CD1865

Intelligent Eight-Channel Communications Controller

Datasheet

Product Features

- Eight full-duplex asynchronous channels supporting data rates up to 115.2 kbps

  Note: To support this data rate, the specified system clock frequency is required.
- Register-based interrupt acknowledges eliminate need for separate interrupt acknowledge signals
- Automatic prioritizing scheme allows device to respond to an interrupt acknowledge with the highest internal interrupt pending (host-programmable)
- Sophisticated interrupt schemes
  - —Vectored interrupts
  - —Fair Share interrupts
  - —Good Data™ interrupts for improved throughput
  - —Simultaneous interrupt requests for three classes of interrupts: Rx, Tx, and modem state changes
- Independent baud-rate generators for each channel/direction
- Software compatibility with the CD180 and CD1864 devices
- Generation and detection of special characters
- Automatic flow control
  - —In-band (Xon, Xoff generation, and detection)
  - —Out-of-band (DTR/DSR or RTS/CTS)
- On-chip FIFO — 8 bytes each for Rx, Tx, and Status
- Line break detection and generation
- Multiple-chip daisy-chain cascading feature
- Odd, even, forced, or no parity
- modem/general-purpose I/O signals per channel
- System clock up to 66 MHz (x2), 33MHz (x1)
- CMOS technology in 100-pin MQFP

As of May 18, 2001, this document replaces the Basis Communications Corp. document *CL-CD1865 — Intelligent 8-Channel Communications Controller*. May 2001

Datasheet

Contents

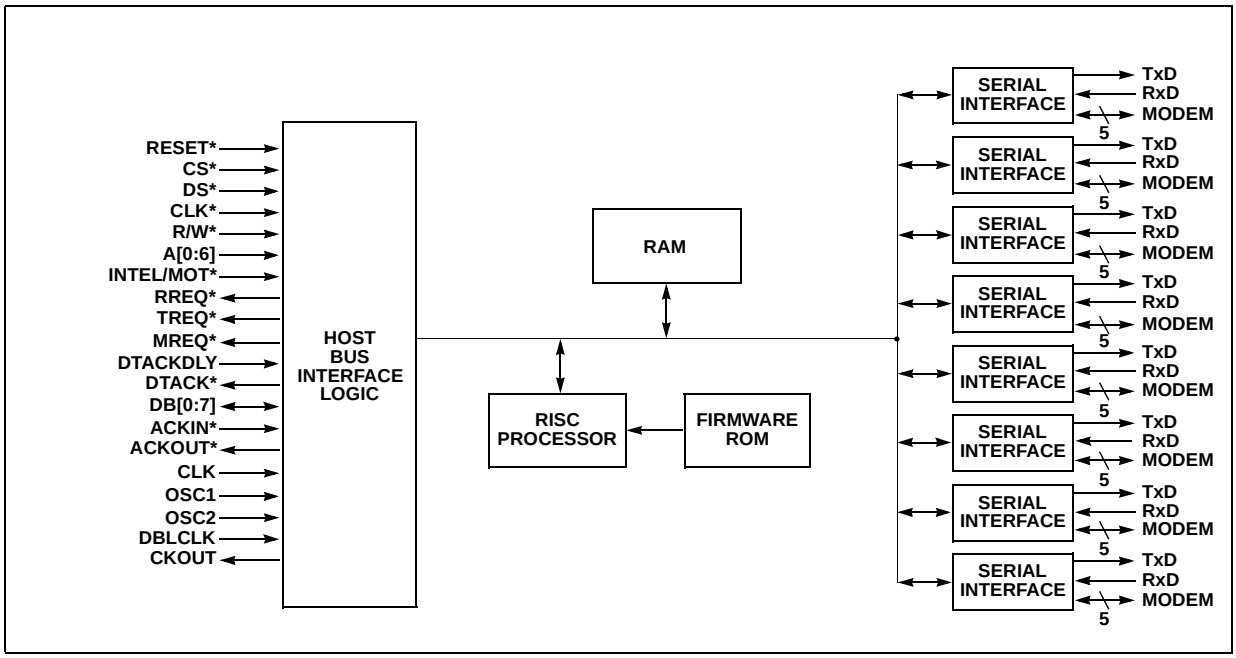
Figures

Tables

Revision History

Revision	Date	Description
1.0	May 2001	Initial release.

Figure 1. Functional Block Diagram



1.0 Overview

The CD1865 is a cost-effective controller capable of controlling eight full-duplex channels transferring data at rates up to 115.2 kbps. The advantage of the CD1865 lies in its ability to efficiently move data from the serial channels to the host. This results in an order-of-magnitude improvement in system-level throughput and a reduction in overhead on the host CPU.

To increase the overall data throughput of the system, the device relies on a combination of features. Most important are the buffers for transmit and receive data. Each serial channel has three 8-byte FIFOs — one each for transmit, receive, and receive exception status. The receive FIFOs have programmable thresholds to minimize interrupt latency requirements.

The CD1865 is based on a high-performance proprietary RISC processor architecture developed by Intel specifically for data communication applications. This processor executes all instructions in one clock cycle, and uses a register-window architecture to ensure zero-overhead context switch for each type of internal interrupt.

The CD1865 is fabricated in an advanced CMOS process. The device's high throughput, low-power consumption, and high-level of integration permit system designs with minimum parts count, maximum performance, and maximum reliability.

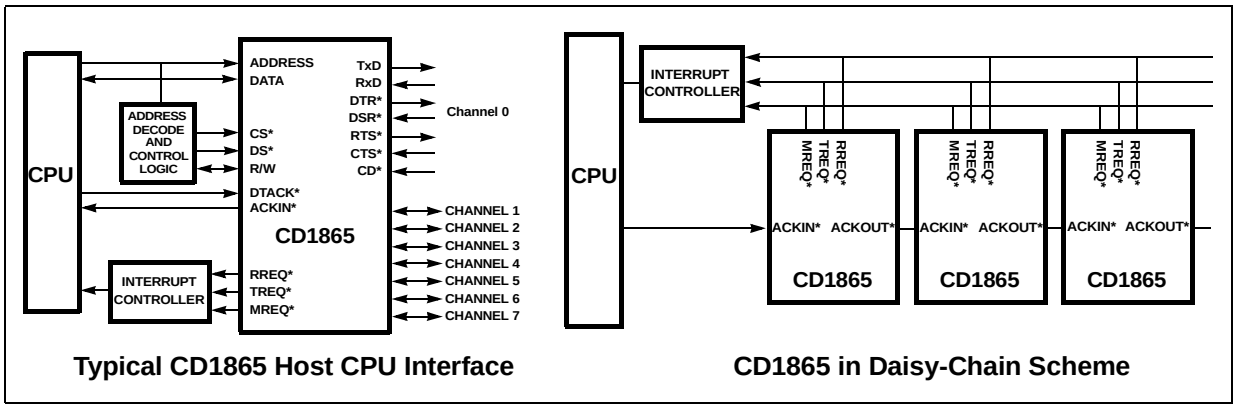
1.1 Theory of Operation

The CD1865 custom RISC processor is assisted by specialized peripheral logic. Serial data transmission and reception is handled by 'bit engines'. Each channel has a bit engine for transmit and another for receive. While each engine handles all bit-level timing, bit-to-character assembly is done in firmware. Bits are passed to the processor by internal interrupts, over a special bus dedicated to this purpose. To reduce internal interrupts to zero, special interrupt context hardware points to the correct register window for every possible context. A unique Global Index register eliminates address calculations by always pointing to the current channel.

The processor assembles bits into characters, checks parity and other formatting parameters, and stores the data in the FIFOs as required. FIFOs are maintained as RAM-based structures. Both the local processor and the host access them by Pointer registers, in effect an Indexed Addressing mode.

The CD1865 communicates with the host by service requests and service acknowledgments. Service requests can be detected by interrupt lines or by on-device registers. Regardless of the method used, the CD1865 has features to minimize both the number of requests to be serviced and the time required to service them. FIFOs help reduce the number of service requests to one every eight characters. To reduce the time required per request, the CD1865 supplies separate vectors for four different types of service requests. This reduces the time required by the processor to effect the proper operation. For instance, there is a unique vector for 'good data', so that the host wastes no time checking status bits or error conditions. If there is an error condition, the CD1865 supplies a unique vector pointing to the error-handling routine. Other vectors report transmit status and modem signal change.

Interrupts can be acknowledged either by an Interrupt Acknowledge pin, or by reading an on-device register. This allows host software maximum flexibility and speed in handling service requests.



Because the CD1865 RISC processor is processing every character sent or received, features such as automatic flow control and special character recognition are easily implemented. This further reduces the processing burden on the host system. Both In-Band (Xon, Xoff) and Out-of-Band (RTS/CTS, DTR) Flow-Control modes are supported. For in-band flow control, the CD1865 automatically starts and stops its transmitter when the remote unit sends flow-control characters. The CD1865 also makes it easy for the local host to flow-control the remote, by the ‘send special character’ commands. For out-of-band flow control, the transmitter optionally asserts RTS and monitor CTS for permission to send; and assert/negate DTR when the Receive FIFO reaches a user- definable threshold. Together, the in-band and out-of-band features not only allow the data flow to be controlled in real time with minimum or no host intervention, it also prevents loss of data.

As shown on the previous page, the CD1865 can interface virtually any CPU, with a minimum of glue logic. Refer to the CD1865 Data Sheet for detailed information on how to interface various microprocessors. Systems with multiple CD1865s are easily implemented, with no external glue, by device a daisy-chain scheme. A ‘fair share’ feature ensures equal access for all service requests, both within one CD1865 and across multiple devices.

FIFO — 24 bytes of FIFO are dedicated to each channel partitioned as 8 bytes for transmitter, 8 bytes for receiver, and 8 bytes for status. The receive FIFO has a user-programmable threshold to optimize system response and latency. The receive FIFO threshold programming range is from 1–8 characters.

Vectored Interrupt Structure — Three interrupt signals ([R, T, M]REQ*) are used. These signals may also be read as an on-device register. Each REQ* signal represents one of three interrupt groups: receive, transmit, and modem signal state changes. Upon servicing by the host, an interrupt vector is generated by the CD1865 to define the interrupt group to be serviced and which CD1865 generated the interrupt. This allows the host software to enter directly into the proper interrupt service routine, reducing the amount of interaction between the host and the controller, and determining the nature of the interrupt.

Good Data Interrupt — If data received is all good, the host is advised of the number of good data bytes in the FIFO, allowing the host to read data without further status queries until all good data has been transferred.

Fair-Share Interrupt Scheme — To ensure equal service of all channels, a fair share scheme is used for each interrupt group. No channel can interrupt for the same condition until all others have a chance to be serviced for the same interrupt condition.

Note: To support 115.2 kbps, a system clock of 66 MHz is required. System design is simplified in the CD1865 by providing a choice of crystal or external clock operation, at 1×- or 2×-rated frequency.

2.0 Conventions

2.1 Abbreviations

Symbol	Units of measure
°C	degree Celsius
µF	microfarad
µs	microsecond (1,000 nanoseconds)
Hz	hertz (cycle per second)
Kbit	kilobit (1,024 bits)
kbps kbits/second	kilobit (1,000 bits) per second
Kbyte	kilobyte (1,024 bytes)
kbytes/second	kilobyte (1,000 bytes) per second
kHz	kilohertz
kΩ	kilohm
Mbyte	megabyte (1,048,576 bytes)
MHz	megahertz (1,000 kilohertz)
mA	milliampere
ms	millisecond (1,000 microseconds)
ns	nanosecond
pV	picovolt
V	volt
W	watt

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
AC	alternating current
CMOS	complementary metal-oxide semiconductor
DC	direct current
DMA	direct-memory access
DRAM	dynamic random-access memory
FIFO	first in/first out

Acronym	Definition (Continued)
HDLC	high-level data link control
ISA	industry standard architecture
LSB	least-significant bit
MSB	most-significant bit
PPP	point-to-point protocol
MQFP	metric quad flat pack
RAM	random-access memory
R/W	read/write
SDLC	synchronous data link control
TTL	transistor-transistor logic

3.0 Device Selection Considerations

The CD1865 device is an enhanced version of the same product family as the CD180 and CD1864. The CD1865 is software compatible with both the CD180 and CD1864. If this is a new CD1865 design, please skip this page.

The CD1865 is recommended for any new designs. Please note that to achieve the high data rates, 66-MHz system clock is required. To support data rates of up to 115.2 kbps, the specified system clock frequency is required. Please refer to the differences in pins between the CD1864 and CD1865. It is recommended that the 66-MHz, 2×-clock option (oscillator or crystal) is used wherever possible.

Table 1. CD18XX Product Family

Features	CD180	CD1864	CD1865
Package	84-pin PLCC	100-pin PQFP	100-pin MQFP
System clock	12.5 MHz	25 MHz (×2) or 12.5 MHz (×1)	66 MHz (×2) or 33 MHz (×1)
Maximum bit rates	64 kbps	64 kbps	115.2 kbps
Pins	DTRSEL *	–	–
	4 modem/IO signals per channel	5 modem/IO signals per channel	5 modem/IO signals per channel

Note: This input (DTRSEL) on the CD180 sets the mode for the DTR*/CD* pins. When DTRSEL is high, the DTR*/CD* pins implement the DTR* output; when low, the DTR*/CD* pins become CD* inputs.

The CD1864 and CD1865 have separate DTR and CD pins and so the DTRSEL is eliminated.

Table 2. Differences Between the CD1865 and CD1864

Pin Number	CD1865 Pin Name	CD1864 Pin Name	Comments
1	V_{CC}	n/c	Note 1
16	GND	n/c	Note 1
37	V_{CC}	n/c	Note 2

NOTES:
1. Pin 1 and pin 16 are truly no-connects on the CD1864 device.
2. Pin 37 on the CD1864 is not a true no-connect, and can cause problems if connected to V_{CC}. To make a single board design be compatible with either the CD1864 or CD1865, a configuration jumper must be used to allow pin 37 to be a no-connect or a V_{CC} connection.

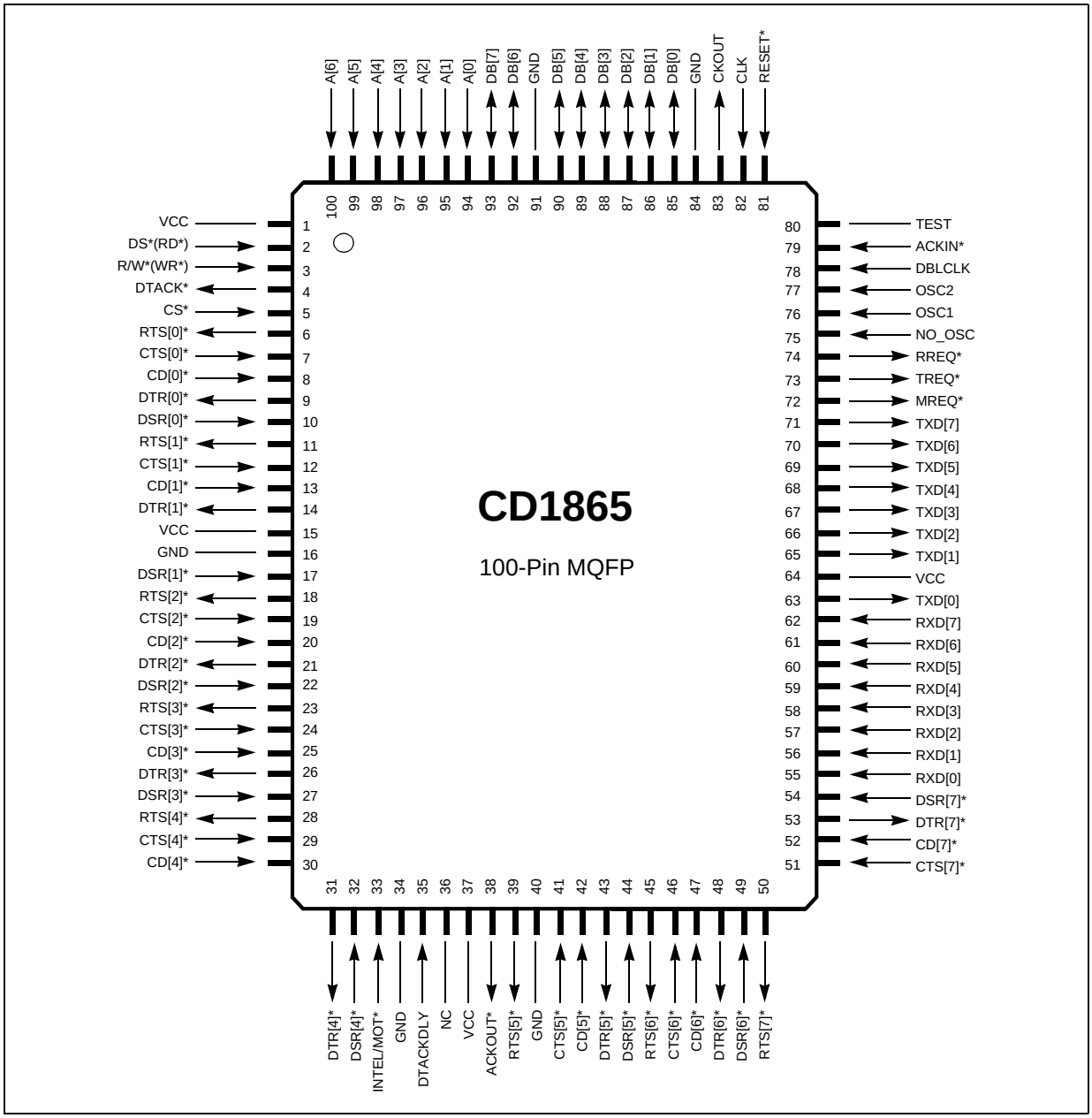
Note: In January 1995, Intel changed all 100-pin PQFP package types from EIAJ to JEDEC. The CD1865 is now available in a JEDEC package. Before beginning any new design or converting from CD1864 to CD1865, please contact Intel for package details.

Warning: The CD1865 device may have potential latch up problems if used in socket. It is recommended that this device be surface mounted.

4.0 Pin Information

The CD1865 is available in a 100-pin MQFP (metric quad flat pack device) configuration as shown below.

4.1 Pin Diagram



NOTE: (*) Denotes an active-low (negative-true) signal.

4.2 Pin Assignments

The following conventions are used in the table below: (*) denotes an active-low signal; I = input; I/O = input/output; O = output; OD = open drain; a (:) indicates decending pin numbers; a (–) indicates ascending pin numbers.

5.0 Functional Description

5.1 Introduction

The CD1865 I/O coprocessor controls eight full-duplex channels that transfer data at rates up to 115.2 kbps. The CD1865 efficiently moves data between the serial channels and the host, resulting in a great improvement in system-level throughput and a reduction in overhead on the host CPU. This improvement is obtained by reducing the number of service requests (interrupts) the host must respond to and reducing the complexity and time required to handle each service request.

The CD1865 relies on a combination of features to reduce the number and complexity of service requests. Most important are the buffers for transmit and receive data. Each serial channel has three 8-byte FIFOs — one each for transmit, receive, and receive-exception status. The Receive FIFOs have programmable thresholds to minimize interrupt latency requirements. The vectored service requests and the Good Data™ interrupt allow the host system to immediately transfer data upon beginning processing of a service request, without tedious checking of flags and error conditions.

