Write

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
8-bit Stale Data Timeout Value							

This register provides a user-defined period value for use as the timeout value of the stale data timer (see SDTCR).

With a 25-MHz CLK input to the device, the resolution of this timer is 0.1 ms (with a maximum value of 25.5 ms). The 25-MHz clock is divided by 250 to produce a 10-µs intermediate clock for this timer. A fixed, divide-by-ten prescaler produces 0.1-ms ‘ticks’ to the stale data timer. To ensure accuracy for small timeout values, the prescaler is reset each time the stale data timer is reloaded. (A user selection of 0.1-ms timeout results in a time delay between 0.09 and 0.1 ms.)

The SDTPR is cleared by a device reset.

7.8 Channel Registers — Parallel Port

7.8.1 EPP Address Register

Register Name: EAR							8-Bit Hex Address: 25
Register Description: EPP Address							Default Value: 00
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
8-bit Binary Value							

This register is only used during EPP mode.

The CD1284 deposits the value obtained during an EPP address write command in this register. The CD1284 provides this value in response to an EPP address read command.

7.8.2 Host Timeout Value Register

Register Name: HTVR							8-Bit Hex Address: 24
Register Description: Host Timeout Value							Default Value: FF
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
HTVR[7]	HTVR[6]	HTVR[5]	HTVR[4]	HTVR[3]	HTVR[2]	HTVR[1]	HTVR[0]

This register holds the 8-bit value used to set the Host timeout period. The HTVR is an unsigned, binary value. The reset state of this register is ‘0xFF’.

A function missing in Revision C and earlier devices is an on-chip timer to indicate that the remote host has not responded in a specified time period. The Host timeout is defined in the IEEE STD 1284 specification as a period of one second.

Revision D and newer devices add a user-programmable timer to provide a timeout if the remote host does not respond to specific parallel port transactions. The timer is started by the parallel port state machine each time it starts a sequence requiring a host response. Activation of the timer is automatic and an interrupt is generated to the local host CPU if the timer expires before the remote host responds.

Note: Users familiar with the IEEE specification note that the events that start the timer cause the peripheral device to wait for a remote host-generated event. For example, during the negotiation sequence after event 2, the peripheral waits for event 3 – a host-generated event. If the host does not respond and moves the negotiation sequence to event 4 within one second, the peripheral enters the ‘host timeout’ condition.

The timer is a 14-bit counter clocked by the system clock (CLK) prescaled (divided) by 2048. Then the 8-bit HTVR (address offset 0x24) is programmed and compared with the most-significant 8 bits of the 14-bit counter. Each time the parallel port executes an event requiring a host response, the

14-bit counter is started (from 0x00). It counts up until either the expected event occurs or the count matches the value in HTVR. If a match occurs, a timeout condition exists. The HTVR need only be loaded once, typically during device initialization.

The value placed in HTVR yields an approximate one second count time, based on the value of the input CLK. For example, if the system clock driving the device is 25 MHz, the HTVR should be loaded with 0xC0. The following equation provides an example.

$$\frac{25MHz}{2048} = 12207_{10} = 2FAF_{16}$$

The computed value is rounded up to the next largest whole hex value, in this case ‘0x3000’. Load the HTVR with the most-significant 8 bits of this value, left-shifted two places since HTVR is a 14-bit counter. This results in a value of ‘0xC0’. For 20 MHz, the value is computed to be ‘0x9C’; for 16 MHz, the value is ‘0x7C’; values for other clocks can be easily computed in the same manner. At reset, the HTVR defaults to a value of ‘0xFF’; this prevents the extremely short timeouts that occur if the register is cleared at device reset and is not initialized.

A timeout causes a negotiation status change interrupt. This status is displayed as ‘0x22’ in the NSR (NSR[5] and the code for return to Compatibility mode – ‘0010’ – in the result code field). When Compatibility mode is reentered, the port control state machine waits in a locked state until signals on the parallel port return to normal Compatibility mode conditions.

For debug purposes, disable the host timeout timer by setting PCR[3:2] (HTmrTst[1:0]). In this case, no timeouts occur and the link can hang indefinitely while waiting for a host-generated event.

7.8.3 Input Value Register

Register Name: IVR						8-Bit Hex Address: 2E	
Register Description: Input Value						Default Value: XX	
Access: Read only							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
0	0	0	0	A1284	nInit	HstBsy	HstClk

This register always shows the current state of the external handshake pins.

Bit	Description
7:4	These bits are not used and return ‘0’ when read.
3	A1284
2	**nInit** (low active Init input)
1	**HstBsy** (host busy)
0	**HstClk** (host clock)

7.8.4 Manual Data Register

Register Name: MDR							8-Bit Hex Address: 21
Register Description: Manual Data							Default Value: 00
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
8-bit Binary Data							

This read/write register can read the state of the PD[7:0] signals in any mode. If the ManMd bit (PCR[7]) and the MMDir and ManOE bits (PCR[1:0]) are set, then the value written into this register is driven onto the PD[7:0] signals.

7.8.5 Negotiation Enable Register

Register Name: NER							8-Bit Hex Address: 28
Register Description: Negotiation Enable							Default Value: 00
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
0	RID	0	EPP	RLE	ECP	RVB	RVN

Each bit set along with EICR (PCR[6]) allows the CD1284 to engage in IEEE STD 1284 negotiations and move into the corresponding protocol. It is assumed that the peripheral host software responds to a request for slave ID and is able to send an ID string in any supported protocol. In response to an ID request, the CD1284 does not provide a method of storing and automatically sending an ID string. Note that the EPP protocol does not have provision for slave ID requests.

Bit	Description
7	**Reserved:** This read-only bit is always ‘0’.
6	Request Slave ID
5	**Reserved:** This bit must always be ‘0’.
4	EPP Mode Enable
3	Run Length Encoding in ECP Mode Enable
2	ECP Mode Enable
1	Reverse Byte Mode Enable
0	Reverse Nibble Mode Enable

7.8.6 Negotiation Status Register

Register Name: NSR							8-Bit Hex Address: 29
Register Description: Negotiation Status							Default Value: 00
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
NegOK	NegFI	HostTO	ImedTerm	4-bit Negotiation Result Code			

The results of negotiation attempts are stored in this register.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7</td><td>Negotiation OK: The state of this bit indicates that the negotiation was successful.</td></tr>
<tr><td>6</td><td>Negotiation Failed: The state of this bit indicates that the negotiation failed. The result code indicates which mode was attempted</td></tr>
<tr><td>5</td><td>Host Timeout: This bit indicates that a host timeout occurred on the parallel channel. The accompanying 4-bit result code indicates that the link has returned to Compatibility mode (x02). See the description of HTVR in Section 7.8.2 on page 146.</td></tr>
<tr><td>4</td><td>Immediate Termination: This bit indicates that the A1284 signal has unexpectedly gone inactive as a result of an immediate termination from the host and the interface and has reentered Compatibility mode. The 4-bit negotiation result code should indicate which mode was terminated.</td></tr>
<tr><td>3:0</td><td>The lower 4 bits of this register contain a result code that shows the current mode. The following table shows the encoding of the result code.
<table>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>Compatible mode — no negotiation.</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>Failed negotiation.</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>Compatible mode — termination of a 1284 mode.</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td rowspan="3">Reserved.</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>EPP mode.</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>Reserved.</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>Reverse Nibble mode.</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>Reverse Nibble mode — ID request.</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td><td>Reverse Byte mode.</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td><td>Reverse Byte mode — ID request.</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td><td>ECP mode without RLE.</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>ECP mode without RLE — ID request.</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>ECP mode with RLE.</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>ECP mode with RLE — ID request.</td></tr>
</table>
</td></tr>
</table>

Any change in the mode of the parallel port is reported to the peripheral host by interrupt if the NegCh bit (PCIER[5]) is set; host software then reads the NSR to determine the current status and condition. Once the host has read the NSR status resulting from the current negotiation, it should clear the register in preparation for additional negotiation cycles. The NSR can be cleared by writing any value.

7.8.7 Ones Detect Register

Register Name: ODR
Register Description: Ones Detect
Access: Read/Write
8-Bit Hex Address: 2D
Default Value: 00

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
0	0	0	0	A1284	nInit	HstBsy	HstClk

Setting the bits in this register enables the CD1284 to generate an interrupt – if SigCh (PCIER[4]) is set – when the selected signal changes from low-to-high (rising edge). Bits 7:4 are reserved and must be written as zeros; they return zero when read. The settings in this register have no effect (that is, a SigCh interrupt is not generated) unless the device is in Manual mode.

7.8.8 Output Value Register

Register Name: OVR — 8-Bit Hex Address: 2B
Register Description: Output Value — Default Value: 48
Access: Write only

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
PerBsy	PerClk	AkDaRq	xFlag	nDatAv	0	0	0

This register controls output signals. In Manual mode, all signals are controlled by these register settings. In Compatibility and EPP modes, PerBsy and PerClk are controlled by the internal parallel port state machine, while AkDaRq, xFlag, and nDatAv are controlled by this register. In ECP mode, the settings in this register have no effect.

Bit	Description
7:6	**Peripheral Busy and Peripheral Clock:** User-controlled in Manual mode only.
5	**Acknowledge Data Request:** In Compatible mode, this signal is the PError (Peripheral Error) signal. In EPP mode, this signal is auxiliary and is a user-defined signal (USER 1).
4	**XFlag:** In Compatible mode, this signal is the SELCT (Select) signal. In EPP mode, this signal is auxiliary and is a user-defined signal (USER 2).
3	**Negative-true Data Available:** In Compatible mode, this signal is the nFault (negative-true fault) signal. In EPP mode, this signal is auxiliary and is a user-defined signal (USER 3).
2:0	**Reserved**: These bits must be written as ‘0’.

7.8.9 Parallel Channel Interrupt Enable Register

Register Name: PCIER — 8-Bit Hex Address: 22
Register Description: Parallel Channel Interrupt Enable — Default Value: 00
Access: Read/Write

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 21*	*Bit*	*Bit*
0	TimEn	NegCh	SigCh	EPPAW	DirCh	IDReq	nINIT

7.8.10 Parallel Channel Interrupt Status Register

Register Name: PCISR — 8-Bit Hex Address: 23
Register Description: Parallel Channel Interrupt Status — Default Value: 00
Access: Read/Write

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
0	TimOvr	NegCh	SigCh	EPPAW	DirCh	IDReq	nINIT

PCIER and PCISR provide control and status of interrupts generated by the parallel channel control state machine. They have the same bit definitions. Each bit in the PCIER enables the interrupt of the same type in the PCISR. A write of any value to the PCISR in response to an interrupt request causes it to clear and the interrupt request is removed.

Bit	Description
7	This bit must always be ‘0’
6	**Timer Enable and Timer Over:** These two bits are for factory test purposes only and should never be written.
5	**Negotiation Change:** The state of this bit indicates that a change occurred in the negotiation status of the port. The NSR indicates the new status of the parallel port.
4	**Signal Change Enable:** This enable instructs the parallel port to generate an interrupt when any of the signals specified by the ZDR or ODR change state as programmed. This interrupt is only generated during Manual mode, however, it *cannot* be cleared by terminating Manual mode.
3	**EPPAW:** The state of this bit indicates that the remote master has written an EPP address to the CD1284. The new EPP address value is placed in the EAR.
2	**Direction Change:** This bit indicates that the host-side parallel port changed the direction of the interface. Generally, this is in response to a request made by the CD1284 through the RevRq bit (SCR[0]). DirCh indicates that the direction was reversed through the defined protocol and the CD1284 can now send data to the master.
1	**ID Request:** The state of this bit indicates that the host has requested that the CD1284 send its ID data string. The peripheral host sends the appropriate ID string (this is application dependent).
0	**nINIT:** This interrupt is generated when an nINIT pulse is received while in Compatibility mode. The interrupt occurs on the leading edge of the nINIT pulse.

7.8.11 Parallel Configuration Register

Register Name: PCR						8-Bit Hex Address: 20	
Register Description: Parallel Configuration						Default Value: 00	
Access: R/W							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
ManMd	E1284	ETxfr	Ig_SEL	HTmrTst[1]	HTmrTst[0]	MMDir	ManOE

This register controls the overall configuration of the parallel port, each of which is described in IEEE 1284 format below.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7:5</td><td>Mode Control: These three bits control the type of transfer desired and whether or not it is enabled to do so.
The ManMd bit selects Manual mode, which allows the user direct control over all parallel data and parallel port control signals. MMDir controls the direction of the MDR (Manual Data register), and ManOE is the output enable when MMDir = 1 (output mode).
E1284 allows the parallel port to engage in IEEE 1284 negotiations; ETxfr enables data transfers. The ETxfr enable is only used for data transfers. EPP address read and write functions do not require that the ETxfr bit be set.
<table>
<tr><th>ManMd</th><th>E1284</th><th>Etxfr</th><th>Mode</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Compatibility mode; transfers disabled.</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Compatibility mode; transfers enabled.</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>IEEE 1284 negotiation; transfers disabled.</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>IEEE 1284 negotiation; transfers enabled.</td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>Manual mode.</td></tr>
</table></td></tr>
<tr><td>4</td><td>Ig_SEL: This bit prevents the CD1284 from considering the state of the SLCTIN* input when deciding whether or not to accept Compatibility mode forward data transfers.
When Ig_SEL is reset, SLCTIN* must be active (low) to receive data on the parallel port in response to a STROBE* input. If Ig_SEL is set, SLCTIN* is not considered and data is accepted regardless of its state. The Ig_SEL bit should be set/reset together with the E1284 bit.</td></tr>
<tr><td>3:2</td><td>Host Timer Test Control [1:0]: These two bits control the clock rate of the host timeout timer and are intended primarily for manufacturing test purposes. As such, normal user-level programming should leave these bits cleared (‘0’). When these bits are set to ‘1’, the timer is completely disabled – useful for factory debug purposes.</td></tr>
<tr><td>1:0</td><td>Manual Mode Control: These two bits provide direction and output enable manual control over the parallel port.
<table>
<tr><th>MMDir</th><th>ManOE</th><th>Mode</th></tr>
<tr><td>0</td><td>0</td><td>Reverse direction.</td></tr>
<tr><td>0</td><td>1</td><td>Reverse direction.</td></tr>
<tr><td>1</td><td>0</td><td>Forward direction disabled.</td></tr>
<tr><td>1</td><td>1</td><td>Forward direction enabled.</td></tr>
</table></td></tr>
</table>

7.8.12 Special Command Register

Register Name: SCR — 8-Bit Hex Address: 2A
Register Description: Special Command — Default Value: 00
Access: R/W

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
0	0	0	TestMux	ClrPs	SetPs	EPIrq	RevRq

This register provides the peripheral host processor to issue special commands to the channel control state-machine. In response, the state-machine will perform the indicated IEEE STD 1284-defined handshake on the parallel interface.

Bit	Description
7:5	These read-only bits are always ‘0’.
4	**TestMux:** When this bit is set, the state of the state machine is multiplexed onto the GPIO pins for debugging purposes. GPIO is not possible when this bit is set.
3:2	**Clear Pause and Set Pause:** These commands implement an error pause in Compatibility mode. Usually, errors are presented to the host parallel port by the peripheral during the active BUSY period of a data transfer. SetPs remains set until ClrPs is set, at which time both clear. In most cases, the slave host also sets RevRq at the same time when SetPs is set to: 1) Lockup Compatibility mode with BUSY high, and 2) Request a reverse transfer if the master requests that an additional status be sent in the reverse direction
1	**EPP Interrupt Request:** This command causes the state machine to generate the EPP interrupt sequence. This bit clears on the initiation of the Intr (PerClk) pulse on the parallel port interface.
0	**Reverse Request:** This command requests that the host parallel port initiate the defined interface reversal handshake as defined by the IEEE STD 1284 specification. The command bit clears to indicate completion after the command executes on the interface. For Reverse Nibble and Reverse Byte modes, this occurs after negotiation is complete; in ECP mode, it occurs after the Reverse Request signal on the parallel port interface goes low. In ECP mode, nPeriphRequest (nFault) is driven low to request that the host-side parallel port reverse the direction of the interface. When this bit is set upon termination of Compatibility mode, the CD1284 can indicate that reverse data is available (through the nDataAv signal) immediately upon recognition of a Reverse Nibble or Reverse Byte negotiation. To obtain this behavior, this bit should be initialized to ‘1’ and set to ‘1’ upon termination of Compatibility mode.

7.8.13 Short Pulse Register

Register Name: SPR Register Description: Short Pulse Access: Read/Write						8-Bit Hex Address: 26 Default Value: 00	
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
8-bit Binary Value							

This register performs two functions,

- It sets the duration of the short pulse used by the IEEE 1284 protocols for all modes other than Compatibility;
- In Compatibility mode, it sets the duration of the ACK* pulse.

For non-compatible modes, SPR must be set to n – 2, where n is the number of CLKs in a 500-ns pulse. The peripheral host initializes this register with the appropriate value to generate a 500-ns pulse width based on the operating frequency of the device. In Compatibility mode, SPR should be set to the needed length of the ACK* pulse. This is provided to enable the device to interface to

slow masters that require an ACK* pulse longer than the maximum specified in the IEEE STD 1284 specification. The table below shows some examples of the necessary binary value for various system clock frequencies to set the 500-ns pulse width.

Clock (MHz)	SPR Value	Resultant Pulse Width (ns)
16	8	500
20	10	500
25	13	520

7.9 Pin Control Registers

The parallel port has five outputs and four inputs. The pin assignments are the same as those defined in the IEEE STD 1284 specification. The definition of the pins depends on the current negotiated mode; these are detailed in the following descriptions.

7.9.1 Signal Status Register

Register Name: SSR
Register Description: Signal Status
Access: Read/Write
8-Bit Hex Address: 2F
Default Value: 00

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
0	0	0	0	A1284	nInit	HstBsy	HstClk

The bits in this register show the results of changes specified in the ODR and ZDR. Normally, the host reads this register in response to a signal change interrupt generated by the CD1284. This register is active and valid only in Manual mode. Bits 7:4 return zeros when read. A write of any value to the register clears it.

7.9.2 Zeros Detect Register

Register Name: ZDR
Register Description: Zeros Detect
Access: Read/Write
8-Bit Hex Address: 2C
Default Value: 00

0	0	0	0	A1284	nInit	HstBsy	HstClk

Setting the bits in this register enables the CD1284 to generate an interrupt – if the SigCh bit (PCIER[4]) is set – when the selected signal changes from high-to-low (falling edge). Bits 7:4 are reserved and must be written as ‘0’; they return ‘0’ when read. The settings in this register have no effect (that is, the SigCh interrupt is not generated) unless the device is in Manual mode.

8.0 Electrical Specifications

Note: Verify with your local sales office that you have the latest datasheet before finalizing a design.

8.1 Absolute Maximum Ratings

- Supply voltage (V_{CC})+7.0 V (volts)
- Input voltages, with respect to ground-0.5 V to V_{CC} +0.5 V
- Operating temperature (T_A)0°C to 70°C
- Storage temperature-65°C to 150°C
- Power dissipation0.25 W (watt)

Note: Stresses above those listed under Absolute Maximum Ratings can cause permanent damage to the device. This is a stress rating only, and functional operation of the device at these or any conditions above those indicated in the recommended operating conditions is not implied. Exposure to absolute maximum rating conditions for extended periods can affect device reliability.

8.2 Recommended Operating Conditions

Supply voltage (V_{CC})5 V ± 5%

Operating free air ambient temperature0°C < T_A < 70°C

System clock25 MHz

ESD (Human body model)	100 pF. 1.5 kΩ, ± 2 kV	Mil-Std-883D Method 3015.7
ESD (Machine model)	200 pF, 0 Ω, ± 200 V	EIAJ IC-121
Latch-up	I/O ± 100 mA, V_{CC} = 5 V Temperature = 25°C and 70°C	JEDEC number 17
	V_{CC} ramp 5 V to 9 V Temperature = 25°C and 70°C	JEDEC number 17
Hysteresis	200 mV	

(@ V_{CC} = 5V ± 5%, T_A = 0°C to 70°C)

Symbol	Parameter	MIN	MAX	Units	Test Conditions
V_{IL}	Input low voltage	–0.5	0.8	V	
V_{IH}	Input high voltage	2.0	V_{CC}	V	[1]
V_{OL}	Output low voltage		0.4	V	I_{OL} = 2.4 mA [2]
V_{OH}	Output high voltage	2.4		V	I_{OH} = -400 µA

(@ V_{CC} = 5V ± 5%, T_A = 0°C to 70°C)

I_{IL}	Input leakage current	–10	10	µA	$0 < V_{IN} < V_{CC}$
I_{LL}	Data bus tristate leakage current	–10	10	µA	$0 < V_{OUT} < V_{CC}$
I_{OC}	Open-drain output leakage current	–10	10	µA	$0 < V_{OUT} < V_{CC}$
I_{CC}	Power supply current		60	mA	CLK = 25 MHz
C_{IN}	Input capacitance		10	pF	
C_{OUT}	Output capacitance		10	pF	

NOTES:
1. V_{IH} is 2.7 V minimum on RESET*, CLK, and DMAACK*.
2. V_{OL} for open-drain signals is 0.5 V @ 8 mA sinking because these signals can be wire-OR'ed in some systems and can have multiple pull-up resistors that increase the load on the output.

The signals specific to the parallel port meet all requirements of the IEEE STD 1284 specification, except for input signal protection (–2.0 to + 7.0 V); external circuitry is required to meet this specification.