The CD1865 is based on a high-performance, proprietary RISC processor architecture developed by Intel specifically for data communications applications. The CD1865 processor executes all instructions in one-clock cycle, and it uses a register window architecture to ensure zero-overhead context switch for each type of internal interrupt. The instruction set of this processor is optimized for bit-oriented tasks that combined with instantaneous response to sending or receiving one bit, allow highly efficient processing of characters. All firmware for the CD1865 processor is contained in an on-device ROM, and requires no user programming.

The CD1865 processor is assisted in its task by specialized peripheral logic. Serial data transmission and reception is handled by ‘bit engines’. Each channel has a bit engine for transmitting and another for receiving. While each engine handles all bit-level timing, bit-to-character assembly is done in firmware. Bits are passed to the CD1865 processor by internal interrupts over a special bus dedicated to this purpose. Special internal-interrupt context hardware reduces overhead on internal interrupts to zero by pointing to the correct register window for every possible context, and a unique Global Index register eliminates address calculations by always pointing to the current channel. External service requests to the host system are also hardware assisted. There is a queue for each of the three classes of external service requests, and the request/acknowledgment mechanism is entirely in hardware to minimize response time.

The CD1865 processor assembles bits into characters, checks parity and formatting parameters, and stores the data in the FIFOs as required. FIFOs are maintained as RAM-based structures, and both the local CD1865 processor and the host access them by Pointer registers by an Indexed Addressing mode.

The CD1865 communicates with the host by service requests and service acknowledgments. Service requests can be handled either as interrupts or by polling. Regardless of the method used, the CD1865 has features to minimize both the number of requests to be serviced and the time required to service them. The number of service requests is reduced by the FIFOs since a service request is required only every eight characters. To reduce the time required per request, the CD1865 supplies separate vectors for four different types of service requests. This reduces the time required by the host CPU to determine what action to take. For example, there is a unique vector for Good Data so that the host wastes no time checking status bits for error conditions. If there is an error condition, the CD1865 supplies a unique vector pointing to the error-handling routine. Other vectors report transmit status and modem signal change.

Service requests to the host system are implemented on the CD1865 by three hardware service request state machines. Each machine has the ability to ‘queue-up’ multiple requests. The state machines are designed to offer the fastest response possible. Whenever the CD1865 processor determines that a condition needs a service request, it queues the request with the appropriate state machine. The state machine posts the external request, monitors acknowledgment cycles from the host, and informs the CD1865 processor when a valid service acknowledgment has been completely serviced. This allows the CD1865 to correctly maintain the internal context for processing the channel being serviced.

Because the CD1865 processor processes every character sent or received, features such as Automatic Flow Control and Special Character Recognition are easily implemented. This reduces the processing burden on the host system. Both In-Band (Xon, Xoff) and Out-of-Band (RTS/CTS, DTR/DSR) Flow Control modes are supported. For In-Band Flow Control, the CD1865 automatically starts and stops its transmitter when the remote unit sends flow-control characters. The CD1865 makes it easy for the local host to flow-control the remote by the ‘Send Special Character’ commands. For Out-of-Band Flow Control, the transmitter optionally asserts RTS and monitors CTS for permission to send, and assert/negate DTR when the Receive FIFO reaches a user-definable threshold. DSR can be used to gate the receiver on and off. Together, the In-Band and Out-of-Band features allow the data flow to be controlled in realtime with minimum or no host intervention, and this also prevents loss of data.

Systems with multiple CD1865s are easily implemented, with no external glue, by a daisy-chain scheme. A fair-share feature ensures equal access for all service requests, both within one CD1865 and across multiple devices. Alternately, multiple CD1865s can be operated in parallel as independent devices.

Serial channels on the CD1865 are entirely independent of one another. Any channel can be programmed to a combination of features regardless of the state of other channels. Bit-rate generators are programmed by loading a divisor value, so the transmitters and receivers can each operate at any standard or non-standard data rate.

The CD1865 can detect the received line-break condition, send break characters of any length, and transmit delays. This is done by transmit commands embedded in the Transmit Data Stream. The CD1865 can also be programmed to detect user-defined special characters and generate a special service request to the host. Parity checking is performed automatically, but can be overridden by the host to force parity errors for test purposes. Character length and Stop bit length are also programmable per-channel.

Modem pins on the CD1865 are general-purpose, that is, they are not hard-wired into the UART functions. If modem pins are not needed to interface to actual modems, they can be used as general-purpose I/O pins. In either case they are readable and writable directly by the host system. In addition, the CD1865 can be programmed to monitor levels on modem input pins and generate service requests to the host upon detecting a specified change.

The CD1865 is fabricated in an advanced CMOS process. Its high throughput, low-power consumption, and high level of integration permits system designs with minimum parts count, maximum performance, and greater reliability.

There is a significant difference between the CD1865 and conventional dumb UARTs; the CD1865 is more efficient and intelligent, even when operating in a polled environment. Systems built with the CD1865 interface between the host and the I/O device at a higher level than systems built with conventional UARTs. For example, with a dumb UART, the host must test each channel for presence of data, a process that is time-consuming. With the CD1865, the host queries the entire

serial I/O subsystem for the presence of data. If data is present, the CD1865 determines which channel it is on, and whether it is good or erroneous. Thus, using the CD1865, the host-peripheral interface is easier to implement, faster, and more efficient.

5.2 Internal Operation

The internal architecture of the CD1865 is shown in Figure 2. The foundation of the design is a custom-designed CPU that Intel has developed especially for this application. This CPU is optimized for bit-oriented tasks associated with UART functions, and it has a set of registers for each channel, arranged in a register window architecture. These registers and the ALU are eight bits wide. The CD1865 processor has a 16-bit instruction word that it retrieves from an on-device ROM. Every instruction is one-word long and executed in one-clock cycle.

Whenever an internal interrupt occurs (from a bit engine), the CD1865 processor automatically switches context to that channel's block of registers. No time is lost in saving any machine state. The CD1865 processor executes the instructions necessary to handle that bit (typically three to six instructions) and then returns to the context it was in prior to the internal interrupt. All internal interrupts are at the same priority level; the interrupt handler block ensures fair-share access across channels.

Each channel's serial interface logic consists of a receive-bit engine, a transmit-bit engine, a receive-baud-rate generator, a transmit-baud-rate generator, and a timer. The receive-bit engine samples the state of the RxD pin at the time indicated by the receive-baud-rate generator, and it reports this value to the CD1865 processor as an interrupt. The transmit-bit engine works in a similar manner. At the baud rate tick, it outputs the next bit and generates an interrupt to the CD1865 processor requesting the following bit.

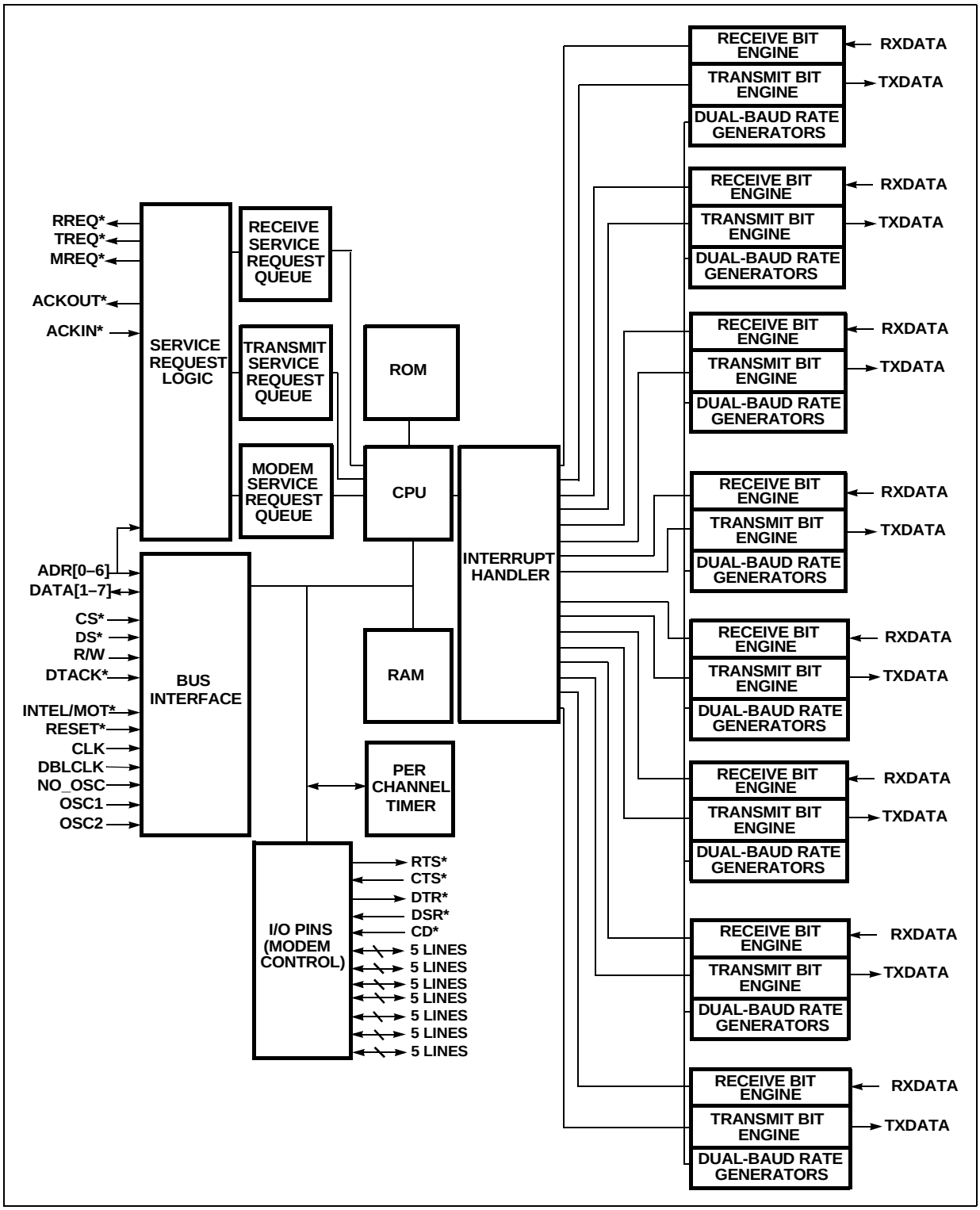
The baud-rate generators are 16-bit dividers that operate from a master clock, which is the system clock divided by 16. All baud-rate generators are independent, so a channel can send and receive at any speed. In addition to the baud-rate generators, there are two channel timers for each channel. One is an 8-bit divider, operating off the master prescaler timer tick. This timer is used to time-out partially full FIFOs to avoid 'stale' data. The other is used to time embedded delays in the transmit data stream.

All eight channels are continuously scanned by internal logic that generates interrupts to the CD1865 processor in a 'fair' manner. This fair-share interrupt feature is the same as the mechanism used to share service requests across multiple devices. Whenever two or more channels are contending for interrupt service, the channel that is serviced first does not assert again until all other currently pending channels are serviced. This prevents a fast, 64-kbps channel from demanding service from a slow 1200-bps channel, yet it allows the faster channel the additional service it needs to support its higher speed. This allows more overall throughput than a 'round-robin' or an 'equal-access' method would provide.

Service requests for the host are handled by fast, dedicated logic on each of the three levels provided. Whenever the CD1865 processor detects a condition requiring external-host service, it queues the request with the service-request machine for that level. This machine asserts the External Request pin, and it monitors for a service acknowledgment of the same level. When a service acknowledgment is sensed, the machine automatically provides the vector to the host and sets up the internal context of the CD1865 for service. Upon completion of the service, the machine restores the normal context. The queue for service requests is two deep, so in a busy system there can be another request immediately pending when the first one is completed. This method avoids any delay between requests, and improves overall efficiency.

Modem I/O signals are implemented as ‘conventional’ input-output circuits, readable, and writable by either the on-device or the host CPU. This allows maximum flexibility in using these signals either in the conventional way, or for any other I/O function required. When the CD1865 processor is using these pins to implement flow-control functions, it reads them under software control and implements the function that way. There is no direct hardware association between the modem pins and the serial I/O hardware.

Figure 2. Internal Block Diagram



The CD1865 workload can be divided into two categories:

- Bit-to-character conversion (and vice versa) — the ‘traditional’ UART function
- Character-level processing such as flow control, FIFO management, and host interface functions

The CD1865 internal processor handles all these tasks in firmware. A foreground/background scheme is used: foreground for internal bit-engine interrupts and background for everything else. This internal structure represented in Figure 3 on page 24, shows how the foreground communicates with the background. Foreground code handles bit-to-character assembly for receive, and character-to-bit disassembly for transmit. In either case a Holding register, together with a Full/Empty bit, acts as the ‘gateway’ between the interrupt-driven foreground and the polling-loop background code.

The background code executes the polling loop as shown in Figure 4. After power-on reset, the software runs continuously in an inner and an outer loop. Lower-priority tasks are handled in the outer loop, and higher-priority tasks are handled in the inner loop. The highest-priority tasks are bit events that are handled by foreground (that is, interrupt-driven) code.

The inner loop executes eight times as often as the outer loop. It checks each channel’s Full/Empty bits to sense if another character needs to be moved. It first checks receive, and if there is a character to be moved, it is moved and execution moves on to the next channel. If receive data does not need processing, then transmit is checked. This mechanism gives a slightly higher priority to receive than to transmit, and is favorable because missing a receive character is a fatal error and being late in transmitting one is not an error. (The effect of this can be observed by programming the CD1865 for higher-than-rated serial baud rates and providing a source of receive traffic with virtually 100-percent loading. As the CD1865 is heavily loaded, it leaves short gaps between transmit characters because the firmware is following the ‘receive’ path through the code. Refer to Section 6.2.5 for details on maximum performance and maximum line speed).

After eight passes through the inner loop (for example, checking all eight channels for data), one pass is made through the outer loop. This pass checks one channel for host commands (such as ‘Send Special Character’), timer functions, and a condition that requires posting an external service request (for example, Receive FIFO full, Transmit FIFO empty, modem signal change, and so on). If required, the firmware posts the service request within the queue of the appropriate service-request logic. It then continues normal operation, until the host responds to the service request. After a single pass through the outer loop, eight passes through the inner loop are again made.

In most cases the CD1865 checks the appropriate bit in RAM to determine which options are enabled and then modifies its processing accordingly. Some control bits must be interpreted and moved by the CD1865 firmware from their location in Option Bit registers to other locations in the device. Therefore, the host must notify the CD1865 when these bits are modified. Then, the CD1865 alters the channel as commanded. For details on channel command functions, refer to Section 7.2.

Figure 3. Foreground/Background Internal Structure

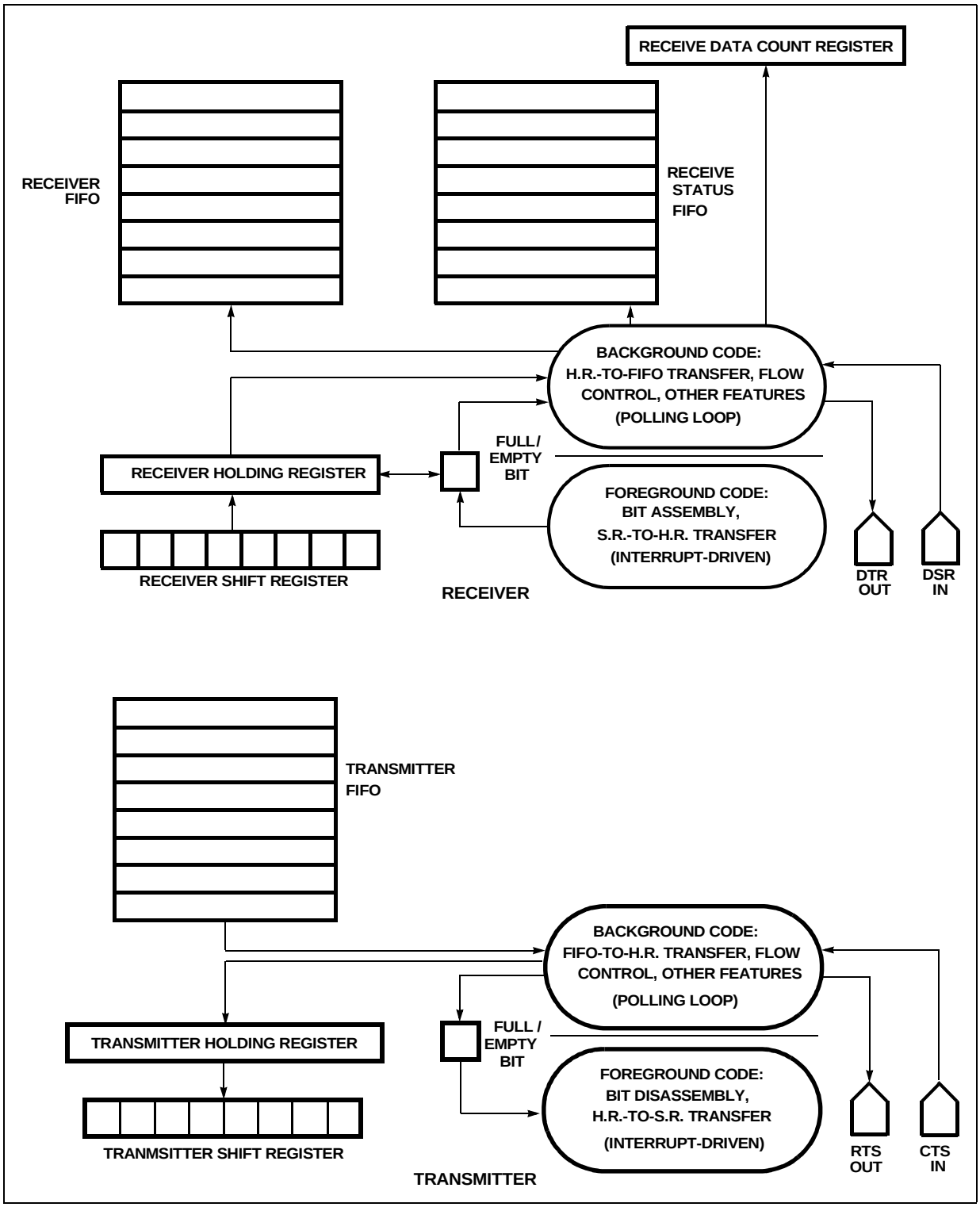
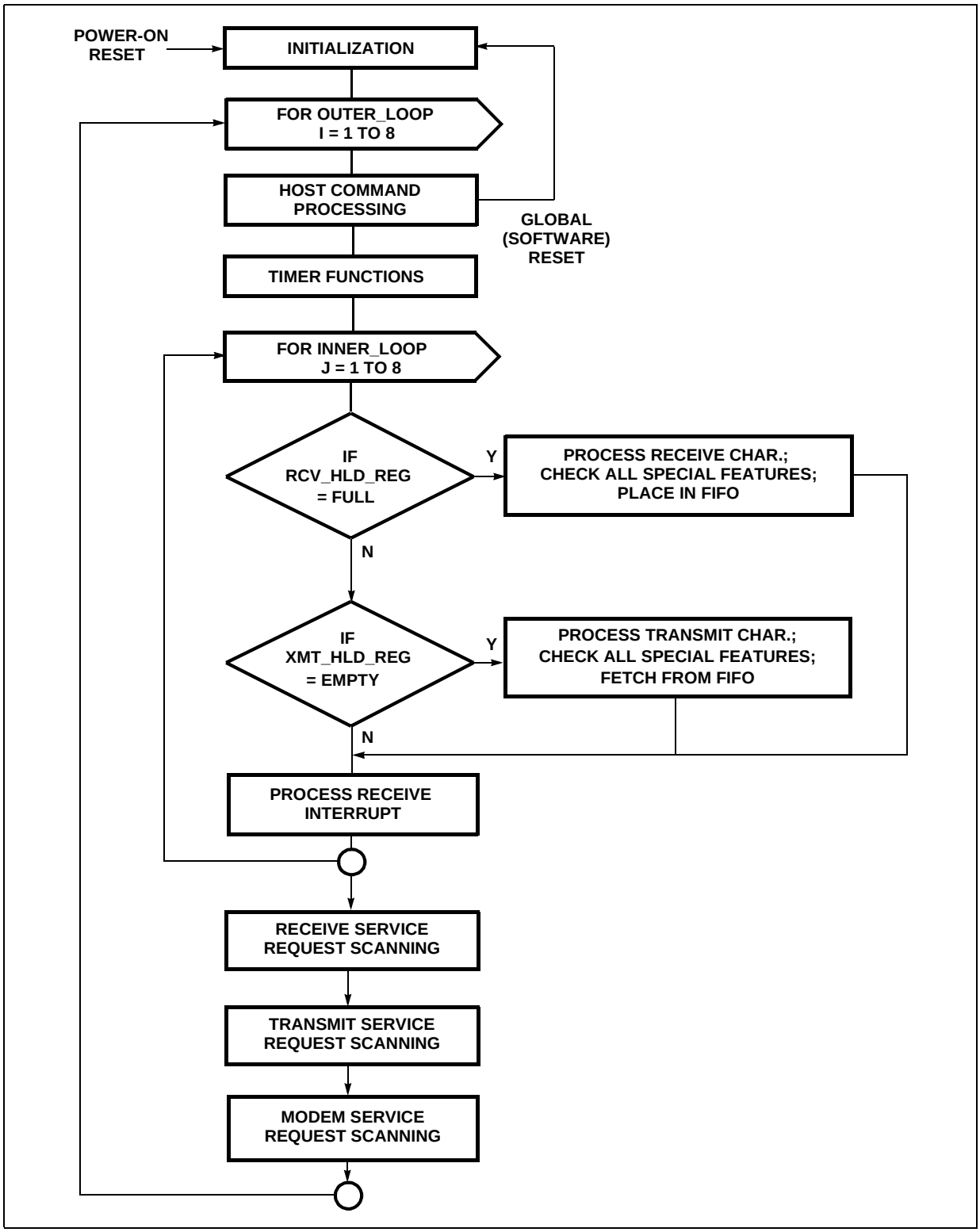