Symmetrical input/output drive: ±14 mA
Controlled voltage slew rate: 0.4 V/µs
Input hysteresis: 0.8 V

Note: While the CD1284 is a highly dependable device, there are a few guidelines to ensure that the maximum possible level of overall system reliability is achieved. First, design the PC board to provide maximum isolation of noise. A four-layer board is preferable, but a two-layer board will work if proper power and ground distribution is implemented. In either case, decoupling capacitors mounted close to the CD1284 are strongly recommended. Noise typically occurs when either the CD1284 data bus drivers come out of tristate to drive the bus during a read, or when an external bus buffer turns on during a write cycle. This noise, a rapid rate-of-change of supply current, causes 'ground bounce' in the power-distribution traces. This ground bounce, a rise in the voltage of the ground pins, effectively raises the input logic thresholds of all devices in the vicinity, resulting in the possibility of a '1' being interpreted as a '0'.
To reduce the possibility of ground-bounce affecting the operation of the CD1284, Intel has specified the input-high voltage (V_{IH}) of the CLK and RESET* pins at 2.7 V, instead of the TTL-standard 2.0 V. This eliminates any sensitivity to ground bounce, even in extremely noisy systems. Although 2.7 V is higher than the industry-standard 2.4-V output (V_{OH}) specified for TTL, there are several simple ways to meet this specification:

1. Use any of the available advanced-CMOS logic families (FACT, ACL, etc.). These CMOS output buffers will pull-up close to V_{CC} when not heavily loaded. In addition, AS and ALS TTL can be used if the output of the TTL device is only driving one or two CMOS loads.
2. As noted in the Texas Instruments *ALS/AS Logic Data Book* (1986 — pages 4-18 and 4-19), the V_{OH} output of these families exceeds 3.0 V at low-current loading. Other manufacturers publish similar data. Intel recommends the use of one of these two options for the CLK input to ensure fast, clean edges.
   Note that RESET* can, if desired, be pulled up passively with ≤ 1-kΩ resistor.

8.3 AC Characteristics

8.3.1 Asynchronous Timing

Refer to the Figures 6-1 through 6-7 for the reference numbers in the following table.

Table 27. Asynchronous Timing Reference Parameters (Sheet 1 of 2)

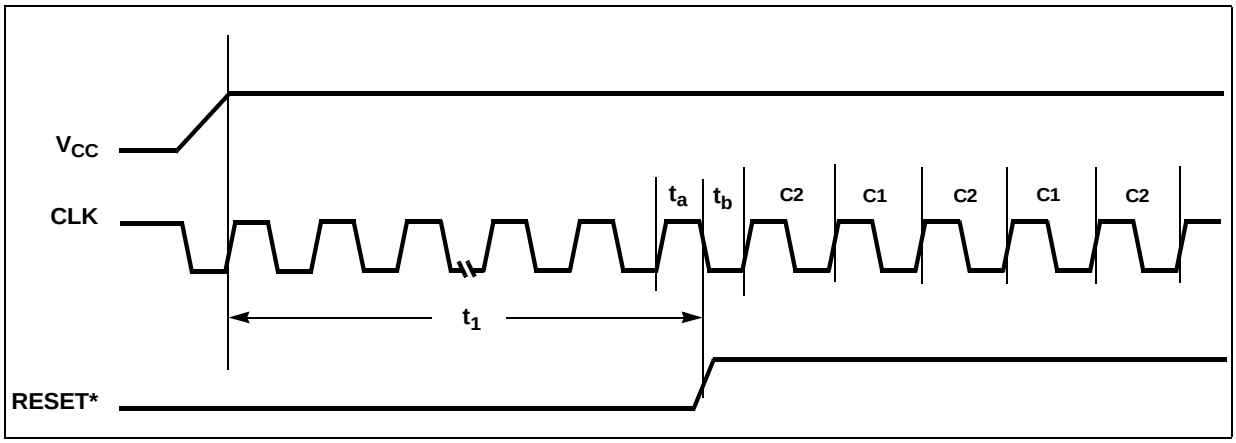
Timing Number	Figure	Parameter	MIN	MAX	Unit
t_1	Figure 20	RESET* low pulse width	10		T_{CLK}
t_2	22	Address setup time to CS* or DS*	10		ns
t_3	22	R/W* setup time to CS* or DS*	10		ns
t_4	22	Address hold time after CS*	0		ns
t_5	22	R/W* hold time after CS*	0		ns
t_6	22	DTACK* low to read data valid		10	ns
t_7	22	DTACK* low from CS* or DS[2]	2 T_{CLK}	4 T_{CLK} + 30	ns
t_8	22	Data Bus tristate after CS* or DS* high	0	30	ns
t_9	22	CS* or DGRANT* high from DTACK* low	0		ns
t_{10}	22	DTACK* inactive from CS* or DGRANT* and DS* high		40	ns
t_{11}	22	DS* high pulse width	10		ns
t_{12}	23	Write data valid from CS* and DS* low		1T_{CLK}	ns
t_{13}	23	Write data hold time after DS* high	0		ns
t_{14}	21	Clock period (TCLK)[1, 3]	40.0	1000	ns
t_{15}	21	Clock low time[1]	0.3 T_{CLK}	0.7 T_{CLK}	ns
t_{16}	21	Clock high time[1]	0.3 T_{CLK}	0.7 T_{CLK}	ns
t_{17}	24	Propagation delay, DGRANT* and DS* to DPASS*		35	ns
t_{18}	24	Setup time, SVCACK* to DS* and DGRANT*	10		ns
t_{19}	25	Setup time, DMAACK* to rising edge of CLK	10		ns
t_{20}	25	Hold time, read data after rising edge of CLK	10	30	ns
t_{21}	27	Setup time, write data to rising edge of CLK	0		ns
t_{22}	22	DTACK* active pull-up time[4]		see note 4	ns
t_{23}	25	Data valid after falling edge of CLK (DMA read)		25	ns
t_{24}	25 27	Hold time, DMAREQ* after DMAACK* falling edge, last DMA cycle	10	1 CLK + 15	ns

Table 27. Asynchronous Timing Reference Parameters (Sheet 2 of 2)

Timing Number	Figure	Parameter	MIN	MAX	Unit
The following timing numbers are for the back-to-back asynchronous DMA timing diagrams.					
t_{25}	26	Hold time, DMAACK* active (DMA read/write)	3 CLK		
t_{26}	26	Delay, data valid after falling edge DMAACK* (DMA read)	0.5 CLK + 20	1.5 CLK + 25	ns
t_{27}	26	Hold time, data valid after rising edge DMAACK* (DMA read)	10	30	ns
t_{28}	26 28	Inactive time, DMAACK* (DMA read/write)	10		ns
t_{29}	26 28	Hold time, DMAREQ* rising edge after DMAACK* falling edge (DMA read/write)	10	1 CLK + 15	ns
t_{30}	28	Hold time, DMAACK* active (DMA write)	2.5 CLK		
t_{31}	28	Delay, data valid after falling edge DMAACK* (DMA write)		1.5 CLK	
t_{32}	28	Hold time, data valid (DMA write)	3 CLK + 10		ns

NOTES:
1. Timing numbers for RESET* and CLK in the table above are valid for both asynchronous and synchronous specifications. The device operates on any clock with a 40–60 duty cycle or better.
2. On host-I/O cycles immediately following SVCACK* cycles and writes to EOSRR, DTACK* is delayed by 20 CLKs (1 µs @ 20 MHz, 800 ns @ 25 MHz). On systems that do not use DTACK* to signal the end of the I/O cycle, wait states or some other form of delay generation must be used to assure that the CD1284 is not accessed until after this time period.
3. As TCLK increases, device performance decreases. A minimum clock frequency of 25 MHz is required to ensure performance as specified. The recommended maximum TCLK is 1000 ns.
4. DTACK* sources current (drives 'high') until the voltage on the DTACK* line is approximately 1.5 V; then DTACK* goes to the 'open-drain' (high-impedance) state.

Figure 20. Reset Timing



Note: For synchronous systems, it is necessary to determine the clock cycle number so that interface circuitry can stay in lock-step with the device. CLK numbers can be determined if RESET* is released within the range t_a–t_b; t_a is defined as 10-ns minimum after the rising edge of the clock; t_b is defined as 5-ns minimum before the next rising edge of the clock. If these conditions are met, the cycle starting after the second rising edge is C1. See the synchronous timing diagrams for additional information. Clock numbers are not important in asynchronous systems.

Figure 21. Clock Timing

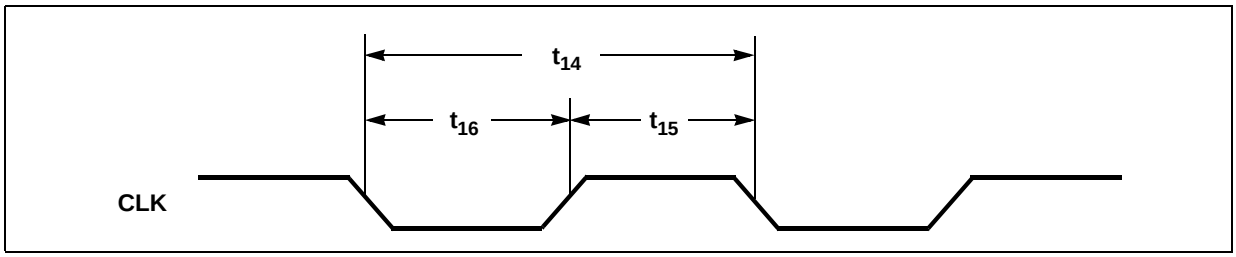


Figure 22. Asynchronous Read Cycle Timing

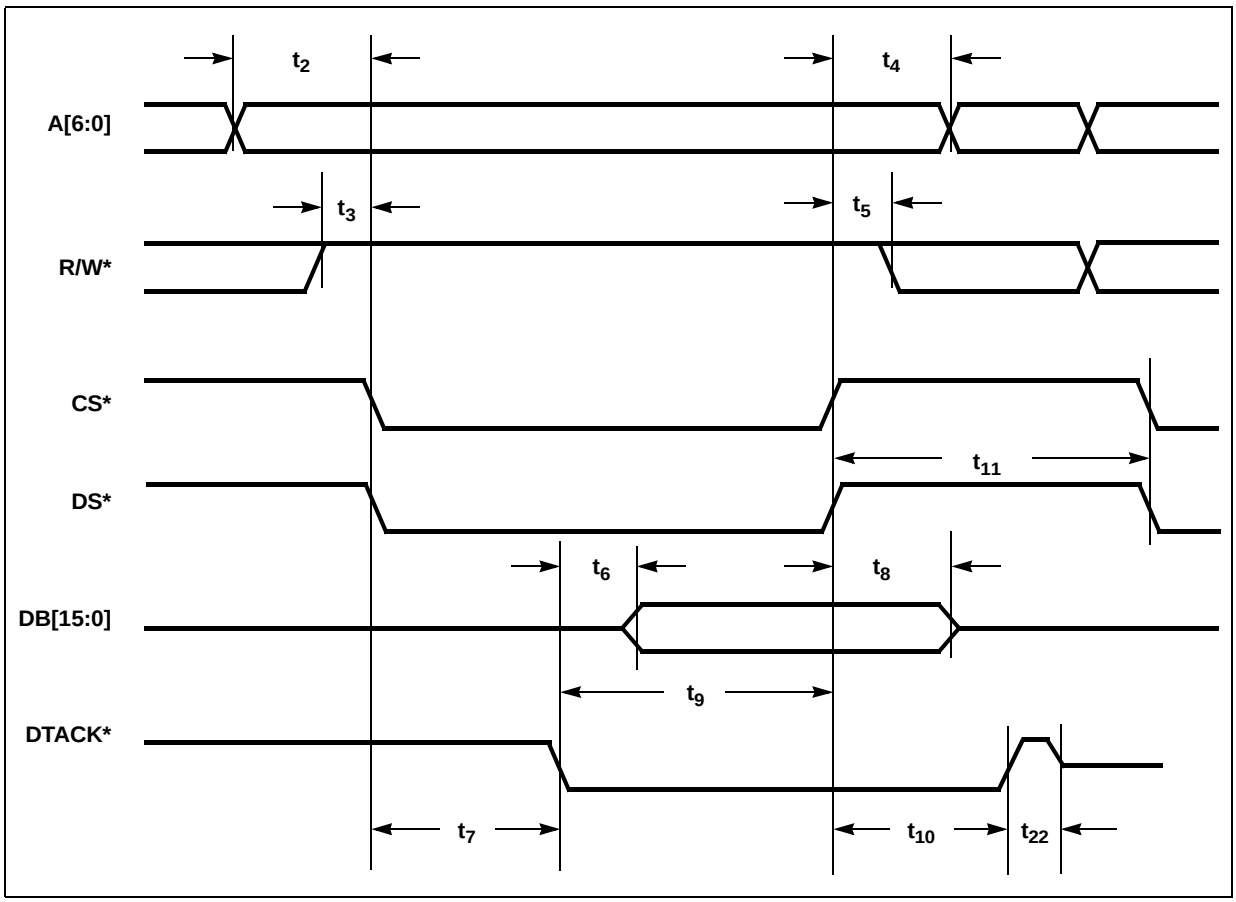


Figure 23. Asynchronous Write Cycle Timing

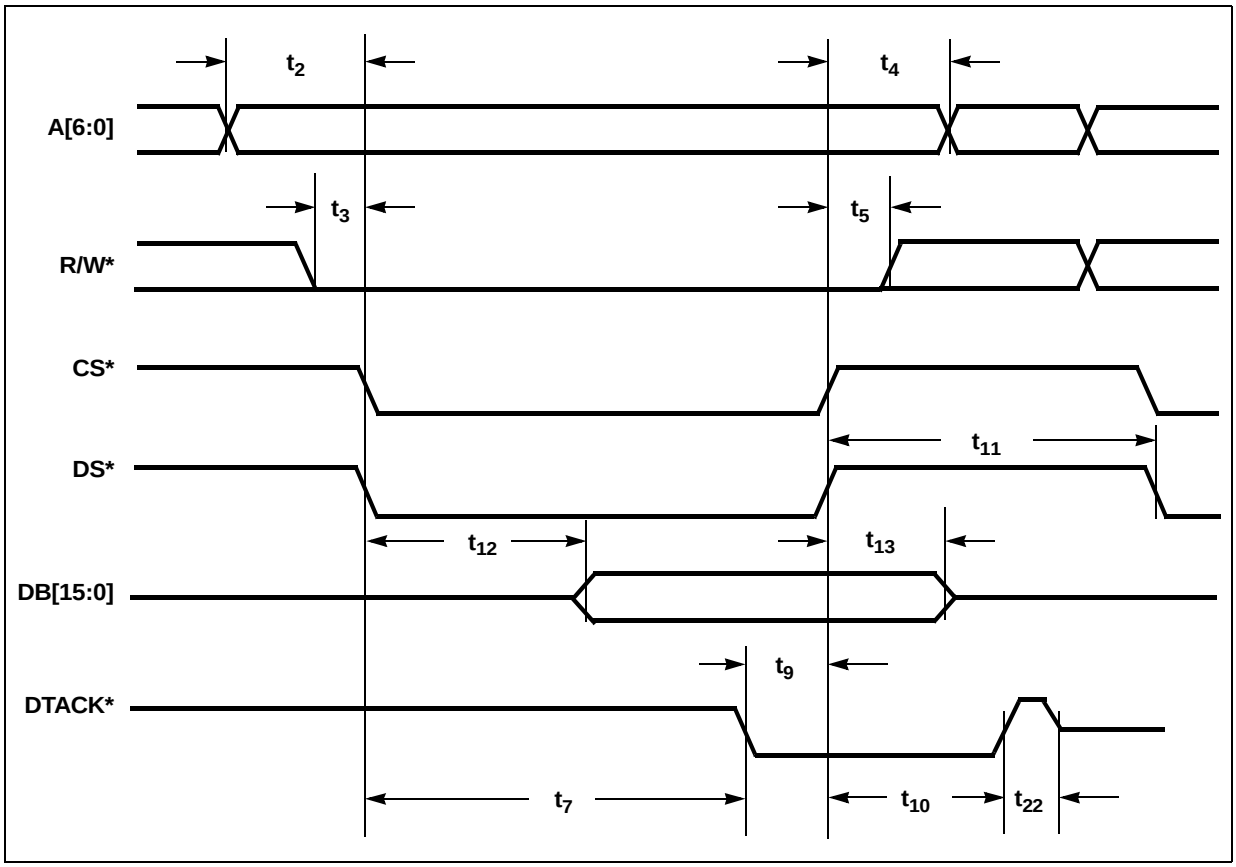


Figure 24. Asynchronous Service Acknowledge Cycle Timing

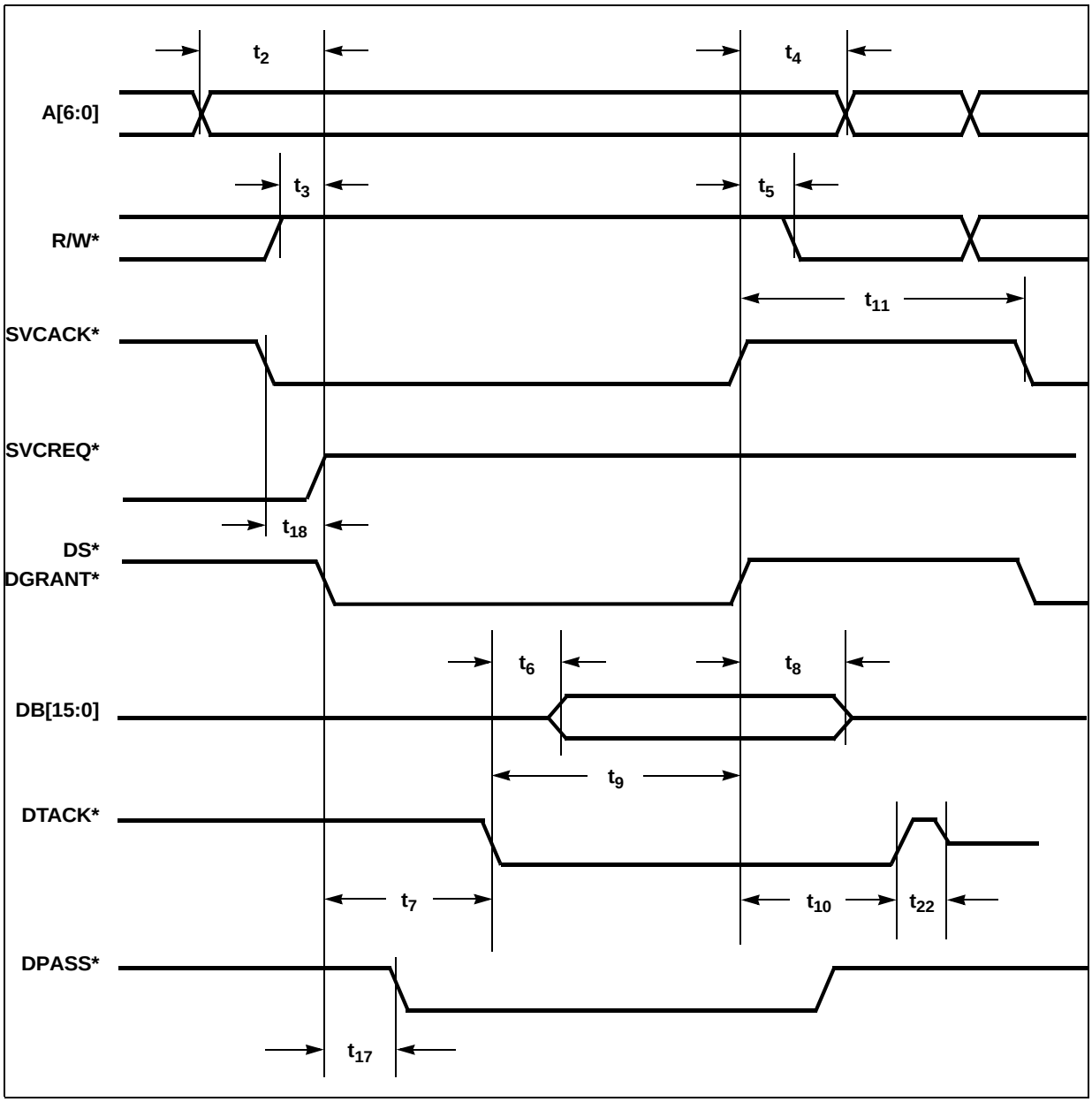
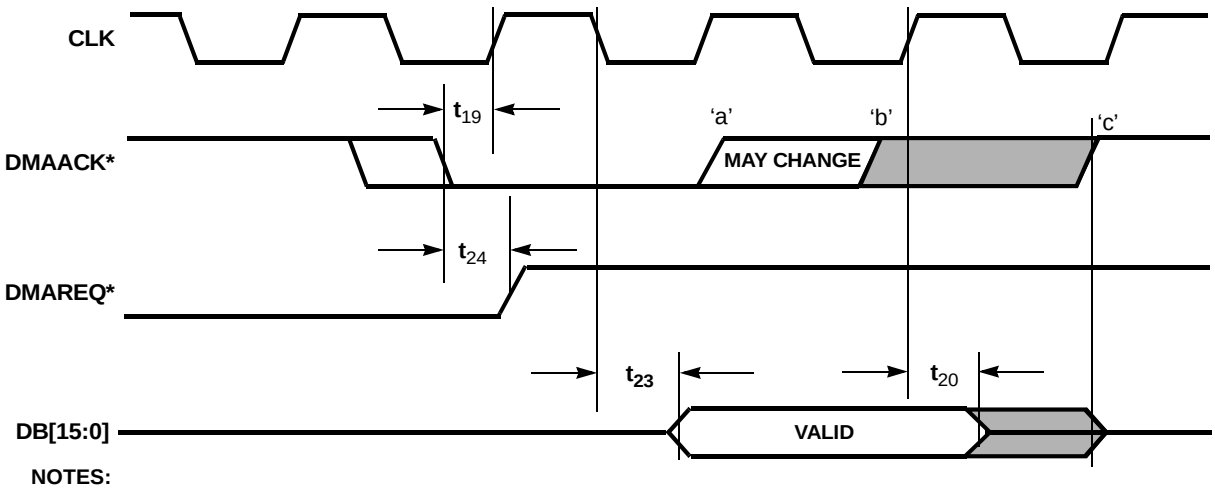
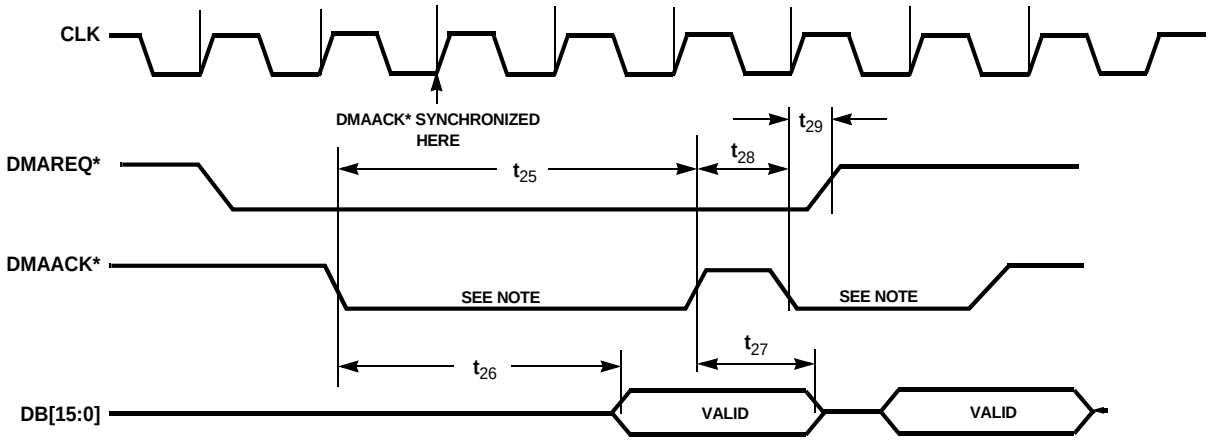