Figure 4. Internal Operation Flow Chart



5.3 Service Request and Interrupt Operation

The CD1865 enhances design efficiency, because it is an intelligent device that more closely resembles an add-in controller board than a mere collection of TTL. Conventional UARTs are basically passive, ‘dumb’ logic. For example, when polling a device for channels requiring service, each channel is not individually tested. Because of this, certain restrictions are placed on when and how FIFOs are accessed. The CD1865 processor must determine what the host is doing, and when to manage the queue of events correctly and efficiently.

Interrupt-Driven Versus Polled

Choosing the software interface, interrupt-driven versus polled, is critical to overall system performance. This choice also affects how the software is written. In hardware implementation, a programmer has a choice of Mixed mode, that is, when to poll versus when to be interrupt-driven. Mixed-mode operation allows a programmer to optimize the efficiency of the system according to changing needs. The advantages of each method are discussed in Section 5.5.

5.3.1 Theory of Operation

The CD1865 has three independent service request levels, one for each of the three categories — Receive, Transmit, and Modem signal change. The priority of these lines is not fixed, but can be determined in one of the following three ways:

- It can be set within the CD1865 by the AutoPriority Option bits.
- A system designer can assign priorities by the manner in which the three service request lines are connected to the host interrupt controller.
- Under software control, the host system can define and redefine the order of service requests.

The Service Request interface to the host is implemented with five signals — *, *, *, ACKIN*, and ACKOUT*. *, *, and * are asserted when a service request is pending; ACKIN* is asserted during service-acknowledgment cycles; and ACKOUT* is used in multiple CD1865 designs to share service requests and daisy-chain acknowledgments.

Whenever the CD1865 processor determines that one or more channels need service from the host, it loads the appropriate service-request state machine with the information about the type of request. The service-request state machine for that level then asserts its request signal. Note that all three request signals can be active at the same time. At this point, the CD1865 has not determined which request should be handled first — it simply asserts any and all lines, as required by the status of various channels. (This is true even if the AutoPri Option is enabled; AutoPri takes effect when a service request is acknowledged, and at that time the CD1865 determines which is the most important request.)

The host, after noticing that one or more of the three service request pins are active — either because the host is interrupted or it polled an external or internal CD1865 status register — decides which of the requests (if more than one is active) it services first. The host begins the service operation by issuing a Service Acknowledge cycle. The purpose of this cycle is to cause the CD1865 to set up its internal state for that type of request. (Note that if AutoPri is set, it is not necessary for the host determine which level of service request to acknowledge; it simply acknowledges the CD1865 request and the CD1865 returns the vector for the highest-priority active request.)

If AutoPri is not being used, the CD1865 needs to be informed which one of the three possible pending requests the host wants to acknowledge. There are two different ways CD1865 can be informed of this — hardware and software.

The hardware method is based on the value in the address bus. The CD1865 determines the type of request being acknowledged by the value placed in the address bus during the acknowledge cycle. This is the method used by Motorola®-family processors. The host places the level of interrupt being serviced on the low-order address bits during an interrupt acknowledgment cycle. When the host performs a Service Acknowledge cycle, the CD1865 compares the value on the address bus with the three unique values stored in three internal registers — the . These values are set by the user at system initialization. A match occurs on only one of these registers, and this informs the CD1865 of the type of request being acknowledged.

In most circumstances the address bus should not have a value that does not match one of the three values during an acknowledgment cycle. This causes the CD1865 to not recognize that any bus cycle is occurring, and it does not assert DTACK*, or terminate the cycle, or take any other action. Doing this does not affect the CD1865, but the system must have some other provision to terminate the bus cycle. If, for example, the CD1865 shares an interrupt level with another device, different values on the address bus should be used to control responses to an acknowledgment, but the bus cycle should terminate in a usable way.

Service acknowledgments can also be performed by software. The host simply reads one of three Request Acknowledge registers, and the CD1865 performs as if a hardware service acknowledge cycle is executed.

Regardless of the method of acknowledgment used, within the CD1865, each service request state machine makes the following determination: if it has an internal service request pending, and there is a service acknowledge of the same type, it asserts its internal-acknowledge-accepted signal back to the Service Request Controller logic, negates the Service Request Output pin, and holds its acknowledge-out daisy chain in a negated state. It also drives the value in the Global Vector register (GVR) onto the data bus, for the host to read as part of the Service Acknowledge cycle. The GVR value placed on the bus during the Service Acknowledge cycle serves two purposes. The least-significant three bits of GVR indicate which of the four types of service requests are occurring. The upper-five bits are user-defined and serve to identify, in daisy-chained CD1865 systems, which of the multiple CD1865s is active.

If the service request state machine does not have a service request pending, and there is a software acknowledgment or address bus match, it passes the service acknowledgment down the chain by asserting ACKOUT*. If there is no match, the state machine remains idle.

If a service request is pending and the Receive Service Request is to be handled, the CD1865 is notified because the three have different values in them; therefore, only one match (receive service, in this case) occurred. The internal grant from the service request state machine causes the receive service type code and active channel number (previously stored at the time the request is posted by the CD1865 processor) to be pushed onto the service request stack. This automatically causes the FIFO pointers to be set up for the active channel, with no host intervention.

The host, at this point, has all the information needed to handle the service request. It determines the exact type of service being requested (Transmit, Receive Good Data, Receive Exception, or Modem signal change) and which of the multiple CD1865s is requesting service. It gets the channel number by reading the Global Channel register (GCR) and then proceeds to service the request. At the completion of the service, the host performs a dummy write to the CD1865 End Of register (), that causes the CD1865 to exit its internal service request state by popping the service

request stack. At this time the CD1865 is ready to be serviced on another of its outstanding requests. If another request of the same level is pending, two clock periods after the write to are required for the CD1865 to reassert the request line.

Because the CD1865 has a service request stack, it can support nested-service requests. For example, the host can be in middle of a Transmit Service Request, detect that Receive Service Request has asserted, process the Receive Service Request, and after exiting the receive service routine, resume the Transmit Service Request. The CD1865 stack is three deep, so all three types (one of each) can be nested if required. The current service request context (for example, the stack) is readable in the Service Request Status register.

The Global Channel registers (GCR) are actually three registers that provide the number of the channel requesting service. Reading any of these registers causes the CD1865 to mask in three bits, specifying the channel number of the currently active channel. Normally these registers are read by the host when it is handling a service request. In this case, the three bits are the number of the channel requesting service. If any of the three GCR registers are read when the CD1865 is not in a service-request context, the three bits are the current value in the CAR. The current channel number is masked into the contents of bits 4:2 of this register by the CD1865 when it is read by the host. The actual contents of the register are not modified.

These three registers are provided as a convenience to the user. In most applications, the user only uses one of these locations, and set the register to some arbitrary value. However, it may be useful to record information about the state of the CD1865 (or the software driving it) that is associated with each of the three service-request types. In this case, the user can store whatever information is required in the unused bits. Then, when entering a service routine, the software can check these bits to find what state they were left in, and this could be used as a ‘sub-vector’.

5.3.2 Internal Implementation of the Service Request Logic

As discussed above, the heart of each service request level is an asynchronous state machine. This state machine has three inputs:

- MATCH from the Priority Interrupt Level register comparator,
- ACKIN* from the host system, and
- INTERNAL_REQUEST from the CD1865.

Note: Software acknowledgments (reads from the Service Request Acknowledge registers), in effect, force the MATCH value true for their respective level.

It also has three outputs:

- Svc_Req to the host system,
- INTERNAL_GRANT to the CD1865, and
- ACKOUT*, which is combined with the other two ACKOUT* signals to provide ACKOUT* to the next CD1865 in the daisy chain.

Figure 5 on page 30 shows logic implemented by the state machine, which is described in Table 3.

Table 3. State Machine Logic

State Name	Output Condition	Comments

IDLE
IF (INTERNAL_REQUEST = 1)
ELSE IF (ACKIN* = 1 & MATCH =1)
ELSE
	all outputs inactive
	GoTo REQ_ACTIVE
	GoTo PASS_ACK
	Stay at IDLE
		
		; normal 'resting' state
		; pass this acknowledge
		; wait here

REQ_ACTIVE
IF (ACKIN* = 1 & MATCH =1)
IF (ACKIN* = 1 & MATCH =0)
ELSE
	
	GoTo KEEP_ACK
	Stay at REQ_ACTIVE
	Stay at REQ_ACTIVE
		request asserted
		; keep this acknowledge
		; wait here, ACK is for some other level (†)
		; wait here

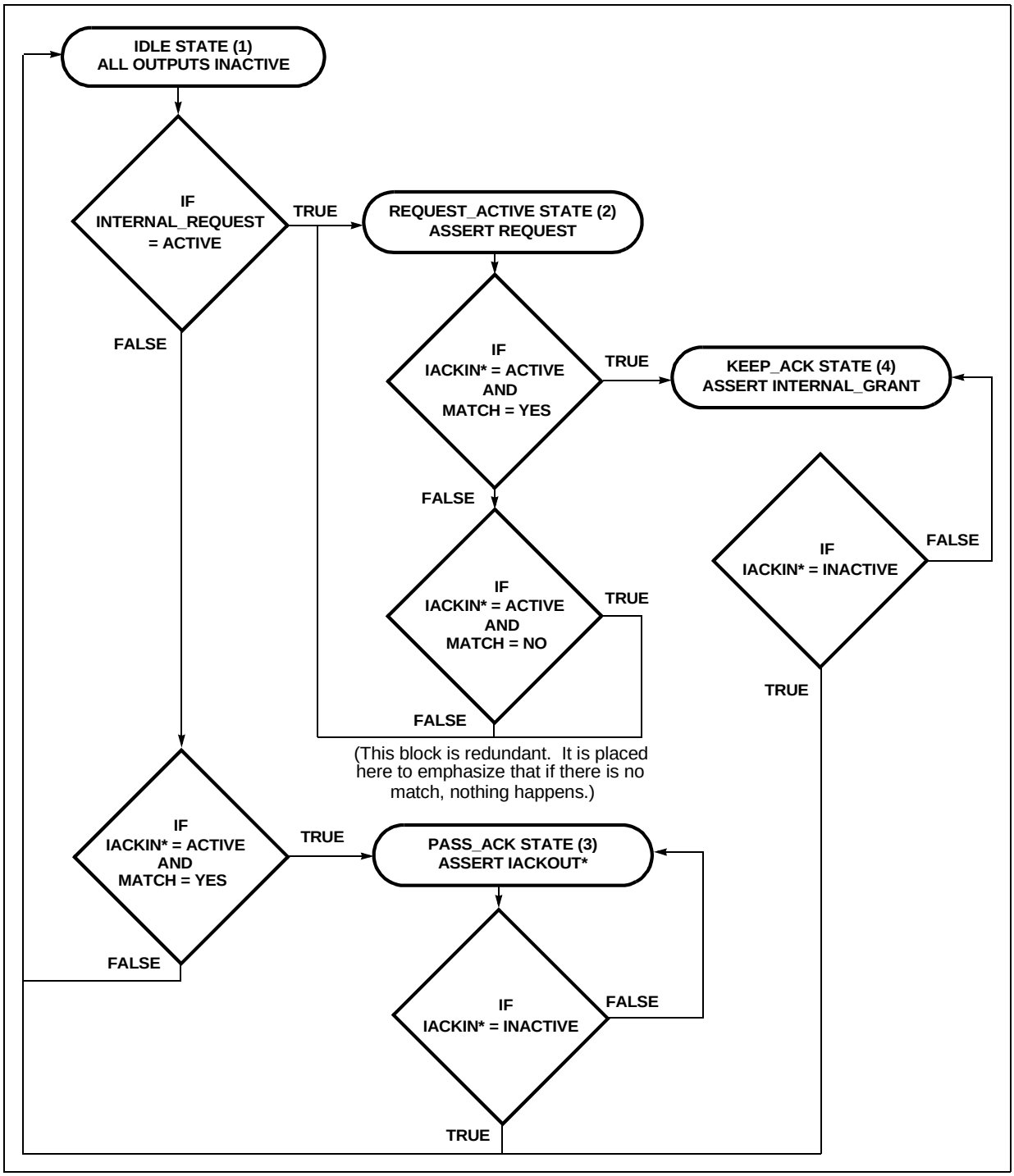
PASS_ACK
IF (ACKIN* = 0)
ELSE
	
	GoTo IDLE
	Stay at PASS_ACK
		ACKOUT* asserted
		; return when ACKIN* is gone
		; wait here while ACKIN* active

KEEP_ACK
IF (ACKIN* = 0)
ELSE
	GoTo IDLE
	Stay at KEEP_ACK
		INTERNAL_GRANT asserted
		; return when ACKIN* is gone
		; wait here while ACKIN* active

NOTE: The (†) denotes the point at which, if there is no match, the CD1865 determines *not* to pass the ACK down the daisy chain. It does this for two reasons: first, it is unacceptable to have the ACKOUT* 'glitch' low; and second, the state machine should be as fast as possible. When the state machine senses an ACKIN* and match is not valid, it cannot conclude that it should assert ACKOUT*; the ACKIN* may be for one of the other two service requests levels. It could wait for the results of the other two MATCH comparators; however, this complicates, and therefore slows down, the response of the state machine.

The reason this complication causes delay is (to implement the logical function 'assert ACKOUT* if no match') it must determine how long to wait before declaring a no-match condition. To implement this delay function, a synchronous state machine is required, which at a 15-MHz clock, means a delay of several hundred nanoseconds from ACKIN* to ACKOUT*, instead of the 65 ns currently specified.

Figure 5. Internal Service Acknowledge Decision Tree

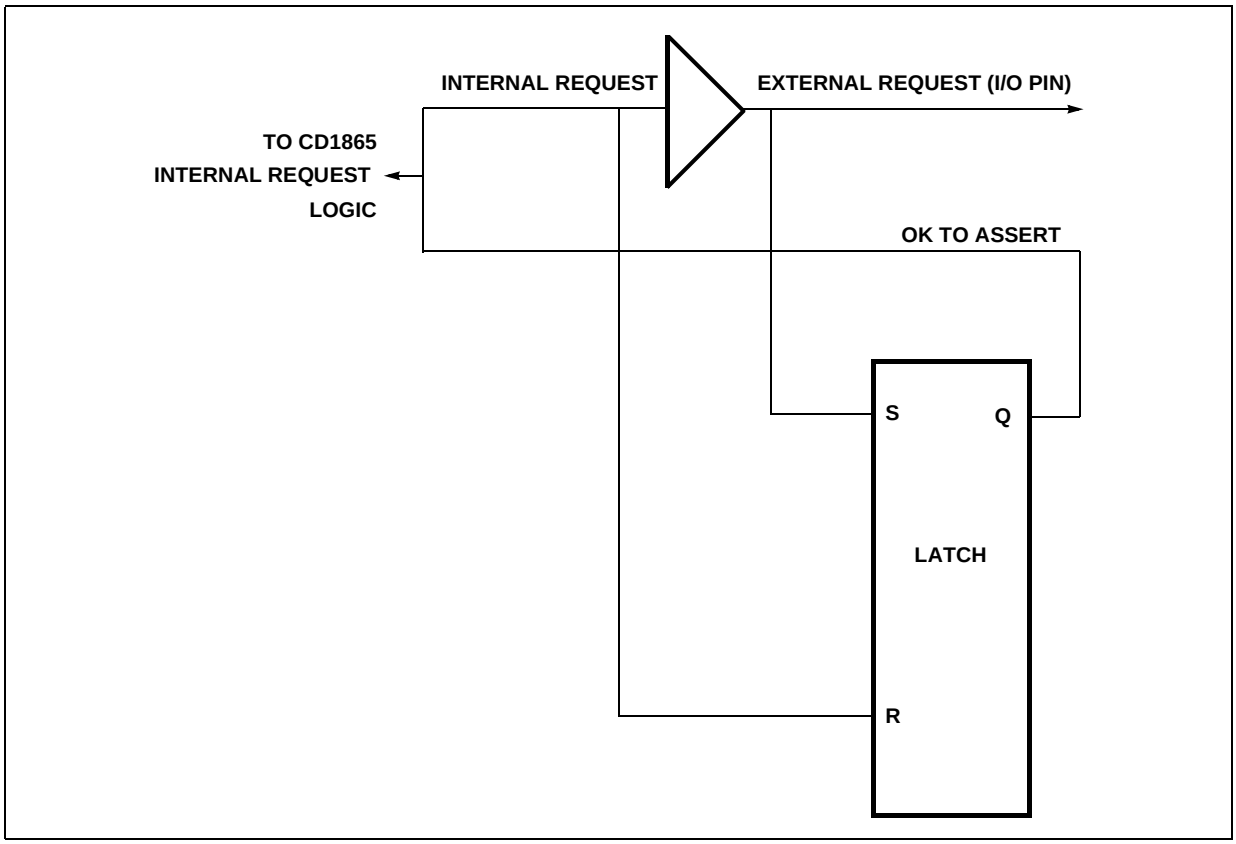


5.3.3 Priorities and Fair Share

The CD1865 implements a fair-share mechanism to ensure that all channels receive equal service, without any ‘data starvation’. Fair share works automatically among the channels in one device and across multiple devices.