Figure 25. Asynchronous DMA Read Cycle Timing



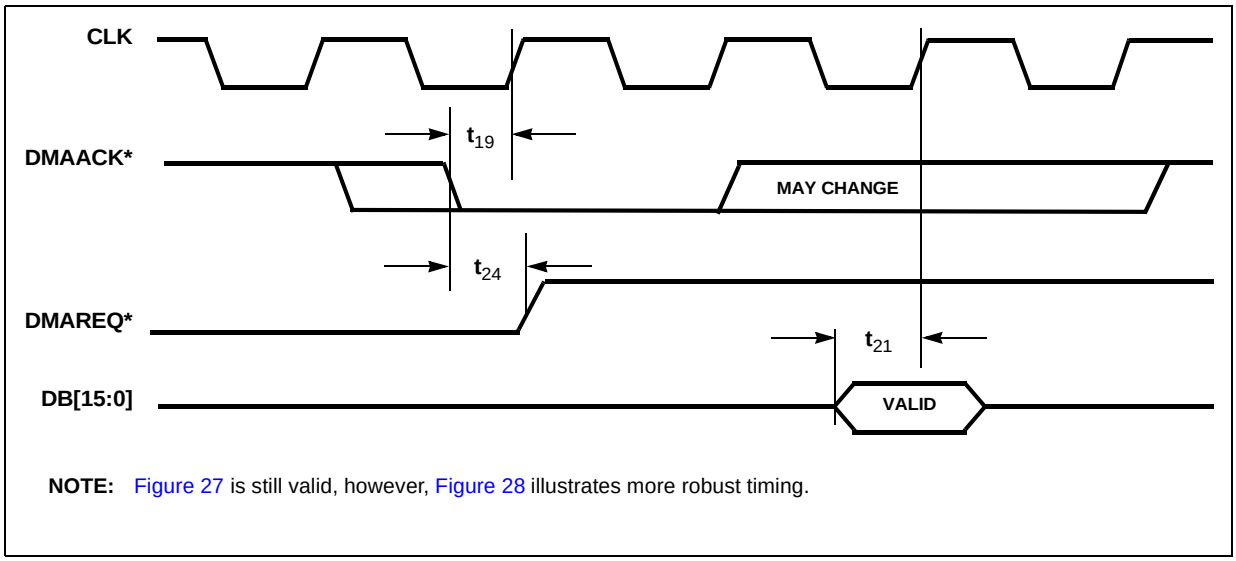
NOTES:
1. The DMA handshake operates in asynchronous mode only if the AsyncDMA bit is set in PACR.
2. If DMAACK* is released after point 'a,' but before point 'b' (two rising CLK edges after the falling edge of DMAACK*), DB[15:0] is released at t_{20} following the rising edge of CLK. If DMAACK* is held past this edge, it controls the release of DB[15:0]; the data bus remains active until DMAACK* becomes inactive (point 'c').
3. Figure 25 is still valid, however, Figure 26 illustrates more robust timing.

Figure 26. Asynchronous DMA Read Cycle Timing (Two Back-to-Back DMA Reads)



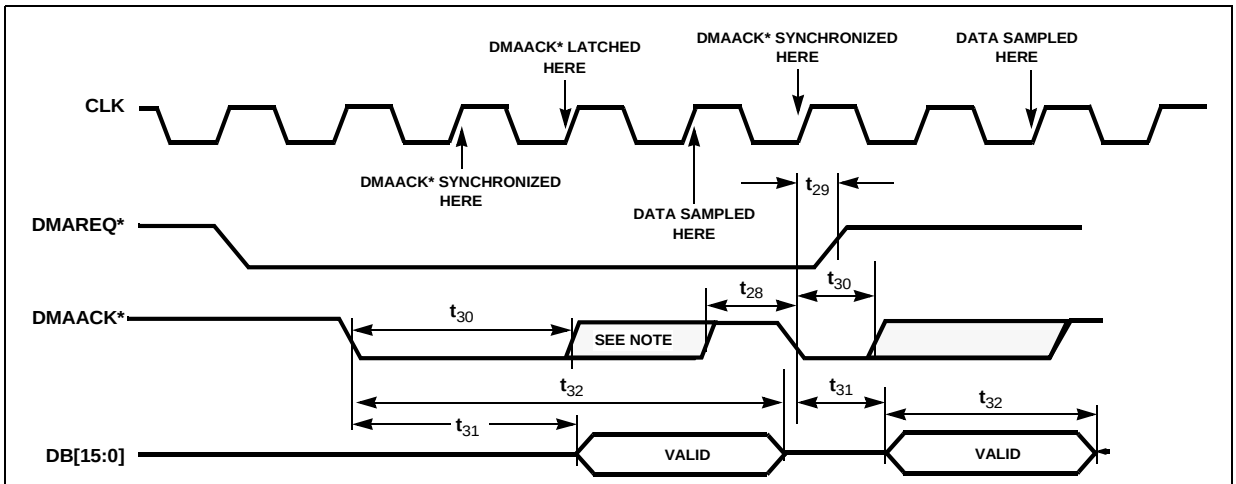
NOTE: The falling edge of DMAACK* is synchronized internally with the rising edge of the clock when asynchronous timing is selected by PACR[1]. The data valid time can vary by as much as one full CLK cycle depending on when DMAACK* falling edge occurs in relation to the CLK rising edge. The minimum DMAACK* active time must be met to ensure that the data has become valid before the rising edge of DMAACK*. The DMAACK* can be extended to any length, which extends the data valid hold time accordingly. If t_{25} is not met and DMAACK* is deasserted in less than t_{25} (MIN), then the data bus tristates t_{27} after the third rising clock edge following the assertion of DMAACK*.

Figure 27. Asynchronous DMA Write Cycle Timing



NOTE: Figure 27 is still valid, however, Figure 28 illustrates more robust timing.

Figure 28. Asynchronous DMA Write Cycle Timing (Two Back-to-Back DMA Writes)



NOTE: The data is sampled on the third rising edge of CLK following the assertion of DMAACK*. If DMAACK* is held active for more than three CLK cycles then the next DMA write cycle will simply be delayed, but the data will still be sampled on the third rising CLK edge following the assertion of DMAACK*. If DMAACK* is active for < 3 CLKs, the n the data is still sampled on the third rising CLK edge following the assertion of DMAACK* (provided that DMAACK* is active long enough for the device to lastch it. Due to this somewhat synchronous behavior, care must be taken to guarantee that the data is valid at this CLK edge. Do not assume that the data will be sampled on the deassertion of DMAACK*.

8.3.2 Synchronous Timing

Use the following table as a reference to timing parameters of figures in this section.

Table 28. Synchronous Timing Reference Parameters

Timing Number	Figure	Parameter	MIN	MAX	Unit
t_1	29	Setup time, CS* and DS* to C1 rising edge	15		ns
t_2	29	Setup time, R/W* to C1 rising edge	15		ns
t_3	29	Setup time, address valid to C1 rising edge	20		ns
t_4	29	C2 rising edge to data valid		60	ns
t_5	29	DTACK* low from C3 rising edge[1]		30	ns
t_6	29	CS* and DS* trailing edge to data bus high-impedance		30	ns
t_7	29	CS* and DS* inactive between host accesses	10		ns
t_8	29	Hold time, R/W* after C3 rising edge	20		ns
t_9	29	Hold time, address valid after C3 rising edge	0		ns
t_{10}	30	Setup time, write data valid to C2 rising edge	0		ns
t_{11}	31	Setup time, DS* and DGRANT* to C1 rising edge	30		ns
t_{12}	31	Setup time, SVCACK* to DS* and DGRANT*	10		ns
t_{13}	30	Hold time, write data valid after C3 rising edge	0		ns
t_{14}	31	Propagation delay, DS* and DGRANT* to DPASS*		35	ns
t_{15}	32 33	Falling edge DMAREQ* after rising edge CLK (DMA write/read)		25	ns
t_{16}	32 33	Hold time, rising edge DMAREQ* after falling edge DMAACK* (DMA write/read)		20	ns
t_{17}	32	Setup time, data valid before rising edge C3 (DMA write)	5		ns
t_{18}	32 33	Setup time, falling edge DMAACK* to falling edge C1 (DMA write/read)	10		ns
t_{21}	29	DTACK* active pull-up time[2]			
t_{22}	32	Hold time, data valid after rising edge C3 (DMA write)	5		
t_{23}	33	Hold time, data valid after rising edge C1 (DMA read)	10	30	
t_{24}	33	Data valid after falling edge C1 (DMA read)		25	
t_{25}	33	Inactive time, DMAACK* (DMA read)	10		

NOTES:
1. On host I/O cycles immediately following SVCACK* cycles and writes to EOSRR, DTACK* are delayed by 20 CLKs (1 ms @ 20 MHz, 800 ns @ 25 MHz). On systems that do not use DTACK* to signal the end of the I/O cycle, wait states or some other form of delay generation must be used to assure that the CD1284 is not accessed until after this time period.
2. DTACK* sources current (drives ‘high’) until the voltage on the DTACK* line is approximately 1.5 V; then DTACK* enters the ‘open-drain’ (high-impedance) state.