Figure 6 on page 31 shows a fair-share operational block diagram. On each of the three service request lines, the CD1865 monitors both the internal and external value of the line. (The external value can differ because, in multiple CD1865 applications, it can be driven by other CD1865s.) At the end of a service acknowledgment bus cycle, the CD1865 checks the state of both request values. If they are different, the CD1865 determines that there is another part also driving the request line, and it does not reassert its own request line until the external request has gone inactive. This inactive level means every other CD1865 with a pending request is serviced; therefore, it is now okay to reassert requests without controlling host bandwidth.

Figure 6. Internal Fair-Share Operation



5.4 Types of Service Requests

The categories of service requests that a CD1865 can generate are explained below. Each channel’s transmitter, receiver, and modem pins require service from the host occasionally; however, each category of service request conditions can tolerate different latencies in being serviced. Conditions for service requests fall into three basic categories:

- Data is received from the remote device and needs to be transferred to the host.

- Data from the host can be given to the Transmitter FIFO, which is now empty.
- A modem signal changes state.

Three separate service request levels are provided to support the following three categories:

Source	Pin Name	Request Match Register Name
Receive data	*	
Transmit data	*	
Modem signal change	*	

5.4.1 Receive Service Requests

The Receive Service Request is unique because it has two subtypes; that is, it is capable of returning one of the two different vectors during a service request acknowledge cycle. The two sub-types are — ‘Receive Good Data’ and ‘Receive Exception’. The reason there are two types within one category of service request is that, while Good Data and Exceptions require different handling, they are both of equal priority, and need to be serviced in the order they are received. For example, suppose two good characters are received, then an exception character, and then another good character is received. There must be a service request for the first 2 bytes of Good Data, then for the Exception, and then for more Good Data. If Exception Service Request is at a different level, the exception character is processed either before or after the Good Data, and not in sequence as it should be. This method also allows the Receive Good Data-handling routine in the host to be very fast and efficient, since it only has to move ‘N’ bytes to a buffer. All special-case conditions can be put in a separate handler, where they do not slow down normal data transfers.

Exception characters are characters with errors or that match the defined special characters, line breaks, and certain time-out conditions.

Data must *not* be read from the Receive FIFO or the Receive Status FIFO except when the CD1865 is within the context of a Receive Data Service Request.

5.4.1.1 Receive Good Data™

A Receive Good Data Service Request is asserted for any of the following three conditions:

1. RxFIFO threshold reached, and the FIFO contains Good Data.
2. RxFIFO threshold not reached, but the FIFO contains Good Data, and the Receive Data Timer times-out.
3. RxFIFO threshold not reached, but the FIFO contains Good Data, and the newly arrived data contains an exception condition.

When any of these conditions occur, the modified service request vector indicates to the host that the service request is for Good Data. The CD1865 continues to add bytes to the FIFO, and it increments the Count register for each good byte added, and this allows for optimally efficient use of the FIFO.

It is not necessary to accept any or all of the Good Data that is available when a Good Data Interrupt is received. If a host buffer is too full to accept 8 bytes, a smaller number (even 0) can be read, the service request context left, and the host buffer handled first. The CD1865 again generates another Good Data Service Request when any of the three conditions listed above are met.

If the condition that caused the request in the first place remains true, the CD1865 quickly generates another service request. If no data is read, this is always the case. If some, but not all, of the available data is read, Conditions 1 and 2 are not true, but Condition 3 may be true if an exception condition is the cause of the Good Data Interrupt. If this becomes a problem, one solution is to temporarily disable receiving interrupts on that channel. To avoid FIFO overflow, do not disable the channel for very long.

5.4.1.2 Receive Exception

Unusual or exception conditions are reported to the host one character at a time through the Receive Exception Service Request. As with normal receive processing, the host determines the requesting channel by reading the GCR. It can then determine the specific exception(s) by reading the Receive Character Status register.

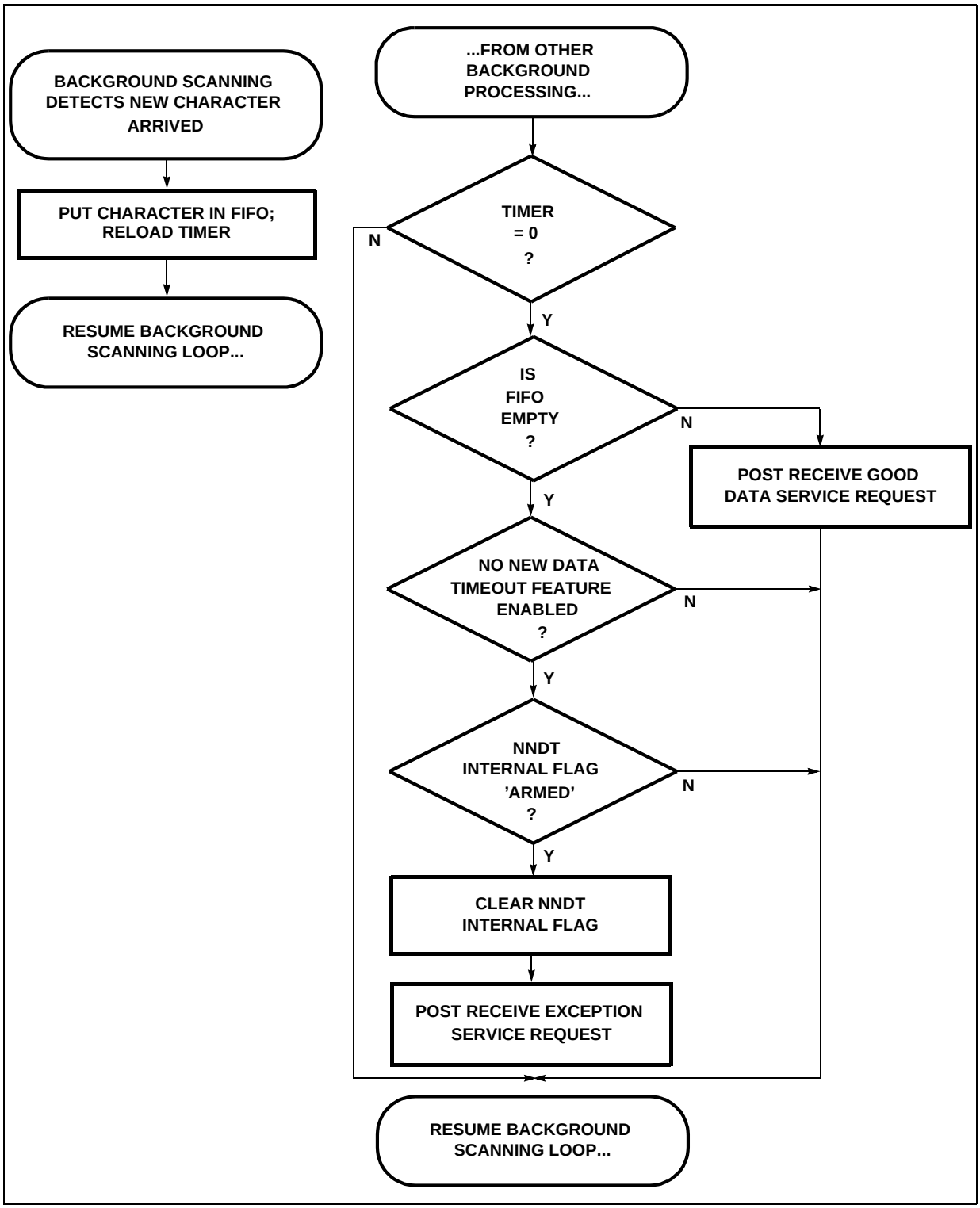
Exception conditions are generated for parity errors, framing errors, FIFO overrun, special character recognition, break detect, and for a special feature called the ‘No New Data Timer’ (NNDT).

NNDT is a receive timer option to generate a service request for the first receive data time-out following the transfer of all data from the FIFO to the host. It is often useful, when managing relatively large I/O buffers, for an I/O processor to determine that ‘no data has arrived lately’. This event is used to transfer the contents of the local buffer that has been storing data from the CD1865 FIFO for host-system processing.

This service request is a receive exception sub-type, and can be used to signal that it is time to transfer the buffer. This feature can be enabled or disabled by controlling the NNDT bit in the Service Request Enable register. As shown in Figure 7, every time a received character is loaded into the FIFO, the timer is restarted. If the timer times-out, the CD1865 checks if there is any data in the FIFO. If there is, a Good Data Service Request is posted to avoid ‘stale data’. If there is no data in the FIFO, the CD1865 checks that NNDT is enabled and ‘armed’. Arming occurs when the last character is transferred out of the FIFO to the host. If NNDT is on and armed, a Receive Exception Service Request is posted to inform the host of this event. Note that the NNDT is not armed if the last character removed from the FIFO is an exception character.

Every Receive Exception is a unique, one-character event. The Receive Data Count register has no meaning, unlike the Receive Good Data case, the Status byte in the receive exception handling routine must be read. The Receive Data Count register and the associated data character is discarded by the CD1865 at the end of the service routine. The Status byte must be read before reading the Data byte. Once the Data register is read, the Status byte is no longer available.

Figure 7. Receive Timer Operation



5.4.2 Transmit Service Requests

Each transmitter contains 8 bytes of Transmit FIFO in addition to the Transmit Holding register and the Transmit Shift register. As data is being transmitted, the FIFO status is being monitored by the CD1865. A service request is invoked for one of the following conditions:

- Transmit FIFO Empty — When the Transmit FIFO is empty, there is still one character in the Transmit Holding register and one character in the Transmit Shift register. The host has two character times to respond to this request without causing a gap in the Transmit Data Stream.
- Transmitter Empty — The Transmit FIFO, Transmit Holding register, and the Transmit Shift registers are now empty. This signifies that all characters written to the FIFO are completely transmitted.

The host can select which one of these causes a Transmit Service Request, and it is used by programming the options in the Service Request Enable register (SRER).

Data must *not* be put into the Transmit FIFO at any time other than when the CD1865 is in a Transmit Service Request context for that channel. During a transmit service, characters (up to eight) are placed into the FIFO by the Transmit Data register (TDR).

5.4.3 Modem Signal Change Service Requests

The CD1865 can be programmed to assert a service request when a channel's modem input signals has changed states. The change-detect options are programmed in the Modem Change Option registers. Individual modem pin service requests are enabled by setting the corresponding bits in the Service Request Enable register.

The host must read the Modem Change register during a modem change service to determine which modem signal changes were detected. This is indicated by a '1' in the appropriate bit location. The Modem Change register must be reset to a '0' by the host before exiting the service request because the CD1865 does not do this. Refer to Section 7.4 for more details.

5.4.3.1 Using Modem Pins as Input/Output

The pins labelled as modem pins are general-purpose I/O pins that can be controlled by either the CD1865 processor or the host system. There is no direct, hardwired connection from any modem pin directly to a transmitter or a receiver. This means that these pins can be used for general-purpose I/O if they are not needed for modem-control purposes. See Section 7.4 for more details.

5.5 Implementing Service Requests

The CD1865 is designed to easily interface with any processor, yet be efficient and flexible enough to provide maximum throughput. The CD1865 generates service requests and waits for acknowledgments of these from the host. However, service requests can be implemented in either hardware or software; likewise, acknowledgments can be affected either way to offer maximum advantages to the system designer and programmer. This interfacing can be grouped as various steps.

Service requests must be 'noticed' by the host system before they can be acted on, and this can be done the following three ways:

1. Provide three levels of interrupt support, with three separate levels and three separate vectors. This is well-suited to Motorola® 680X0 processors.
2. Provide a single level of interrupt support; this is an effective method when using 8-bit processors such as the Z-80 and many Intel® microprocessors.
3. Poll the device directly in software.

Once the host has 'noticed' the service request, it has the following two choices for acknowledging the request and beginning to service it:

- Acknowledge the request by a hardware-based service acknowledgment, as is typically done in interrupt-driven systems.
- Acknowledge the request in software by reading from a register in the CD1865.

Table 4. Service Request Methods

		How the host detects the Service Request		
		1. Three-level Hardware Interrupt	2. Single-level Hardware Interrupt	3. Software Polling
How the host acknowledges the Interrupt	a. Hardware-based service acknowledge	1a Full Interrupt – Type A	2a Not recommended (Inefficient)	3a Not recommended (Inefficient)
	b. Software-based service acknowledge	1b Full Interrupt – Type B	2b Single Interrupt	3b Software Polled

Thus, there are six theoretically possible options for interfacing the CD1865 to the host system. Two of the methods (2a and 3a) are not practical to implement without external hardware, and offer no performance advantage. Each of the other four methods has advantages and drawbacks depending on the type of host CPU being used and whether or not that host CPU supports more than one CD1865. The four methods used are listed in Table 4.

- This method is called 'Full Interrupt – Type A'. The system is fully interrupt driven with acknowledgments in hardware. It requires a host with at least three interrupt priority levels available and the ability to acknowledge on multiple levels. This is the technique used by Motorola 680X0 processors. It is the most efficient method when the host CPU has a relatively fast interrupt context switch time and when the host CPU has duties other than driving the CD1865s.
- This method is called 'Full Interrupt – Type B'. It still has three levels of interrupt, but provides a single acknowledgment level. It is commonly used in Intel-type processor systems where there is an 8259A interrupt controller. The 8259A receives the three levels of interrupt, but it provides its own vector to the host rather than that of the CD1865s. Then the host acknowledges the CD1865s Service Request by reading the Vector register.
- This method is called 'Single Interrupt', and is best-suited to systems having only a single interrupt input, such as most 8-bit microprocessors. After the host receives its interrupt and is entering its interrupt service routine, it reads the CD1865 to evaluate which of the three types of service requests is responsible for the interrupt.Then it acknowledges the interrupt by reading the appropriate Request Acknowledge register. Note that the single interrupt signal must be generated by the logical OR of the three request outputs with external output gates, not by 'wire-OR'ing' them.

- This method is called ‘Software Polled’. Polling is often used in situations where the host system is primarily dedicated to servicing the serial channels and has few other tasks to perform. It is usually better when the host CPU has a long interrupt context switch time. In this method, the host periodically checks the CD1865s to determine if any service requests are pending. If they are, the host acknowledges them in software and proceeds with the service.

One of the advantages of the CD1865 is that it allows the use of any of the above techniques, or a combination. Such a combination is referred to as ‘Mixed-mode operation’. In a typical mixed-mode design, normal interrupts are used to signal to the host that service is required. After the host enters its interrupt service routine, it services the CD1865 that generated the service request. Then the host polls the CD1865s to determine if more channels require service. If the host finds a channel requiring service, it handles it in the usual manner, and then proceeds to poll for more service requests. This process continues until all CD1865s are handled. Because the host is not exiting and re-entering its own interrupt context each time, much host CPU time is saved, resulting in even faster overall performance.

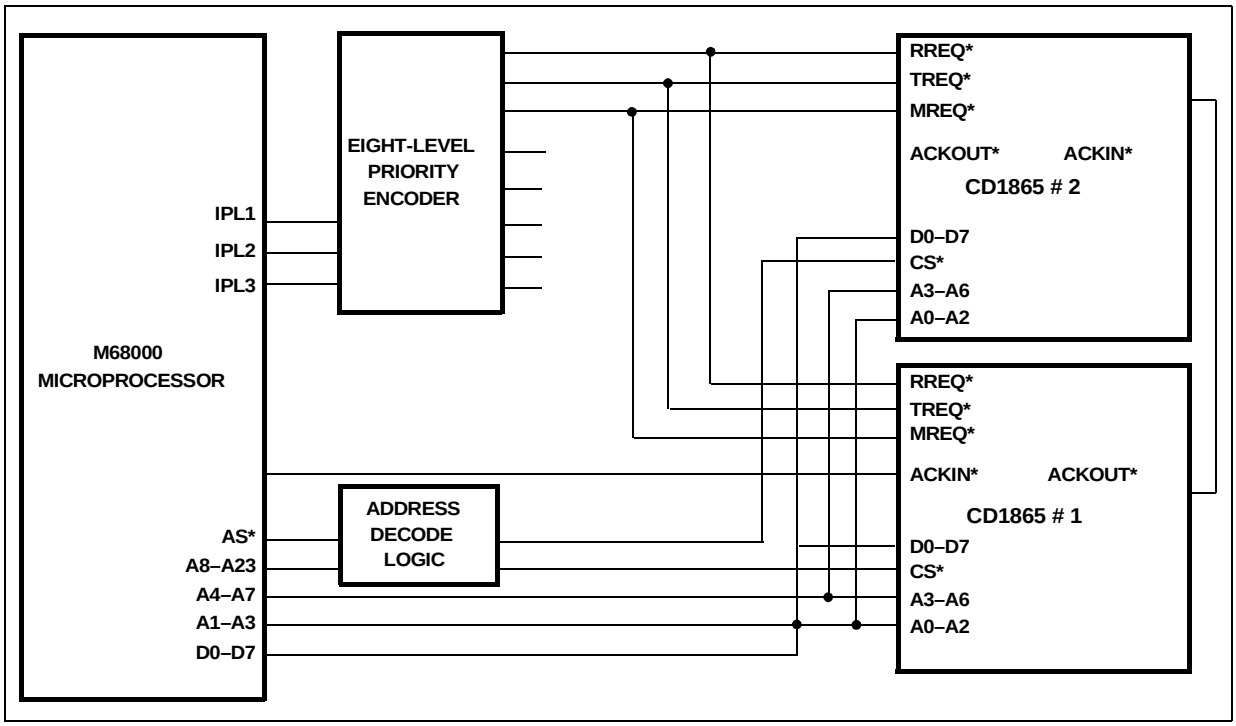
The Advantage of a mixed-mode design is that the software has complete control of whether to be fully interrupt driven or to poll in certain circumstances. A mixed-mode design is recommended to tune a system for optimum performance.

A CD1865 evaluation board can be employed to analyze CD1865 performance and evaluate different software implementations. Intel testing (in an AT-compatible ’386 machine) found that a mixed-mode system provided the highest overall throughput with minimum host CPU loading. This is generally found to be the case with host processors that have relatively long interrupt response times, such as the Intel ’386.

5.5.1 Method 1a — Full Interrupt – Type A, Three-Level Interrupt with Three-Level Acknowledge

This method is illustrated in Figure 8. It is best-suited for 680X0-family processors. The three CD1865 service request lines are connected to the Interrupt Priority Encoder. When the host performs an interrupt acknowledgment cycle, the CD1865 responds with its vector. The host uses this vector to jump directly to the appropriate service routine. Other methods can also be used with a 680X0-based system.

Figure 8. Three-Level Interrupt with Three-Level Acknowledge Example

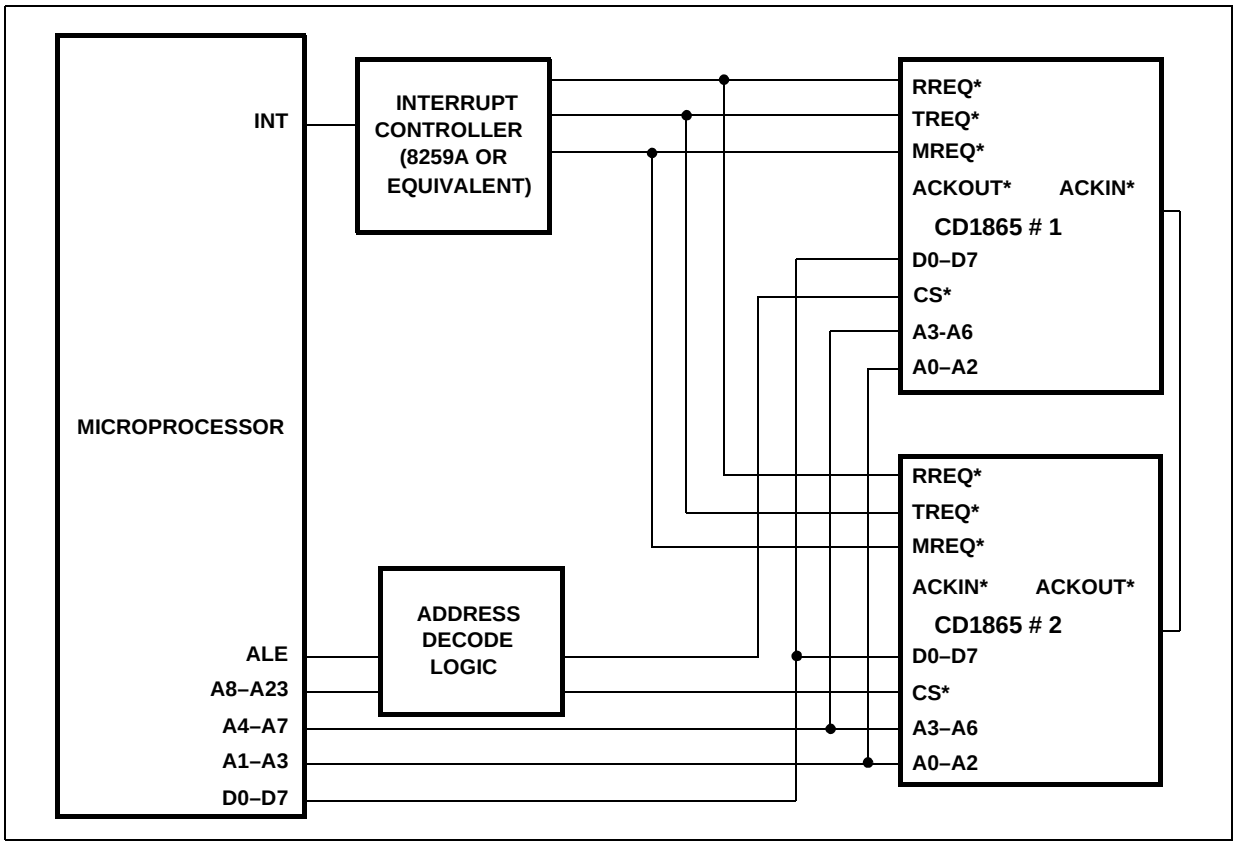


5.5.2 Method 1b — Full Interrupt – Type B, Three-Level Interrupt with Single-Level Acknowledge

This method is illustrated in Figure 9. It is useful with 80X86 systems that use the 8259A Interrupt Controller. Since the 8259A supplies its own vector to the host when an INTA cycle occurs, the host can simply read the CD1865's vector by the method described in the polled interface example or a separate device select decode can be provided to drive the ACKIN* input.

After the 8259A supplies a vector to the 80X86 host CPU, the host performs a software acknowledgment to the CD1865, and transfers the CD1865 vector to the host. This allows the service request to be processed.

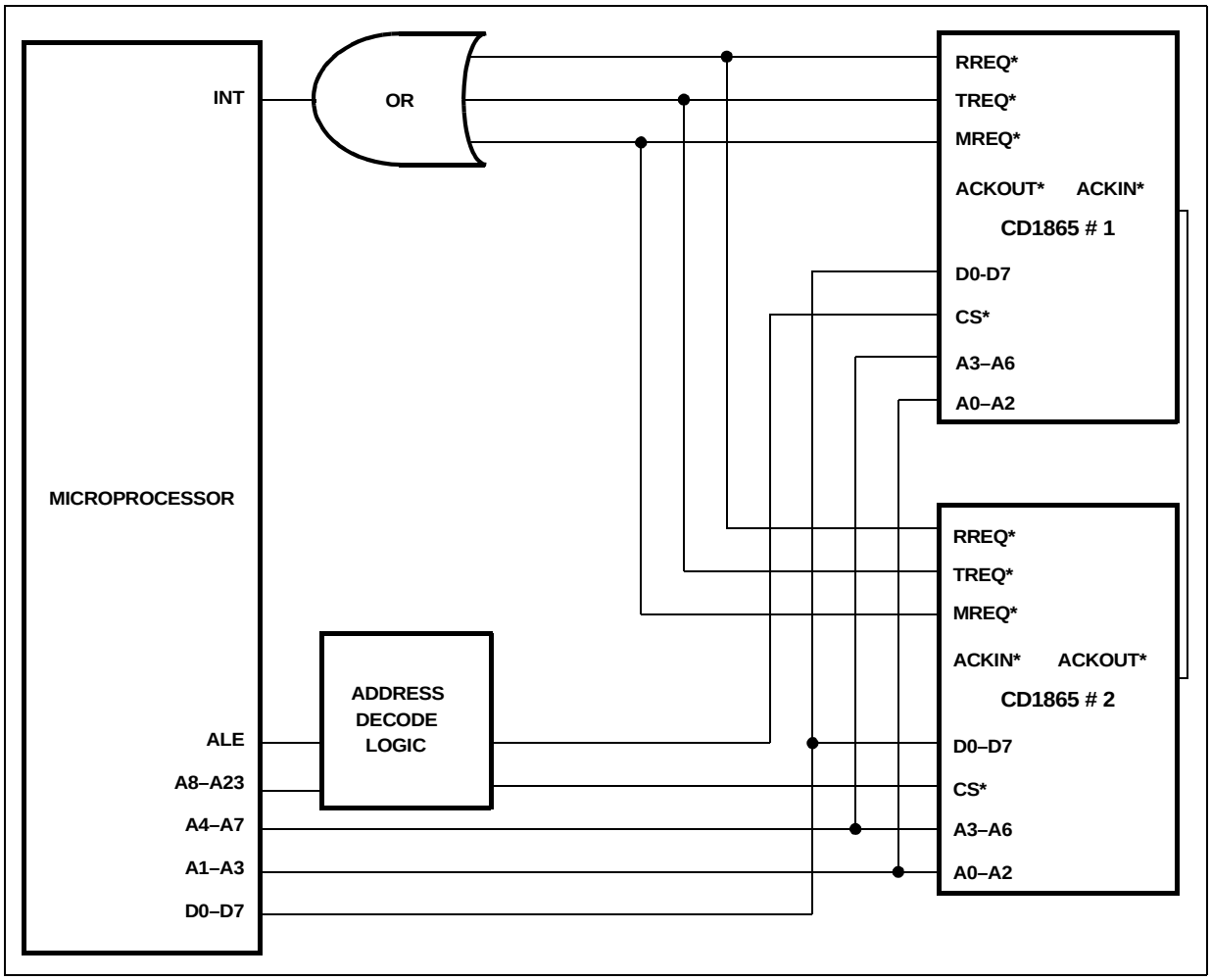
Figure 9. Three-Level Interrupt with Single-Level Acknowledge Example



5.5.3 Method 2b — Interrupt Interface, Single-Level Interrupt with Single-Level Acknowledge

This method is illustrated in Figure 10. It is best-suited to host systems having a single interrupt input. The three service request lines from the CD1865 are run through an ‘OR’ gate to the host’s interrupt input. When an interrupt occurs, the host system polls the CD1865s, determines which of the three levels is interrupted, and acknowledges it accordingly.

Figure 10. Single-Level Interrupt with Single-Level Acknowledge Example



5.5.4 Method 3b — Polled Interface

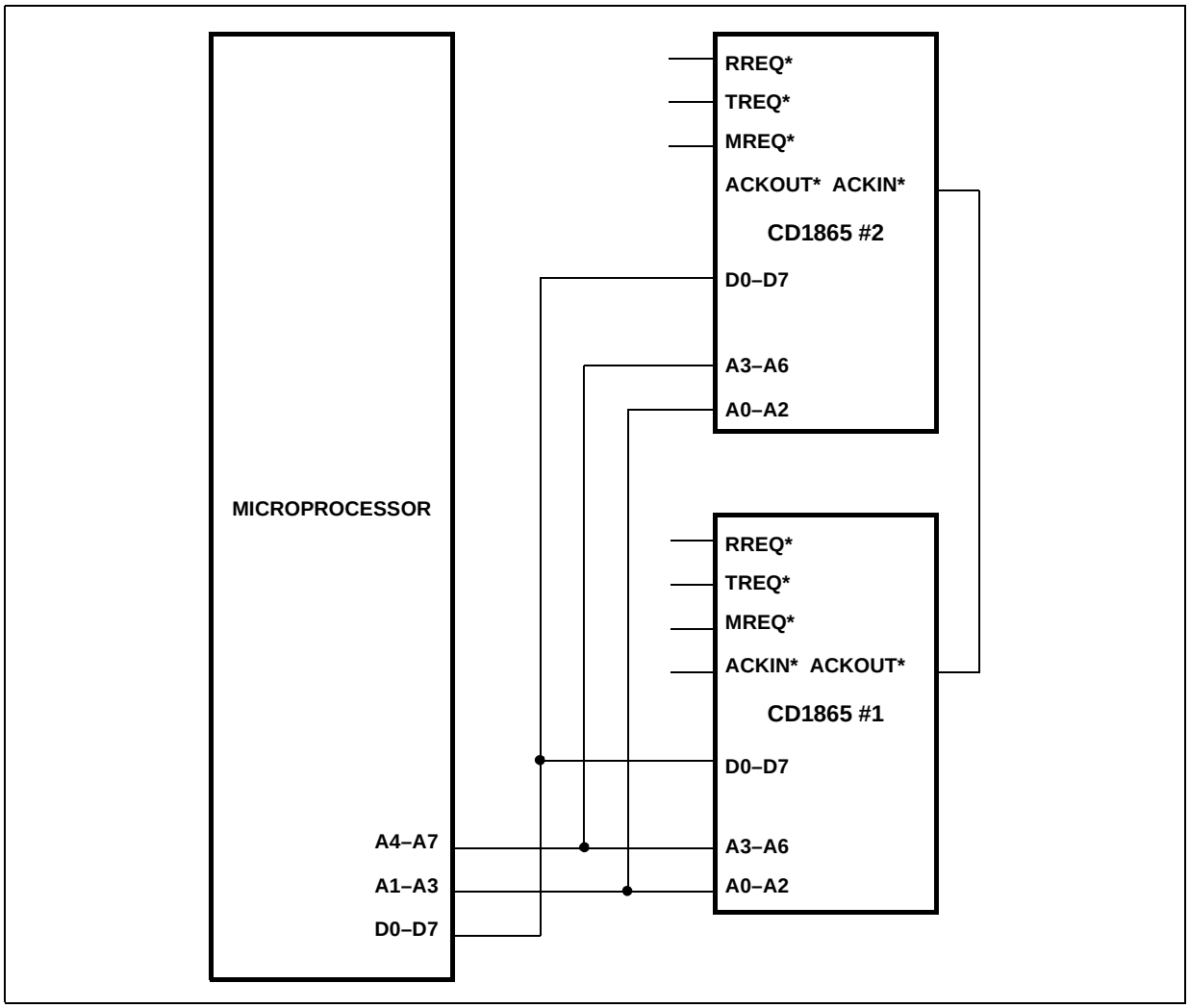
This method is illustrated in Figure 11. Polled operation can be used with any type of host CPU, or it can be used in combination with interrupts to provide a mixed-mode system optimized for a particular application. In a polled system, the host reads the Service Request Status register (SRSR) within the CD1865 to determine whether there are any channels that need service. (Note that unlike traditional UARTs, only one register needs to be read to determine if there are any channels in any device that need attention, and this saves time).

If the host finds channels needing service, it acknowledges the required type by reading one of the three Request Acknowledge registers. These provide a vector that can be used to jump directly to the correct service routine. Processing from this point proceeds as in the case of interrupt-driven operation. Note that the difference between this method and Method 2b lies in how the host system becomes aware of the need to service the CD1865. In Method 2b a single interrupt starts the process. In Method 3b the host polls periodically. The two methods can be combined — an interrupt triggers the first service, but the host continues to poll until any other pending requests are serviced.

There is a difference between the CD1865 and conventional dumb UARTs that makes the CD1865 more efficient even when operating in a polled environment. With a dumb UART, the host polls each channel in turn to determine whether it has any data. With the CD1865, the host polls the CD1865s as a group for whether it has data. If it does, the CD1865s indicates the channel, rather than the host testing each channel in turn. In fact, it is not possible for the host to dictate which channel is serviced; the CD1865 determines this order. This minimizes both the number of polling steps required and the amount of time each needs. This also ensures fair, balanced service of all channels.

There are several ways that a host system can poll the CD1865. Each method has certain advantages. The most direct method is to read the Service Request Status register (SRSR). This register contains three bits that indicate whether there is a request pending for receive, transmit, or modem signal change, on the CD1865 being read. There are three more bits that provide the same information for all CD1865s in the system — these three bits reflect the state of the wire-OR'ed external request lines. Thus a single read operation can determine if there is any activity.

Figure 11. Simple Software Polled Interface Example



5.5.5 Comparison of Interrupt and Polled Code Sequences

Figure 12 and Figure 13 show the code sequences for polled and interrupt service request methods.

Figure 12. Polled Code Sequence

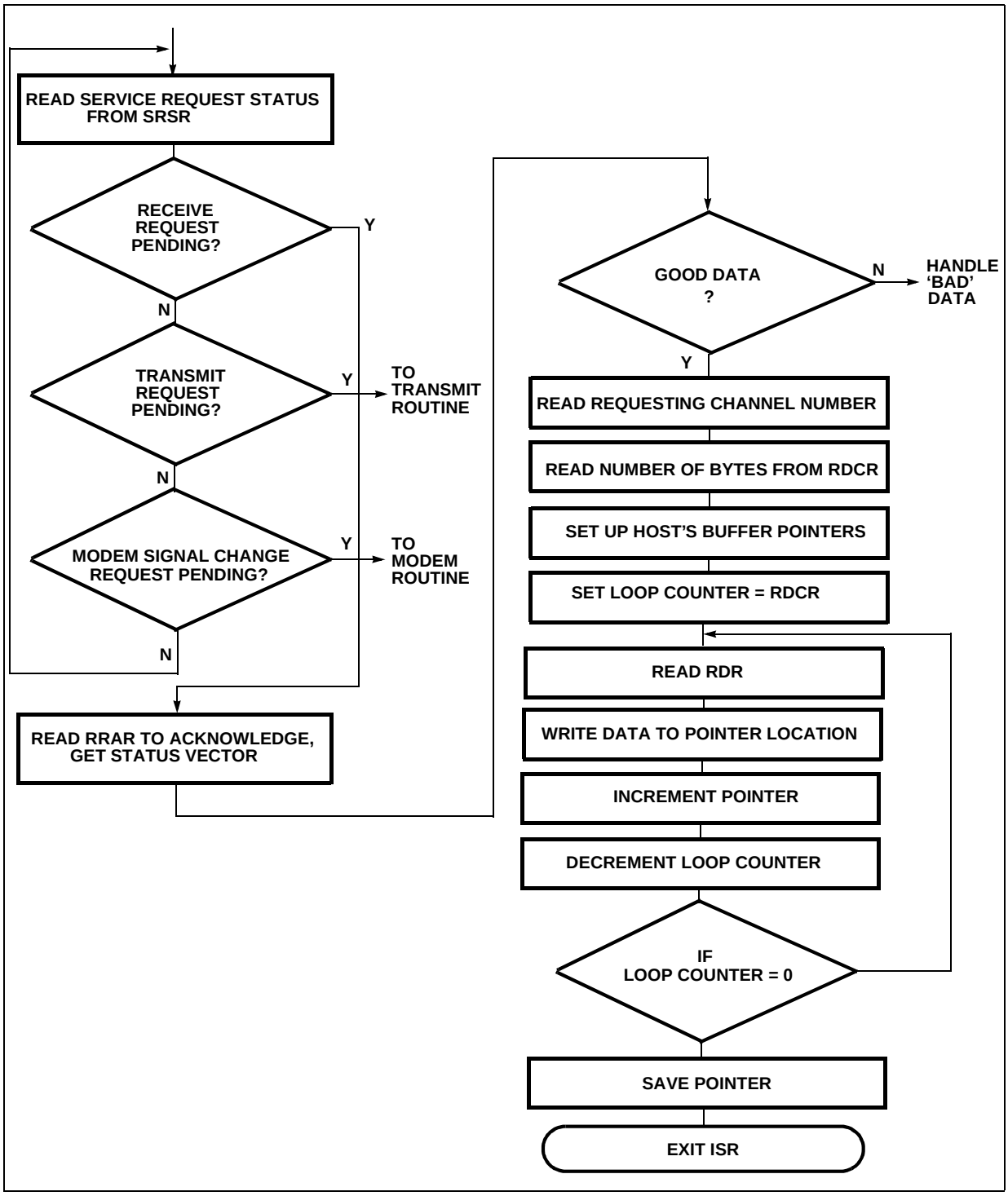
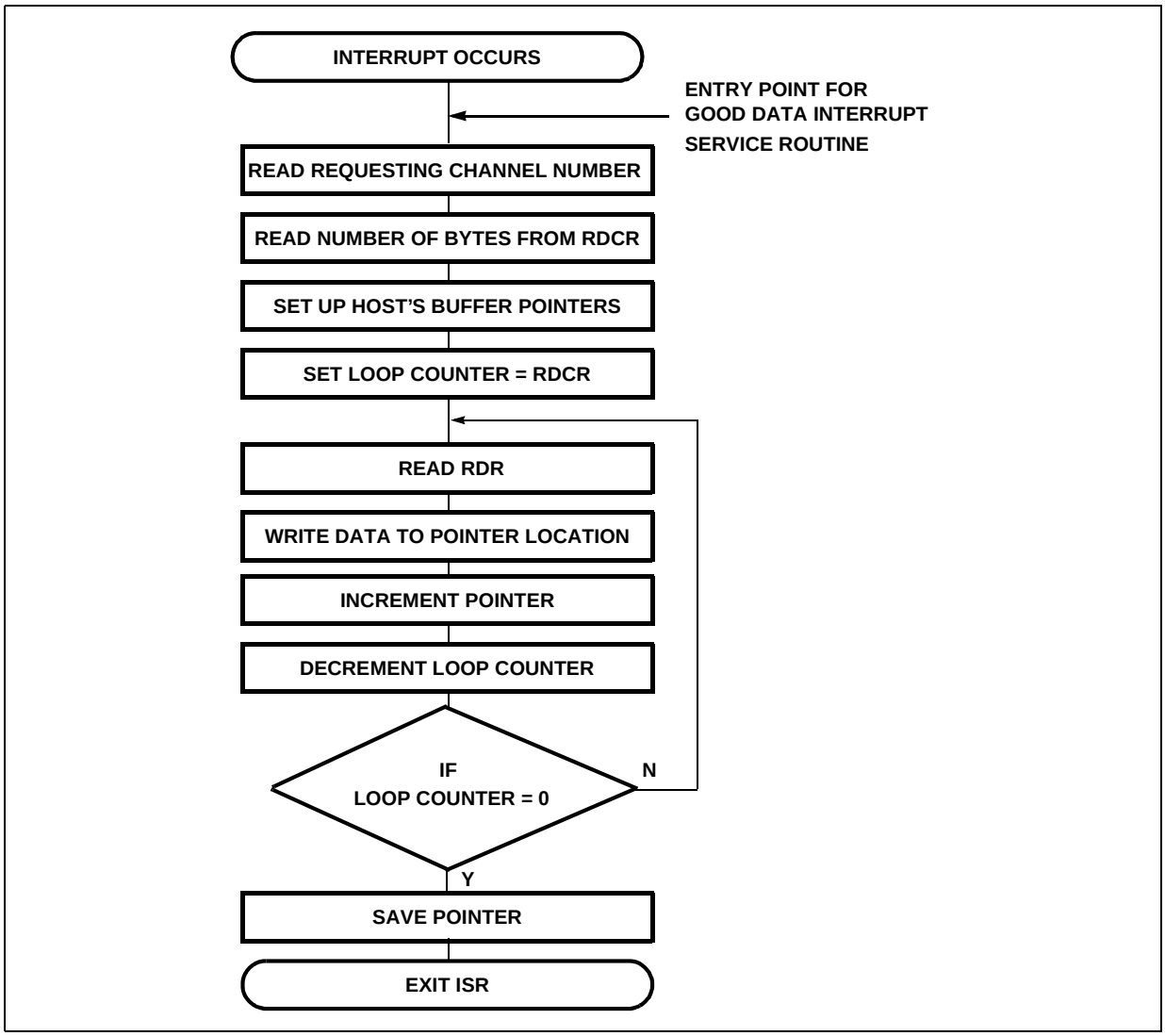