Figure 29. Synchronous Read Cycle Timing

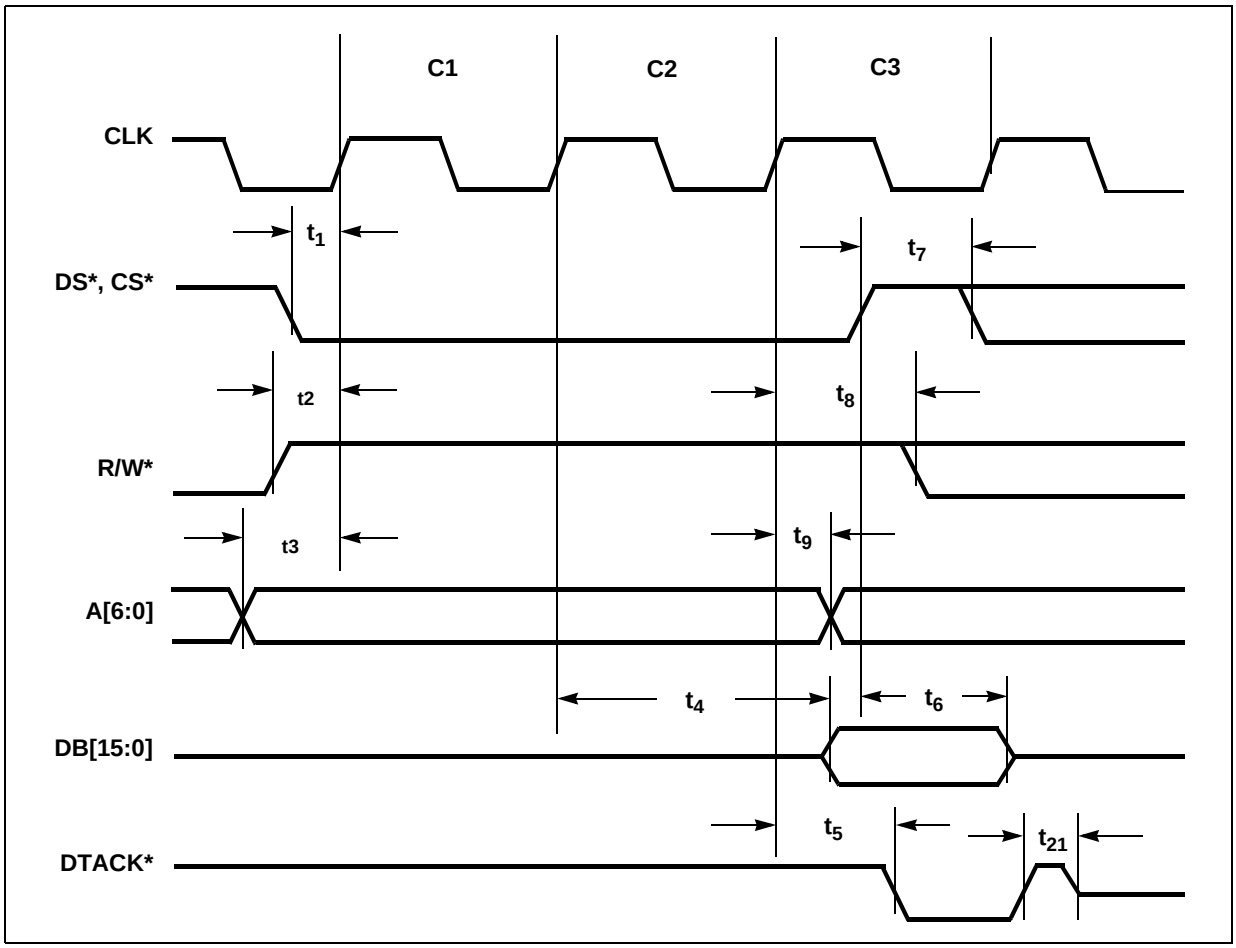


Figure 30. Synchronous Write Cycle Timing

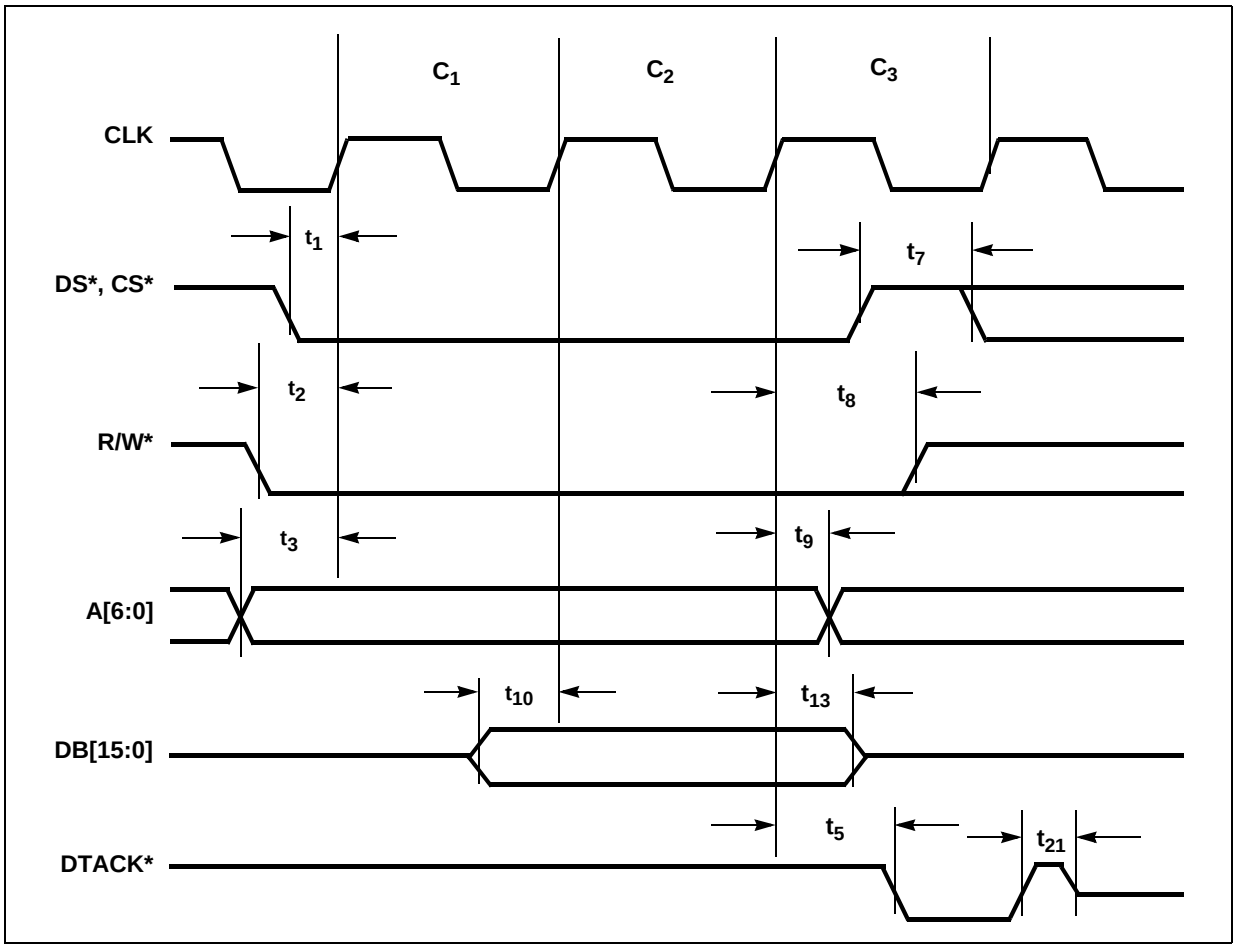


Figure 31. Synchronous Service Acknowledge Cycle Timing

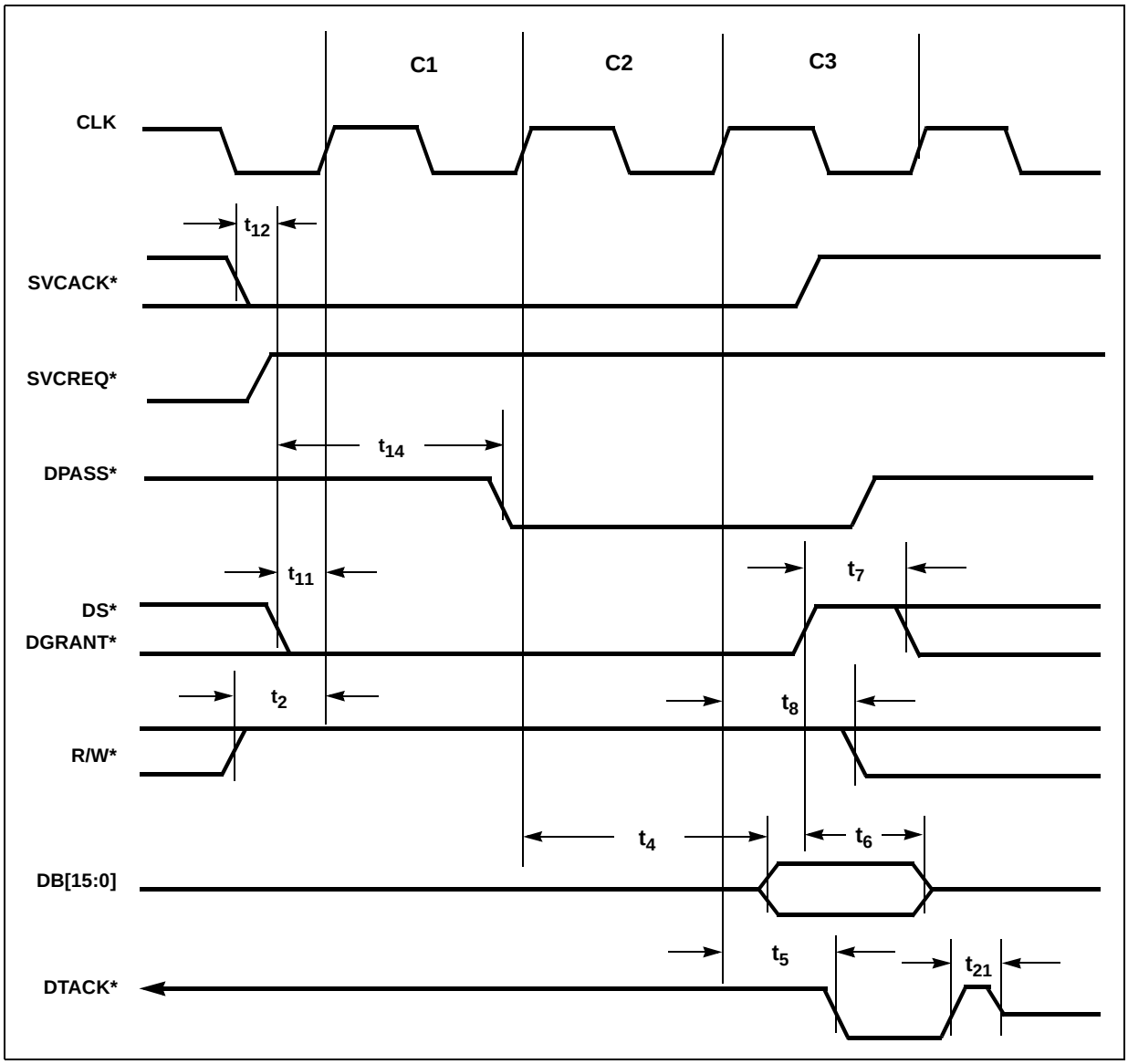
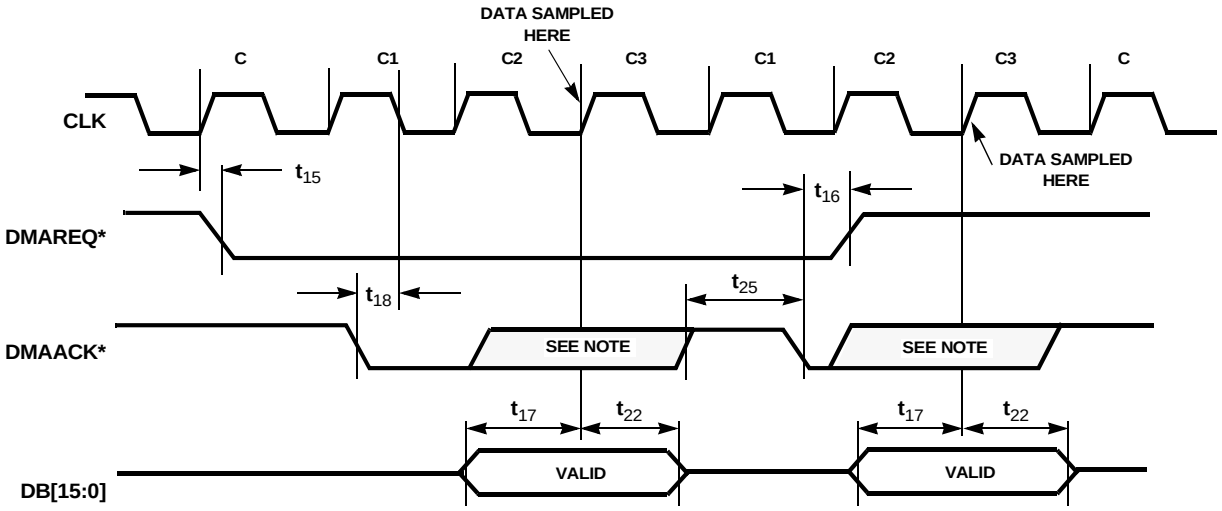
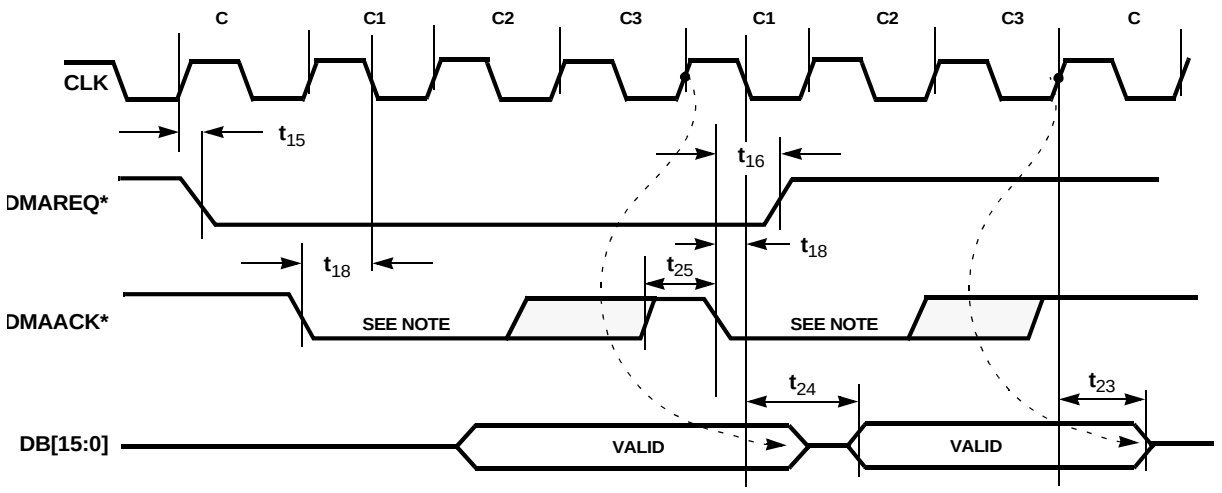