Figure 13. Interrupt Code Sequence



5.5.6 Cascading Service Requests with Multiple CD1865s

Regardless of the method used to support service requests, multiple CD1865s can be cascaded by tying together all * lines, all * lines, and * lines. These lines are open-drain so they may be wire-OR'ed. The CD1865s are then daisy chained by simply connecting the ACKOUT* of one device to the ACKIN* of the next.

The host knows which CD1865 is requesting service by the ID value returned through the Global Interrupt Vector register. Up to 32 CD1865s can be cascaded in any one daisy chain in this manner. Since multiple daisy chains are possible, the maximum number of CD1865s can be large. The 32-per-daisy-chain limit is set by the five bits in the GVR. These bits can be used to identify which CD1865 responded to the service request acknowledge cycle. The user must program different values into the upper-five bits of each CD1865s GVR.

Note that thirty-two CD1865s is the logical limit per daisy chain. Since it takes over 1000 ns for an acknowledgment to ripple down 32 devices, it may not be efficient to have one long chain in heavy-traffic applications.

Note: In some systems that daisy chain many CD1865 devices, a potential timing hazard exists if the host processor does not allow sufficient time for the removal of the ACKIN*/ACKOUT* daisy-chain signal to propagate through all devices. In the event that the host processor begins I/O operations with another section of logic and applies DS* (RD* or WR* in an Intel environment) while an active ACKIN* is being applied to a CD1865 due to propagation delay time, unpredictable results can occur. This constitutes an illegal acknowledge cycle. The failure mode is most often a cessation of service requests from the device, especially of the type that is being serviced when the illegal access occurs. Take care to ensure that the 35-ns propagation delay per device is included in any wait-state generation.

5.5.7 Multiple CD1865s without Cascading

It is possible to interface several CD1865s without using the cascade feature. There is an advantage to this because as there is less delay incurred while waiting for the service acknowledgment to ripple down a chain of devices. There are two possible disadvantages. If each of the CD1865's three service request lines has a separate input to the interrupt controller, the interrupt controller is more complex, and the fair-share feature does not work. If the service request lines are wire-OR'ed, fair share works, but the host has to test each CD1865 in turn to see which one generated the service request. To implement this method, simply connect the CD1865 address and data lines in the usual manner.

5.5.8 Acknowledging Service Requests

As mentioned in Section 5.5 on page 35, two different methods are used to acknowledge a service request. One method is hardware-based, and the other is software-based. The hardware-based mechanism is a specific type of bus cycle that uses the ACKIN* and ACKOUT* signals and the in the CD1865. An acknowledge cycle is defined where ACKIN* and DS* are active and CS* is inactive. This method is used by processors that perform interrupt acknowledge cycles, such as the 680X0.

The software-based mechanism uses three registers — Receive Request Acknowledge register, Transmit Request Acknowledge register, and Modem Request Acknowledge register. Reading any of these registers has the effect of acknowledging a service request, and the data read is the appropriate vector, that is, the contents of the Global Interrupt Request Vector register. The low-three bits of this register are modified to indicate the specific type of interrupt being acknowledged.

If the host reads these registers when no service request is pending, either of two things can happen. If daisy chaining of acknowledgments is enabled, the ACKOUT* pin of the CD1865 asserts. If daisy chaining is not enabled, the part supplies a vector with the low-three bits set to a '0'. Thus, it is possible to 'fish' for service requests, that is, to acknowledge each CD1865 in turn until a non-zero vector is received.

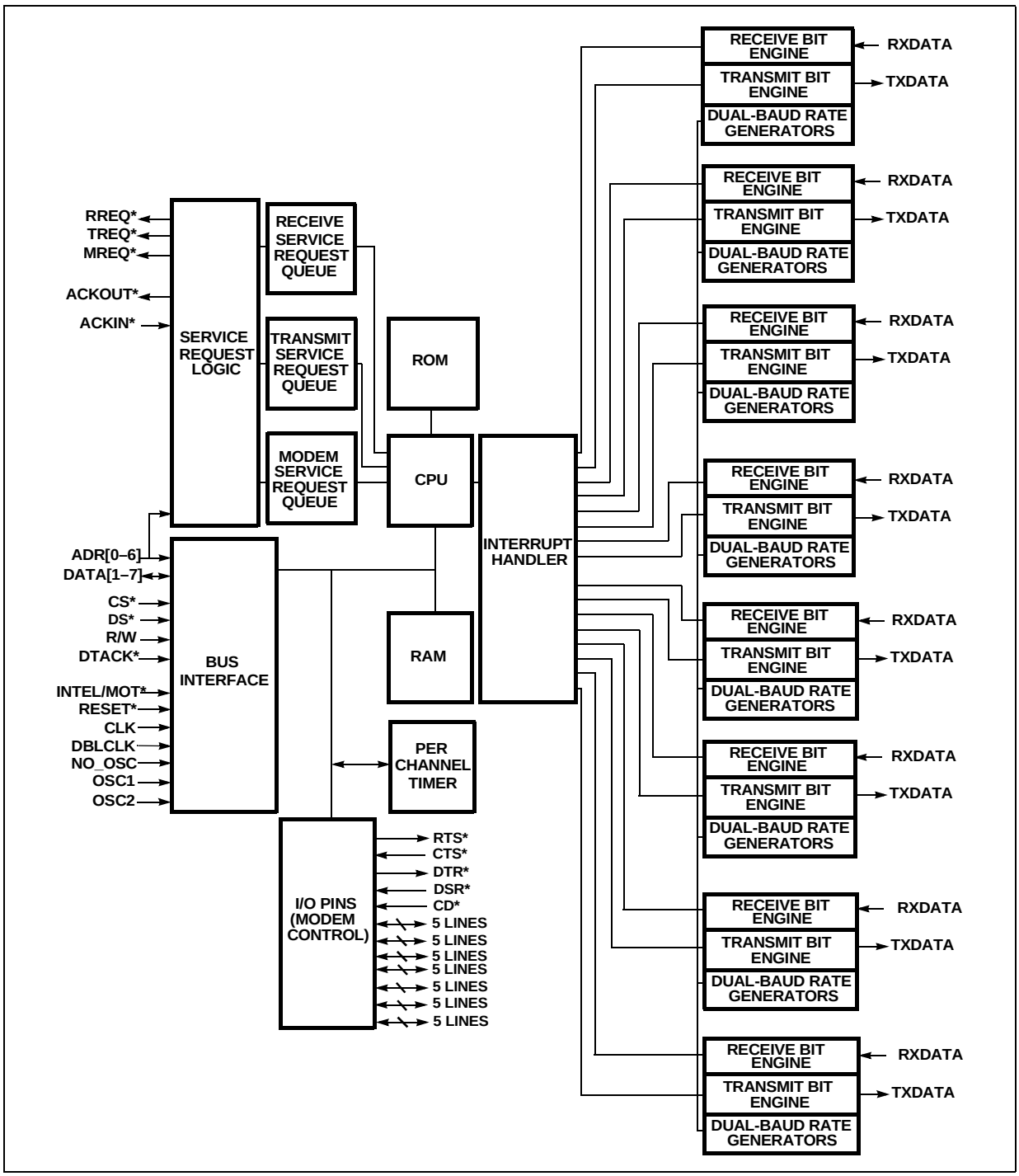
'Fishing' is not usually an efficient software technique, but can be useful in some circumstances. For example, in systems that are normally interrupt-driven, but where interrupts are not available for diagnostics or other reasons, the host can determine if a service request is pending by reading the appropriate Request Acknowledge register. The CD1865 must be configured not to daisy chain; in this case it returns a vector if a request is pending, or '00' if no request is pending. The host can try all three levels of request in turn. This method works for either single CD1865s or multiple

devices. In multiple-device systems, either disable daisy chaining on all devices and ‘fish’ each individually, or disable daisy chaining on the last device only and ‘fish’ the device at the beginning of the chain.

Both methods of acknowledging service requests can be used interchangeably. It is usually advantageous to use Mixed mode. For example, after receiving an interrupt and servicing it in the normal manner, the host should read the Service Request Status register (SRSR) to see if other requests are pending. If so, the host can acknowledge by reading the appropriate Request Acknowledge register (RRAR, TRAR, and MRAR) and proceed to service the request. This avoids the time required for the host to exit its interrupt routine, only to re-enter it immediately for the next request.

6.0 System Bus Interface and System Clock

Figure 14. Internal Block Diagram



6.1 System Interface Considerations

When using the CD1865, two areas where system architects, designers, and programmers should consider options are system clock speed, and unclocked versus clocked-host bus interface.

6.2 System Clock and Bit Rate Options

6.2.1 System Clock

System clock is a high-frequency clock (supplied by the user) used by the CD1865 to receive all the necessary timing. The CD1865 is capable of handling system clock levels of TTL-compatible voltage swings; however, the V_{IL} and V_{IH} specifications are not identical to all families of TTL logic. Specifically, the clock signal (and the reset signal) have lower V_{IL} and higher V_{IH} than the worst-case specifications of some TTL families. In general, any TTL family is adequate if not heavily loaded. Refer to the DC Specifications in Section 10.3 for details.

××××××The CD1865 can be operated from the main system clock or its own clock. Operation from the main system clock can reduce the number of clocks required, and it allows the bus interface between the system and the CD1865 to be clocked, but in general, typical system clock speeds are not exact baud-rate multiples. As bit rates are received from the clock, it is important to consider this when selecting a clock value. If exact baud rates are needed, or the system clock is not a convenient value, the CD1865 must be supplied with its own clock or crystal.

6.2.2 External Clock

It is recommended that the 2×-clock option (oscillator or crystal) be used wherever possible. Figure 15 shows a possible design configuration for the clock circuitry if the crystal is being used. Please refer to the CD1865 Evaluation Kit documentation for details on the design configurations used. The crystal used for the evaluation board is a 66-MHz third overtone part.

Figure 15. 2× Clock Option

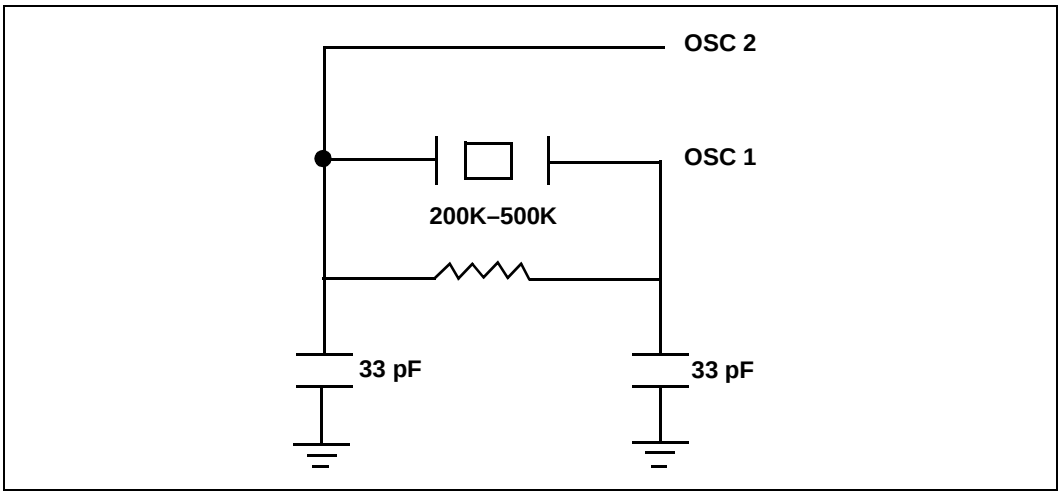
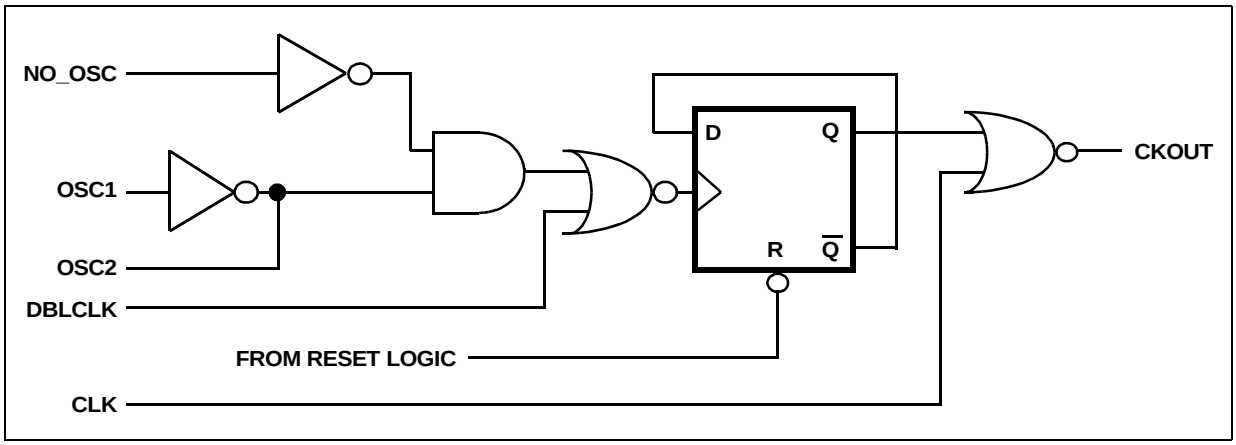


Figure 16.



6.2.3 1× Clock Option

It is recommended that a 2×-clock option be used where ever possible. If using a 1×-clock options, refer to Table 10 on page 129 for clock duty cycle requirements.

6.2.4 Bit Rate Options

The CD1865 supports independent transmitter and receiver bit rates on each of its eight channels. The bit rate is determined by a 16-bit period value (divisor) stored in the Transmitter Bit Rate Period registers (TBPRH and TBPRL), or in the Receiver Bit Rate Period registers (RBPRH and RBPRL). These registers establish the period of the corresponding Transmitter and Receiver Bit Rate counters. To set a given bit rate, the value to be loaded is determined by the following equation:

$$\text{Bit Rate Divisor} = \frac{(\text{CLK frequency \{in Hertz\}})}{\text{(16 x desired Bit Rate \{in bits per second\})}}$$

This equation may yield a non-integer result. The nearest integer value is the optimum choice for that bit rate and system clock combination. The value loaded in the Bit Rate Period registers must be that integer expressed as a 16-bit binary value. If rounding is necessary, the percentage bit rate error can be calculated by:

$$(\text{Bit Rate Divisor} - \text{Integer}) \times 100 / \text{Bit Rate Divisor}$$

The popular bit rates and their corresponding divisors at various system clock rates are shown in Table 5.

Table 5. Bit Rate Constants, CLK = 33 MHz

Bit Rate	Divisor†	Error
110	493e	0.000%
150	35b6	0.000%
300	1adb	0.000%
600	d6e	0.015%
1200	6b7	0.015%
2400	35b	0.044%
4800	1ae	0.073%
9600	d7	0.073%
19200	6b	0.393%
38400	36	0.538%
56000	25	0.461%
57600	24	0.538%
64000	20	0.703%
76000	1b	0.509%
115200	12	0.538%

†All divisor values are in hex.

Table 6. Bit Rate Constants, CLK = 25 MHz

Bit Rate	Divisor†	Error
110	377d	0.003%
150	28b1	0.003%
300	1458	0.006%
600	a2c	0.006%
1200	516	0.006%
2400	28b	0.006%
4800	146	0.147%
9600	a3	0.147%
19200	51	0.467%
38400	29	0.762%
56000	1c	0.352%
57600	1b	0.467%
64000	18	1.696%
76000	15	2.144%
115200	e	3.219%

†All divisor values are in hex.

Table 7. Bit Rate Constants, CLK = 20 MHz

Bit Rate	Divisor†	Error
110	2c64	0.003%
150	208d	0.004%
300	1047	0.008%
600	823	0.016%
1200	412	0.032%
2400	209	0.032%
4800	104	0.160%
9600	82	0.160%
19200	41	0.160%
38400	21	1.376%
56000	16	1.440%
57600	16	1.376%
64000	14	2.400%
76000	10	2.720%
115200	b	1.376%

†All divisor values are in hex.

Table 8. Bit Rate Constants, CLK = 15 MHz

Bit Rate	Divisor†	Error
110	214b	0.003%
150	186a	0.000%
300	c35	0.000%
600	61a	0.032%
1200	30d	0.032%
2400	187	0.096%
4800	c3	0.160%
9600	62	0.352%
19200	31	0.352%
38400	18	1.696%
56000	11	1.547%
57600	10	1.696%
64000	f	2.400%
76000	c	2.720%
115200	8	1.696%
†All divisor values are in hex.		

6.2.5 Maximum Throughput Limits

The CD1865 is internally a fully static, synchronous design. Consequently, the maximum data rate handled by CD1865 is determined by the clock speed at which it is operating. There are a fixed number of CD1865 processor cycles required to process each bit and character; a slower CD1865 processor rate equates to a slower bit rate. The minimum clock frequency required can be determined by the data rate needed for support.

In general, the CD1865 can maintain 100% full-duplex throughput when divisors of 16 or greater are used. For a given master clock frequency, this limitation can be used to determine the maximum bit rate at which the system can sustain 100% throughput on both receive and transmit. Divisors as small as 12 can be used, however a degradation in throughput is observed. This degradation is seen as gaps between transmit characters and are, in effect, extra long stop bits. This is a fail-safe condition. Divisors smaller than 12 can work in an application if less than eight channels are enabled.

6.3 CD1865 Basic Bus Interface and Addressing

The CD1865 is addressed through an active-low Chip Select (CS*) in conjunction with seven Address Inputs A[0:6] that are mapped CD1865 internal addresses in two addressing modes — global and channel. In Channel Addressing mode, the bits defining the channel to be accessed are provided from the Channel Access register (CAR) within the CD1865.

The most-significant Address Input (A6) performs the selection between global- and channel-specific addresses. If this bit is a ‘1’, the address is global, and is not associated with any specific channel. If this bit is a ‘0’, the address is channel-related.

With the exception of the FIFOs, all channel-specific registers are accessed by first setting the required channel number in the low-three bits of the Channel Access register. FIFOs can only be accessed within the context of a service routine. Attempting to force access to a particular FIFO by setting the CAR causes unpredictable and incorrect results. Within the context of a service request, the effective channel access value is automatically controlled by the CD1865, thus the CAR should not be modified by the host system during service-request processing.

The advantage of this method is that the host never performs any address computation to access the CD1865 during service requests. Because only the registers specific for the active channel (that is, the one being serviced) are accessible to the host within a service request routine. An automatic indexing feature handles this, thus avoiding any burden on the host. Refer to Section 9.3 on Indexed Indirect registers for details.

6.3.1 Intel® Versus Motorola® Interface Signals and Addressing

The CD1865 supports two bus handshake methods. One is patterned after the Motorola 680X0-family processors, and the other after Intel 80X86-bus interfaces. bus interface selection is achieved by the INTEL/MOT* signal. When this signal is ‘high’, the Intel bus interface is selected, and when this signal is ‘low’, the Motorola bus interface is selected. This selection affects the logical meaning of two pins, but has no effect on bus timing.

The two signals having dual meaning are RD* versus DS*, and WR* versus R/W*. When the Intel bus interface is selected, these two pins function as RD* and WR*. These pins can be connected to either the IOR* and IOW*, or to MEMRD* and MEMWR* depending whether the CD1865 is

mapped into memory or I/O space. These pins then serve to select the CD1865, and when either is active (along with CS* or ACKIN*) the CD1865 considers itself selected. CS* and ACKIN* must never be active at the same time.

When the Motorola bus interface is selected, these two signals function as DS* and R/W*. DS* must be asserted (along with CS* or ACKIN*) for all types of cycles, and R/W* should be low when writing to the device.

In either case, the choice of bus interface is entirely up to the user. This feature is for user convenience, and to accommodate the address and bus-control logic that are used. The CD1865 has an 8-bit data bus, and it is a common practice (when connecting 8-bit peripherals to 16- or 32-bit systems) to connect them to only one lane, or 1-byte position. Thus, the CD1865 registers only appear in the host's address space at every other byte address. The most common practice is to connect the CD1865 to the portion of the data bus labelled D0–D7. For the little-endian processors, such as Intel's, the CD1865 appears at even addresses (A0 = 0). For big-endian processors, such as Motorola's, the CD1865 appears at odd addresses.

6.3.2 Unclocked Versus Clocked Bus Interface

Depending on the type and speed of the host processor, another important choice is determining whether the system bus interface will be clocked or unclocked with the host CPU clock. Because there is a single clock for both the bus interface and bit-rate generation, the decision to use either Clocked or Unclocked bus interface is affected by whether exact bit rates are required. Most applications do not require exact bit rates, and operate with rates varying by one percent or so. If exact bit rates are required, the clock speed must be a baud-rate multiple.

One method of bus interfacing may be preferable to another in certain applications. Although the easiest way to interface to the CD1865 is by using the unclocked handshake supplied by DTACK*, in some cases it may be better to design a clocked interface. The latter is true if the host system is running at the same clock speed (or a multiple) of the CD1865 speed.

Unclocked Bus Interface

An Unclocked bus interface is the easiest interface to implement. Simply connect the address, data, and control lines in the customary manner, and use DTACK* to control the number of wait states either by connecting it to the processor's DTACK* (if it has one), or by feeding into a wait-state generator. Figure 17 on page 53 shows a typical Unclocked bus interface.

The maximum bus cycle time is two clock periods plus 10 ns, though typically less because this specification is based on worst-case internal synchronization delays. Using DTACK* saves time; however, it is permissible to hard-wire the wait-state generator for the maximum time.

Clocked Bus Interface

The CD1865 bus interface is controlled by a state machine that samples on the falling edge of the clock. External strobes (CS*, DS*, and R/W*; or CS*, and RD* or WR*) that meet the setup time requirement cause a bus cycle to begin. The external interface can be designed to meet these setup time requirements, and to have shorter CD1865 access cycles. Figure 18 on page 54 shows a typical Clocked bus interface.

A bus cycle consists of two half-clock periods. During the clock-low period, the transaction is set up internally, and the local bus arbitration occurs. During the clock-high period, the read or write transaction to RAM occurs. On write cycles, the data from the host is latched internally on the low-to-high clock transition. On read cycles, the data is available shortly after the end of the clock-high period.

Read and write cycles differ slightly in timing; during a write, it is permissible to remove the WR* or DS* relatively early during the high-clock period, however, this cannot be done during read cycles. The RD* or DS* Strobe is used as an output enable, and must remain low for the data to appear on the external data bus.

Service request acknowledgment cycles follow a different timing than ordinary read cycles. First, it is necessary to have the address stable before asserting ACKIN*. Second, the setup time from ACKIN* and DS* (or RD*) going low to the falling clock edge is longer due to additional internal logic involved in service request acknowledge cycles.

Figure 17. Typical Unclocked Bus Interface

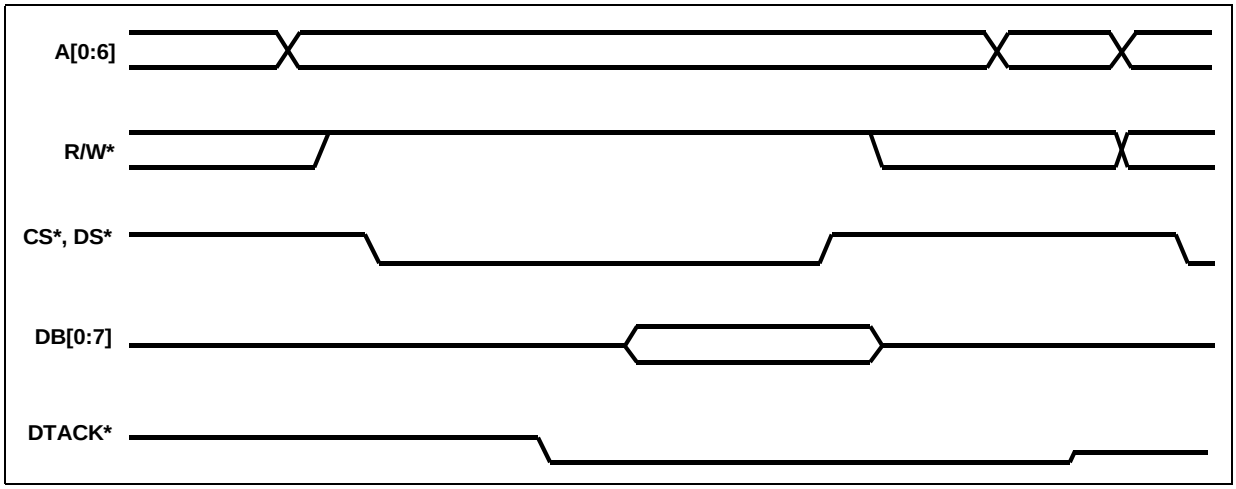
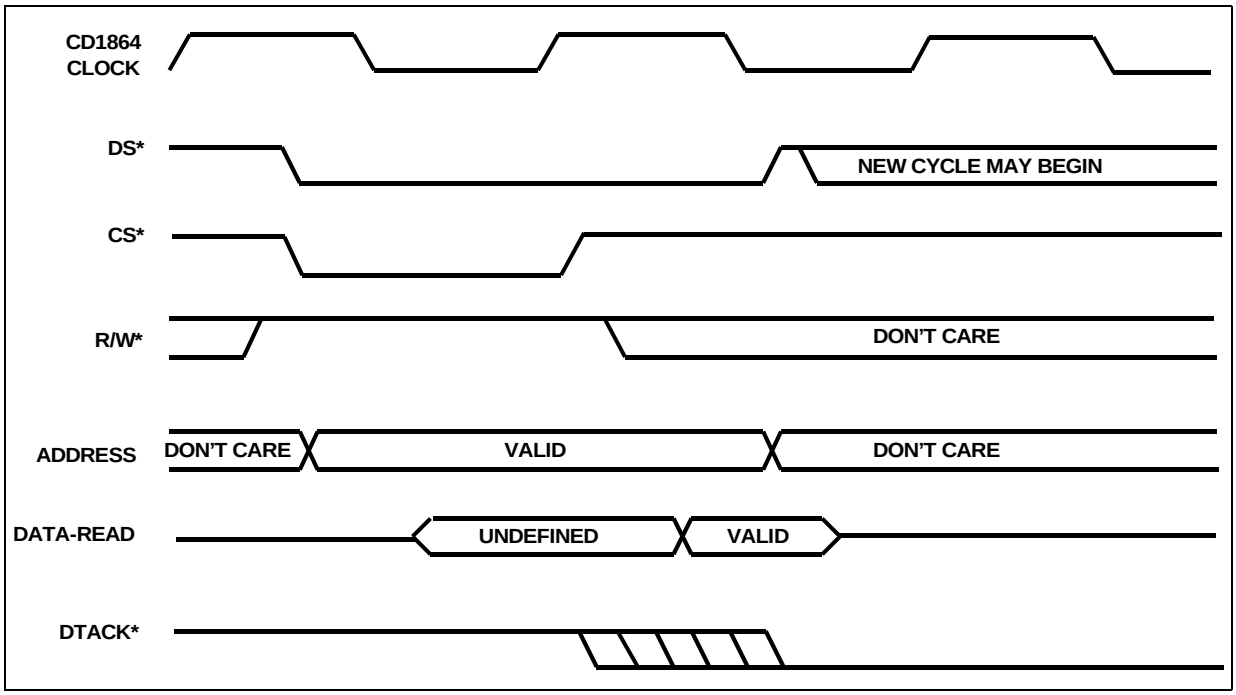


Figure 18. Typical Clocked Bus Interface



6.4 Interface Examples

There are some general design considerations when interfacing the CD1865 to any host environment.

The three Service Request pins (*, *, and *) can change at any time, and this can introduce metastability problems if the interrupt controller requires clocked signals. When designing, take care that all signals are stable when needed.

The Service Request pin of the type being acknowledged is negated at the end of the service acknowledgment bus cycle. Often, during the course of servicing one channel, another channel reaches a state where a request would assert, for example, while servicing receive on channel one, channel two's FIFO fills. The Service Request bits in the Service Request Status register (SRSR) does not reassert until approximately two clock periods after the host completes its write to the End Of register (). In polled or mixed-mode systems, to determine whether another service request of the same level is pending, and to make sure that the host does not re-read the SRSR too quickly, insert a No-Operation (or similar) instruction.

Performing an 'invalid' service acknowledgment bus cycle on the CD1865 is permissible, but it can cause problems in certain circumstances. An Invalid Service Acknowledgment is an acknowledgment for which there is no request pending.

If a service request acknowledgment bus cycle is performed by the host when no service request is pending, either of two things can occur. If the value on the address bus matches one of the three values in the three Service Match registers (), and daisy chaining is enabled, the CD1865 assumes that another device down the daisy chain should receive the request, and asserts its ACKOUT* pin. This propagates down the CD1865 chain until eventually the last CD1865 asserts its ACKOUT*.

At this point, the system waits endlessly unless the bus cycle terminates. The best method is to connect the ACKOUT* of the last CD1865 in the chain to a bus-error input on the host. If there are multiple CD1865s that are not cascaded, the ACKOUT* signals should be OR'ed together through a gate or a PAL.

If an acknowledgment occurs and the value on the address bus does not match any of the Match registers, the first CD1865 in the chain does not pass it along or assert DTACK* and the system waits endlessly unless there is a bus time-out or other mechanism to detect this condition. In either of these circumstances, the 'value' on the data bus is likely to be FFh because the bus is floating (this is system dependent). To make a robust design, do not use FFh as a valid Global Service Vector register (GSVR) value. If daisy chaining is not enabled, then the CD1865 returns a vector of '00' for invalid acknowledgments.

6.4.1 Interfacing to 80X86-Family Processors

The Intel 80X86 family processors often use the 8259A as the interrupt controller, which supplies its own vector during the INTA cycle. The easiest way to interface the CD1865 to an Intel processor is by Mixed mode, as described in Section 5.5.

There is one 'bug' in the 8259A to be aware of. The 8259A can change the prioritizing of its eight inputs, which can result in one of its acknowledge outputs going low briefly (~30 ns) if an input changes at a certain time. This typically occurs if a higher-priority input to the 8259A asserts when the 8259A is about to issue an acknowledge to a lower-priority device. If this occurs at the beginning of a cycle, this brief pulse can cause the CD1865 (and other devices) to malfunction. Be sure that this does not occur. See *Intel 8259A Data Sheet* for details.

6.4.2 Interfacing to 680X0-Family Processors

The 68000-family interface is quite straightforward. The three service request lines go through a priority encoder to the 680X0 IPL inputs. The CD1865s ACKIN* pin is driven by a decoder.

When the 680X0 performs an Interrupt Acknowledge cycle, it drives its address lines A1, A2, and A3 with a three-bit value indicating the level being serviced. The other address lines are set to a 1. If the level being serviced corresponds to a level assigned to the CD1865, external decoding logic should assert the CD1865 ACKIN* pin. The value on address lines A0 to A7 is programmed into the , so the CD1865 recognizes the acknowledgment and proceeds as described in the Service Request Section 5.3.1.

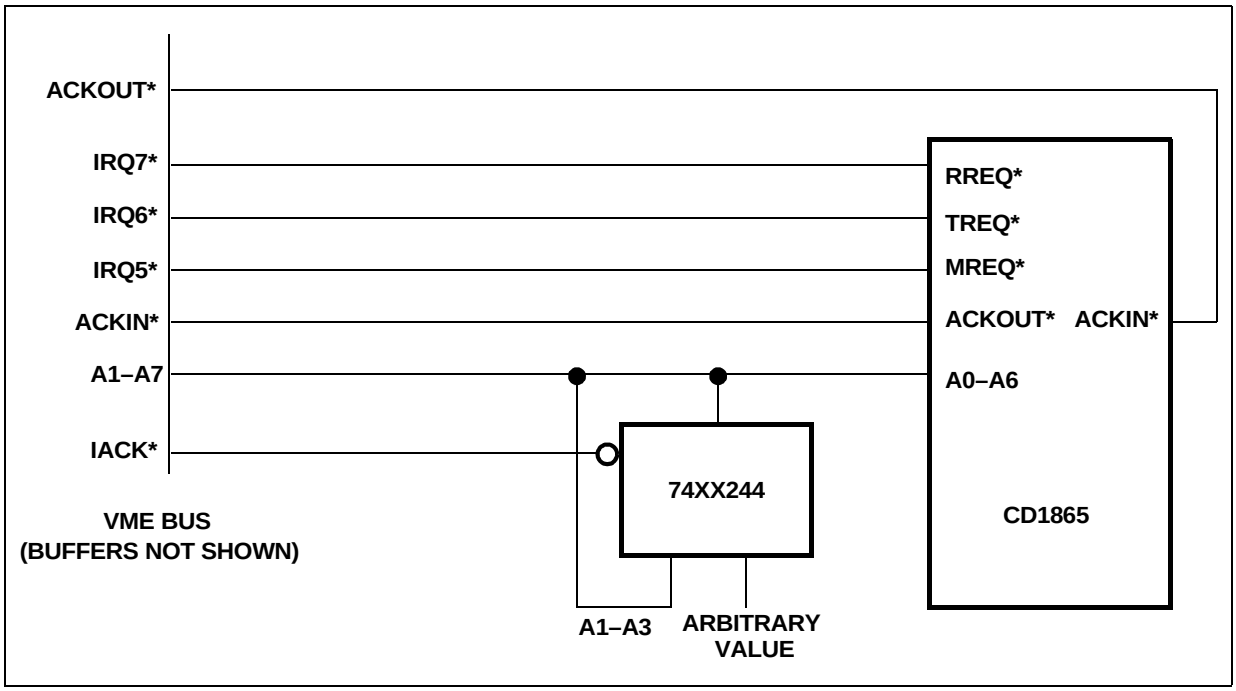
All CD1865 service requests can also be routed to a single interrupt level by using a Mixed-mode interface, as described in Section 5.5.

6.4.3 Interfacing to the VME Bus

The CD1865 can be directly interfaced to the VME bus, and only requires a small amount of logic to complete the interface. This is necessary because service request acknowledgment works differently on the VME bus than on the CD1865. VME defines seven levels of interrupts; each level can be shared among multiple VME cards. During an Interrupt Acknowledge cycle, the VME bus provides three bits on the address bus, indicating the level being acknowledged (A1-A3). Each VME card must pass along an interrupt on all levels it is not using but the CD1865 does not automatically pass an interrupt acknowledgment.