Figure 32. Synchronous DMA Write Cycle Timing (Two Back-to-Back 3-Cycle DMA Writes)



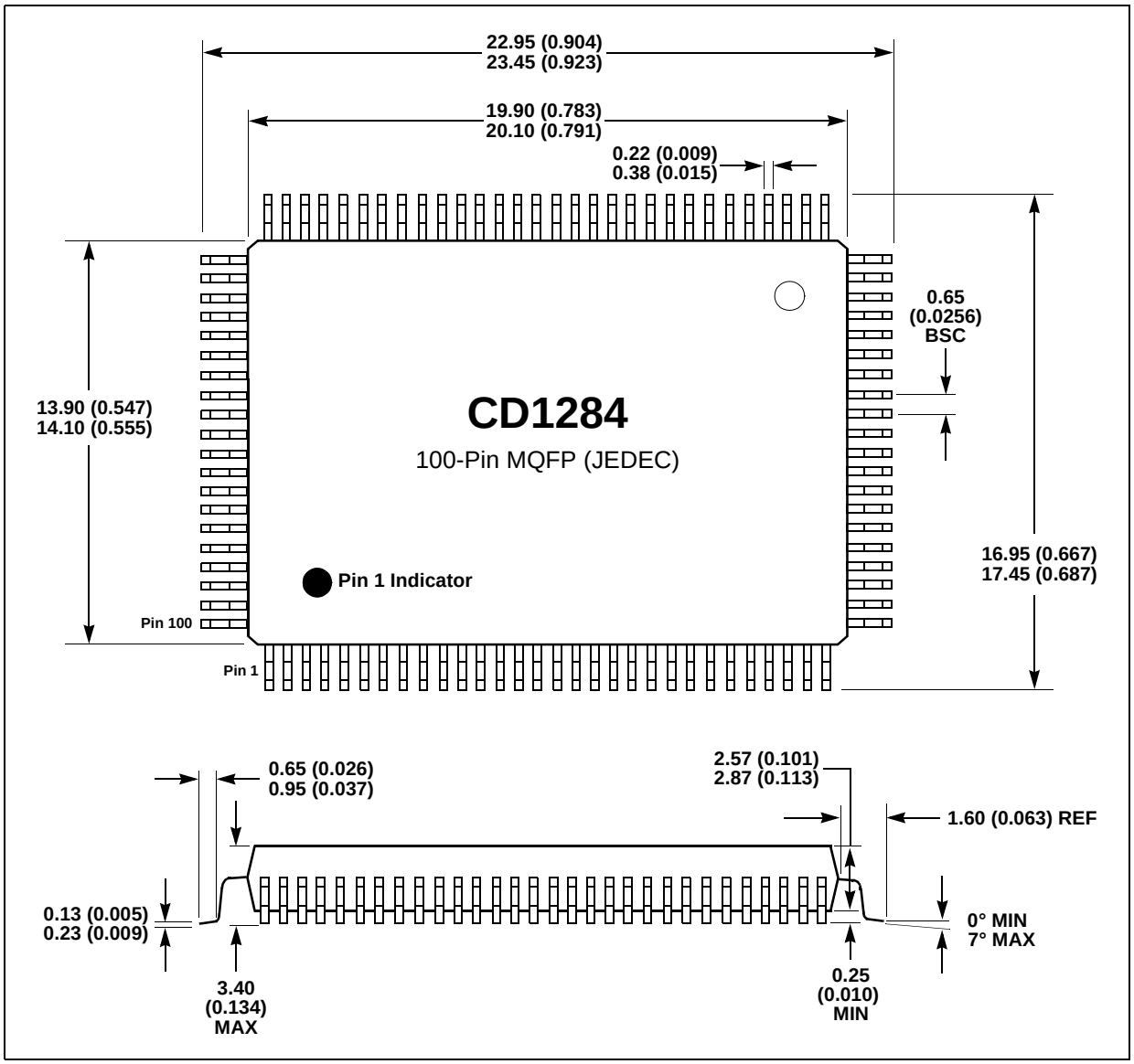
NOTE: The data is sampled on the second rising edge of CLK following the assertion of DMAACK*, as long as setup time (t_{18}) is met. If DMAACK* is held active for more than 2.5 CLK cycles, then the next DMA cycle is simply delayed; the data is still sampled on the second rising CLK edge following the assertion of DMAACK*.

Figure 33. Synchronous DMA Read Cycle Timing (Two Back-to-Back 3-Cycle DMA Reads)



NOTE: The data is driven (t_{24}) after the first falling edge of CLK following the assertion of DMAACK*, as long as setup time (t_{18}) is met. If DMAACK* is held active for more than 2.5 CLK cycles after C1 falling edge, then the next DMA cycle is simply delayed, but the data is still driven (t_{24}) after the first falling CLK edge following the next assertion of DMAACK*.

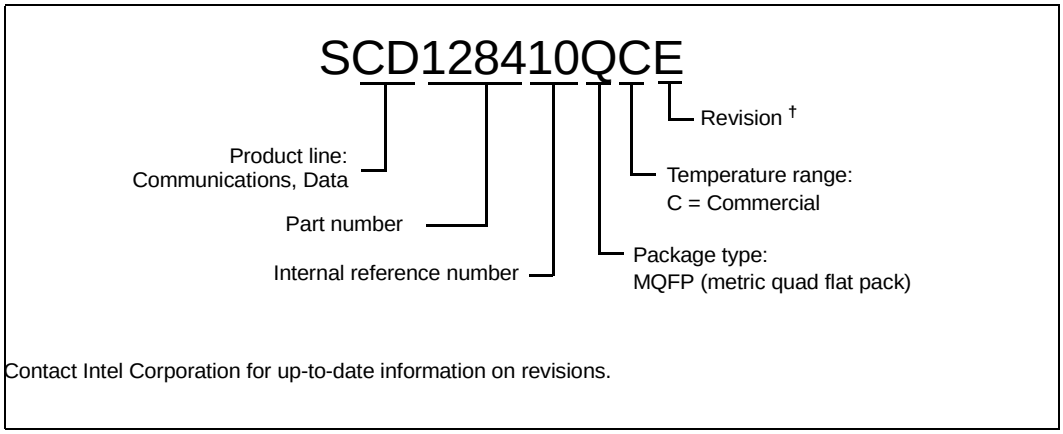
9.0 Package Dimensions



NOTES:
1. Dimensions are in millimeters (inches), and controlling dimension is millimeter.
2. Before beginning any new design with this device, please contact Intel for the latest package information.

10.0 Ordering Information

The order number for the CD1284 is:



Contact Intel Corporation for up-to-date information on revisions.

11.0 Appendix A

11.1 Commonly Asked Questions

- Using the SPR to Change Acknowledge Pulse Width in Compatible Mode

  Some older hosts may require an acknowledge pulse width longer than the default 500 ns in Compatible mode. The SPR can be used to change the pulse width in Compatible mode, but it will also affect the transfer rate in the other modes. If the ACK* pulse width is extended 1 µs, the transfer rates in other modes is slowed also. This should not be a concern as IEEE 1284-compliant hosts work with an ACK* pulse width of 500 ns as specified in the IEEE 1284 specification (page 30). While non-IEEE 1284-compliant hosts cannot support any of the advanced modes. In other words, if the host supports IEEE 1284 advanced modes (for example ECP), then it also supports an ACK* pulse width of 500 ns in Compatible mode. If the host is not IEEE 1284-compliant, then it does not support any of the advanced modes and therefore the SPR is only used for compatibility mode.

- BUSY/ACK* Timing Variations

  The SPR cannot be used to support the Ack-while-Busy timing. If the SPR value is changed to extend the ACK* pulse width, then the BUSY signal is extended as well. This means that the CD1284 only supports the Compatible mode timing, Ack-in-Busy, as specified on pages 28–30 of the IEEE 1284 specification. Please read Section 6.3 “Compatibility Mode” starting on page 28. Based on this description of Compatibility mode, it is our belief that the Ack-in-Busy timing on the peripheral-side interoperates with all existing hosts, including those that monitor BUSY but not ACK*.

- Device ID

  At this time, Intel has no more information about device ID other than that is listed on page 52 of the IEEE 1284 specification. Contact Larry Stein, Chair of the IEEE 1284.3 working group, at Far Point Communications (Fax: (805) 726-4438) for more information. Far Point Communications also sells IEEE 1284-compliant ISA add-in boards for the PC. This board can be useful for testing CD1284 applications.

- Reversing the Channel with Data Remaining in the FIFO

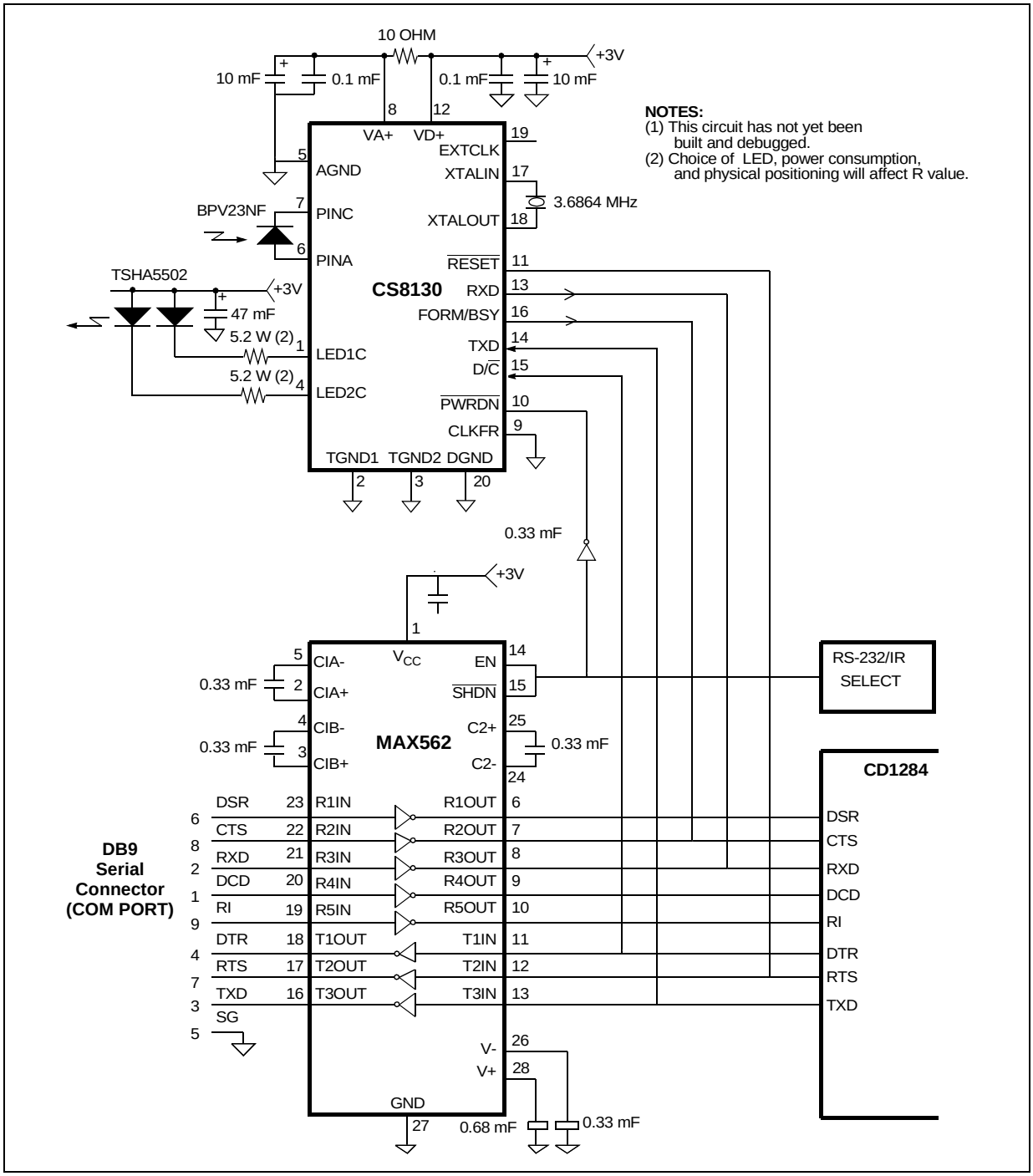
  The software must handle the situation where the host switches the direction of the parallel interface from reverse to forward while data remains in the reverse FIFO. If this occurs then the CD1284 produces a change of direction interrupt. When software detects this interrupt, it must read the value in the PFQR (Parallel FIFO Quantity register) and use this value to determine the bytes remaining in the FIFO. The software buffer pointer(s) must then be adjusted by that amount so that the data in the FIFO can be resent when the direction is reversed again. After the pointers are adjusted, the FIFO must be flushed (cleared) and the direction of the FIFO must be changed to forward so that data may be received from the host.

- RLE Data Count

  Software can access the RLCR to obtain the current count for RLE data.

12.0 Appendix B

Figure 34. UART to RS232 and IR Port Interface Motherboard Example Schematic



Index

A

A_1284 19
AB[6:0] 19
abbreviations 15
acronyms 15
AkDaRq 19
asynchronous serial data protocol 50

B

baud rate
  derivation 102
  generation 72
  tables 103
bit engines 50
BYTESWAP 19, 75

C

CCLK 20
CLK 19
CLK/2 19
common service acknowledge 42
compressed-data sequences 75
context 49
context switch 35
CPU interface 86
CS* 19
CTS* 55
CTS2* 20
CTS3* 20

D

daisy-chaining 41
DB[15:0] 19
detailed register descriptions 108
device reset 90
DGRANT* 19
diagnostic facilities 73
DMA
  buffer 74
  interface 34
  transfers 74
DMAACK* 19
DMAREQ* 19
DPASS* 19
DS* 19
DSR* 55
DSR2* 20
DSR3* 20
DTACK* 19
DTR threshold 58
DTR* 55
DTR2* 20
DTR3* 20

E

EBDIR 19
ECP mode 74
embedded transmit commands (ETC) 67
endian 35
EPP mode 79, 83

F

failed negotiation 81
Fair Share 41
fairness override 43
FIFO threshold 50
flow control 55

G

general-purpose I/O port 83
global function initialization 93
GND 18
GP[7:0] 19

H
hardware-activated service examples 97
HstBsy 19
HstClk 19

I
ID request 82
IEEE Standards Department 73
IEEE STD 1284 73
Implied XON mode 57
in-band flow control 55
initialization 90
interface 74
interrupts 36
    DirCh 48
    EPPAW 44
    IDReq 48
    NegCh 44
invalid termination 81
IVR 79

L
line break 52
line discipline 61
Local Loopback mode 73
loopback testing 73

M
modem service 96, 99
modes
    ECP 74
    EPP 79, 83
    Implied XON 57
    Local Loopback 73
    Manual 79
    Remote Loopback 73
    Reverse Byte 82
    Reverse Nibble 82
    Serial Poll, examples 94
multi-channel processing unit (MPU) 31

N
nDatAv 19
nInit 19
no new data time-out (NNDT) 51

O
odd-byte transfers 35
ODR 79
OUTEN 19
out-of-band flow control 55, 58

P
parallel channel service routines 99
parallel port
    FIFO 74
PD[7:0] 19
PDBEN 19
PerBsy 19
PerClk 19
pins
    descriptions 20
    diagram 17
    list 18
polling 40
protocol timing 83

R
R/W* 19
read cycles 33
receive direction 75
receiving compressed data 75
register summary 24
register usage 27
registers
    Channel – Parallel
        DER 26, 29
        DMABUF (high) 26, 29
        DMABUF (low) 26, 29
        EAR 26, 29
        HRSR 26, 29, 138
        HTVR 26, 29

IVR 26, 29
LIVR 26, 29, 38
MDR 26, 29
NER 26, 29
NSR 26, 29
ODR 26, 29
OVR 26, 29
PACR 26, 29
PCIER 26, 29
PCISR 26, 29
PCR 26, 29
PCRR 26, 29
PFCR 26, 29
PFEP 26, 29
PFFP 26, 29
PFHR1 26, 29
PFHR2 26, 29
PFQR 26, 29
PFSR 26, 29
PFTR 26, 29
RLCR 26, 29
SCR 26, 30
SDTCR 26, 29
SDTPR 26, 29
SPR 27, 30
SSR 27, 30
ZDR 27, 30
Channel – Serial
CCR 25, 28
CCSR 25, 28
COR1 25, 28
COR2 25, 28
COR3 25, 28
COR4 25, 28
COR5 25, 28
LIVR 25, 28
LNC 25, 28
MCOR1 25, 28
MCOR2 25, 28
MSVR1 25, 28
MSVR2 25, 28
RBPR 25, 28
RCOR 25, 28
RDCR 25, 28
RTPR 25, 28
SCHR1 25, 28
SCHR2 25, 28
SCHR3 25, 28
SCHR4 25, 28
SCRH 25, 28
SCRL 25, 28
SRER 25, 28
TBPR 25, 28
TCOR 25, 28
Channel Control Status (CCSR) 56
Global
CAR 24, 27
GFRCR 24, 27
GPDIR 24, 27
GPIO 24, 27
MICR 24, 27
MIR 24, 27
PIR 24, 27
PPR 24, 27
RICR 24, 27
RIR 24, 27
SVRR 24, 27
TICR 24, 27
TIR 24, 27
Parallel Port
NSR 81
PCR 151
SCR 81, 152
Receive Data Count (RDCR) 51
Receive Data/Status (RDSR) 52
Virtual – All
EOSRR 24, 28
Virtual – Serial
MISR 24, 27
MIVR 24, 27
PIVR 24, 27
RDSR (data) 24, 27
RDSR (status) 24, 27
RIVR 24, 27
TDR 24, 27
TIVR 24, 27

Remote Loopback mode 73
RESET* 18, 54
RLE (run-length-encoding) 75
RTS* 55
RTS2* 20
RTS3* 20
RxD 73
RXD2 20
RXD3 20
RxFloff 56
RxFlon 56

S

scanning loop 94
SCHR1 55
SCHR2 55
Serial Poll mode 94
serial receive service 95, 97
serial transmit service 96, 98
service-request/acknowledge 35
special characters 61
SSR 79
stale data timer 76
start bit 50
stop bit 50
SVCACK* 37
SVCACKM* 19
SVCACKP* 19
SVCACKR* 19, 37
SVCACKT* 19, 37
SVCREQM* 19
SVCREQP* 19
SVCREQR* 19
SVCREQT* 19
SVRR 40
synchronous timing reference parameters 157, 164

T

timer 51, 55
Transmitter Holding register 50
Transmitter Shift register 50
TxD 73
TXD2 20
TXD3 19

U

units of measure used 15

W

write cycles 34

X

Xflag 19
XOFF 55
XON 55

Z

ZDR 79