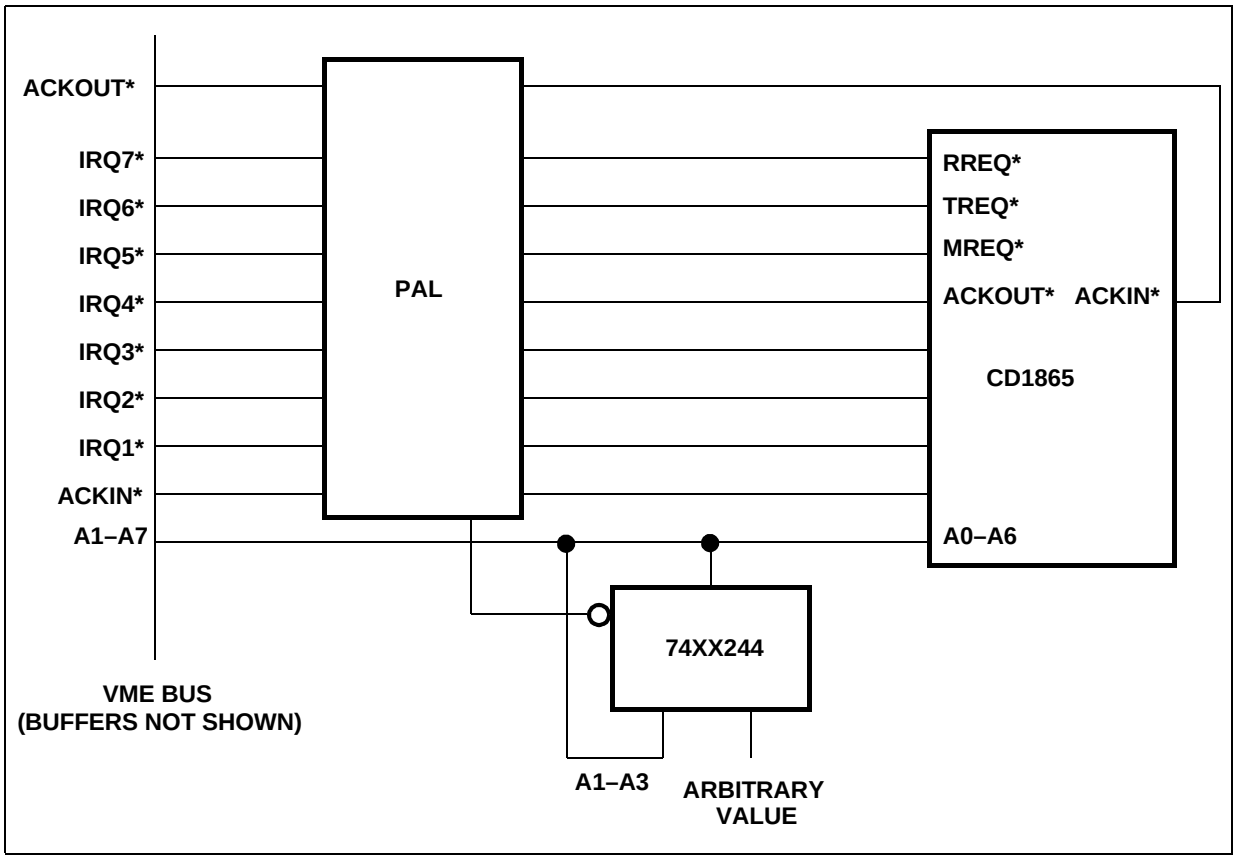
To recognize how this difference can cause a problem, suppose that the three Service Request lines from the CD1865 are connected to levels 7, 6, and 5 of the VME bus (see Figure 19 on page 56). Also, attach a 74XX244 so that during an Interrupt Acknowledgment cycle provides an 8-bit code consisting of the three address bits plus five more hard-wired bits to the CD1865. Now, whenever an acknowledgment of a level 5, 6, or 7 interrupt occurs, the CD1865 either responds or passes the acknowledgment properly. If an acknowledgment occurs on levels 1–4, the daisy chain 'breaks' because the CD1865 does not recognize a match.

Figure 19. Incorrect VME Interface



This condition can be easily rectified, as shown in Figure 20 on page 57. A PAL is used to assert ACKOUT* whenever ACKIN* occurs on a level not being used by the CD1865. The PAL is programmed for fixed levels. For example, if the current VME bus Interrupt level is 1–4, the PAL asserts ACKOUT* whenever ACKIN* is active. If the current level is 5–7, the PAL asserts ACKOUT* when ACKOUT* from the CD1865 is active. If required, the assignment of VME Interrupt levels to the CD1865 can be field-programmable by supplying additional inputs to the PAL, indicating the levels being used by the CD1865.

Figure 20. Correct VME Interface



7.0 Serial Interfaces

7.1 Receiver Operation

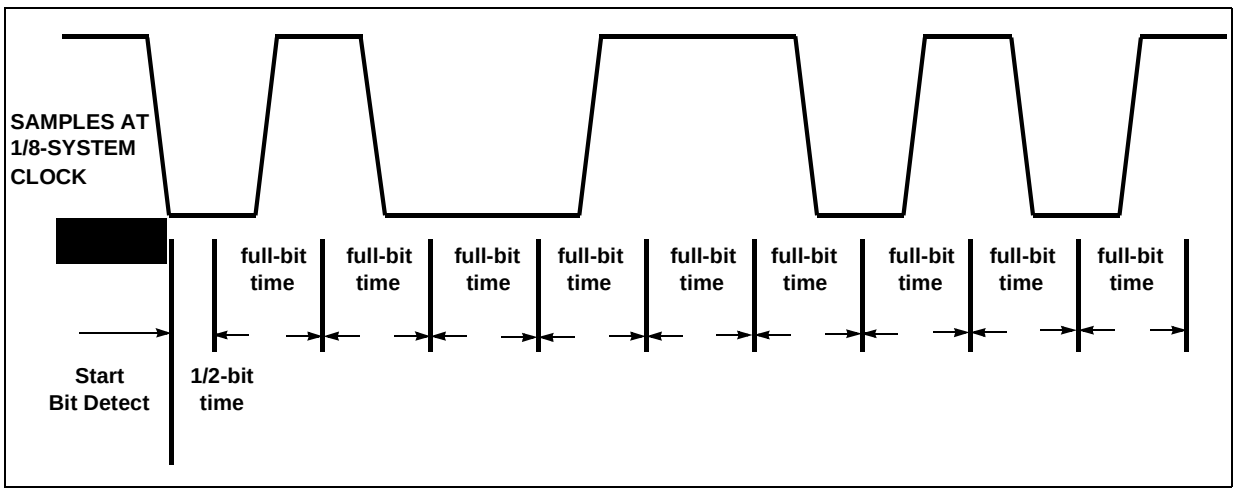
7.1.1 Basic Operation

All receivers are disabled upon master reset. To prepare a receiver, first initialize and then enable it. Once initialized and enabled, the receiver monitors the RxD Line and waits for a high-to-low transition, which indicates a Start bit. This sampling is performed at one-eighth of the System-clock rate regardless of the Programmed bit rate, and it provides accuracy of synchronization with the incoming data. See Figure 21 below for CD1865 bit synchronization. Once a transition is detected, the receiver checks the RxD Input state again (a half-bit time later) to validate that it is a Start bit. A valid Start bit is defined a ‘space’ or a logic ‘0’. If the RxD Input is no longer a ‘space’, then a false Start bit is assumed and the receiver resumes the search for a high-to-low transition. If a valid Start bit is detected, the RxD Input is sampled at one-bit time intervals in the middle of the bit to ensure stable data. Characters are assembled according to the programmed content of the Channel Option register (COR1). Valid character framing (presence of a Stop bit), and Optional Parity bits are checked. After a character is assembled, it is placed in a temporary Holding register. Then the CD1865 processor checks for error conditions, FIFO overrun, and special character match before placing the character and its corresponding status into the Receive and Status FIFOs.

7.1.2 Receive FIFO Operation

Eight bytes of FIFO are assigned to each receiver for data storage, in addition to the Receive Holding register and the Receive Shift register. Once the number of data bytes received and stored in the FIFO reaches a programmed threshold, the CD1865 can be programmed to generate a service request. See Figure 22 on page 59 for Receive Operation. The Receive FIFO Service Request threshold can be selected by programming the RxTH bits 3:0 in the Channel Option register 3. A service request threshold of one-to-eight characters can be selected. Once this threshold is defined, a service request is automatically triggered when the condition is met. It is possible that by the time the host responds to the service request, there is more data in the FIFO than the threshold level.

Figure 21. Bit Synchronization in CD1865



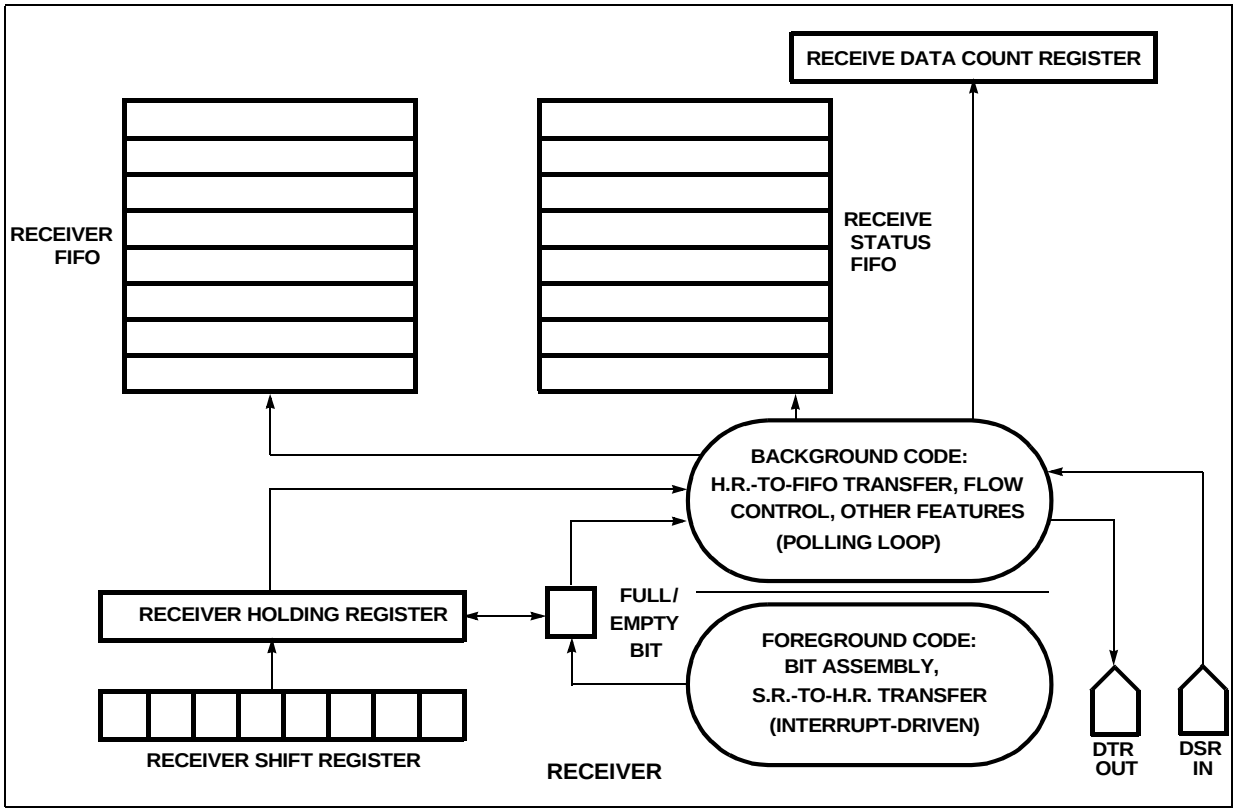
An overrun condition occurs when the new data arrives, but the Receive FIFO and the Receive Holding register are both full. The new data is lost and the overrun indication is flagged on the character in the Holding register. That character and its status including the overrun indication is eventually transferred to the host by a Receive Exception Service Request. Note that this character is good, and is the last character received before the overrun occurred.

Receiver Service Requests are enabled or disabled by the Receive Data bit in the Enable register (). Receive Data bit, when set to a '1', enables service requests to be asserted for the above causes.

The Prescaler Period Counter is a 16-bit counter clocked by the system clock. If the system clock is a 33-MHz clock, the maximum count establishes a clock tick every 1.9859 ms. The Prescaler Period should be set to generate a minimum tick period of 1.0 ms. The Receive Time-out Counter is an 8-bit counter decremental on every tick of the Prescaler Period Counter. At the maximum count per tick, the maximum time-out period is 0.506 seconds.

The Receive Time-out is always enabled to transfer data when the Receive Data Service Request is enabled. From the system applications view-point, this time-out function is important for asynchronous data transmission. This is especially true when a FIFO is in use and a service request threshold for the FIFO is set greater than one character. The Timer Service Request eliminates long response times when excessive delay between characters occurs caused either by the remote operator or due to the line being disabled. The 'No New Data' Timer Service request, which occurs after all data is transferred to the host, may be used to manage transfers from the host's receive data buffers.

Figure 22. Receive Operation



7.1.3 FIFO Timer Operations

The CD1865 uses the Receive FIFO Timer for two purposes. The first is to avoid ‘stuck’ (or ‘stale’) data in the FIFO caused by not receiving enough characters to trip the threshold, which causes a service request to be issued. The second is to signal the host that there has been a relatively long pause in received data. It is useful for the host to know that ‘no data has arrived lately’ when managing relatively large I/O buffers. This event flushes the buffer up to the host for processing.

To avoid ‘stuck’ data, each time the CD1865 moves a character into a channel’s Receive FIFO, it sets the channel’s Receive FIFO Timer to the value contained in the channel’s Receive Time-out Period register (RTPR). If the timer expires before new data arrives, a Receive Good Data sub-type service request is asserted for the channel if the Receive Data Enable bit in the is set.

The other receive timer option is to generate a service request for the first Receive Data Time-out following the transfer of all data from the channel to the host. This is called the No New Data Time-out (NNDT). This service request is a Receive Exception sub-type with a status type of ‘Time-out Exception’. There is no data character associated with the Time-out Exception status. This option can be enabled or disabled by controlling the NNDT bit in the .

If enough data arrives to fill the Receive FIFO to the level set by the RxTh bits in COR3, or if a special character arrives in the Receive FIFO and the RxSC bit of is set, the channel asserts the Receive Data Service Request without waiting for the timer to expire.

If the timer times-out and the FIFO is not empty, the ‘stale data’ condition has occurred, and the device posts a Receive Good Data Interrupt. If the timer times-out and there is no data, two conditions are checked. First, a test is made to see if the feature is enabled, if it is true, then another flag is tested to make sure this is the first time the condition has occurred. If this is true, a Receive Exception Service Request is posted. (The NNDT internal flag is armed when the FIFO is emptied).

7.1.4 Receive Service Requests

The Receive Service Request is unique as it has two sub-types; that is, it is capable of returning one of two different vectors during a service request acknowledge cycle. The two sub-types are Receive Good Data and Receive Exception. The reason there are two types within one category of service request is because while Good Data and Exceptions require different handling, they are both of equal priority and need to be serviced in the order they were received. Suppose, for example, two good characters are received, then an erroneous character, then another good character, then there must be a service request for the first 2 bytes of Good Data, then for the Exception, and then for more Good Data. If Exception Service Requests were at a different level, the erroneous character would be processed either before or after the Good Data, and not in normal sequence. Receiver Service Requests are invoked under several conditions.

Conditions that cause a Receive Good Data Service Request are:

- Receive FIFO threshold reached or exceeded
- Receive FIFO time-out — interval between character receptions exceeds time-out value

Conditions that cause a Receive Exception Service Request are:

- Receive erroneous data (parity error)
- Framing error (No Stop bit)
- No data received time-out (optional)

- Special character detection
- Break detect

Note: Data cannot be read from the Receive FIFO or the Receive Status FIFO except when the CD1865 is within the context of a Receive Data Service Request for a specific channel.

7.1.5 Receive Good Data™ Service Request

A Receive Good Data Service Request is asserted for any of the following three conditions:

1. Receive FIFO threshold reached, and the FIFO contains Good Data.
2. Receive FIFO threshold not reached, but the FIFO contains Good Data and the Receive Data Timer times-out.
3. Receive FIFO threshold not reached, but the FIFO contains Good Data and the newly arrived data contains an exception condition.

When any of these conditions occur, the modified service request vector indicates to the host that the service request is for Good Data.

It is not necessary to take all or any of the available Good Data when a Good Data Service Request is received. If a host buffer is too full to accept 8 bytes, a smaller number (even a ‘0’) can be read. Service request context is then left, and the host buffer is dealt with first. The CD1865 generates another Good Data Service Request when any of the three conditions listed above are met.

The CD1865 immediately generates another service request if the condition that caused it in the first place remains true. If no data is read, this is always the case. If some, but not all of the available data is read, Conditions 1 and 2 are not true; but Condition 3 may be true if an exception condition caused the Good Data Service Request. If this is a problem, one solution is to temporarily disable Receive Service Requests on that channel. To avoid FIFO overflow, do not delay handling the channel for too long.

7.1.6 Receive Exception Service Request

Unusual or exception conditions are reported to the host one character at a time through the Receive Exception Service Request. As with normal receive processing, the host determines the requesting channel by reading the GCR. It can then determine the specific exception(s) by reading the Receive Character Status register before performing the appropriate action. Receive Exceptions are always 1-byte deep; multiple bytes of exception conditions causes multiple Receive Exception Service Requests.

For many exceptions, it is not necessary to read the Receive Data register after the Receive Status register is read. For example, if special character detection is enabled, and the service request is for recognition of a special character, the character is known by definition because the exception code indicates the detected character or character sequence.

However, for every exception a byte is placed in the Data FIFO, even though the contents of that byte may be suspect data, and the byte is discarded at the end of the exception service routine regardless of whether it was read by the host or not. This is done to keep the Status and Data FIFOs in lock-step with each other. This is different in the case of a Receive Good Data Service Request where the user is free to read as many or as few bytes as required.

Regardless of the number or type of exceptions occurring, they are reported to the host one character at a time; that is, the number-of-bytes value in the Receive Data Count register is not meaningful. Since every error is reported individually, there is no Receive Time-out Exception generated if the only characters in the FIFOs are error or exception characters.

7.1.7 Types of Errors

There are four types of errors recognized by the CD1865: parity, framing, line break, and overrun. If parity checking is enabled, parity errors are logged in the Status FIFO and the suspect data is placed in the Receive Data FIFO. An error is also logged for framing, that is, absence of a Stop bit. In these cases, the suspect character is in the Receive Data FIFO and the appropriate status byte is placed in the Status FIFO.

When a line-break condition is recognized (zero data with zero parity, and no Stop bit), one NULL (00) character is loaded into the Receive FIFO, and a break status is recorded in the Status FIFO. Note that if odd parity is set and the bits received are all zeroes, it is marked as both a break character and a parity error. Generally when a break character is received, pre-set parity error can be ignored. No further FIFO entries are made until normal-character reception is resumed, for example, a Start bit is found. The line must go high and then back to low for this to occur.

Multiple errors in 1 byte are possible because the CD1865 evaluates the characters bit-by-bit as it receives them. For example, a parity error is detected and flagged before the CD1865 recognizes that a framing error has occurred. Parity plus framing or parity plus break error can occur, but framing plus a break error cannot occur because, if a character is received with every bit equal to a ‘0’, it is marked as a break character. If some bits are a ‘1’, but the Stop bit is missing, for example, a ‘0’, it is marked as a framing error. Thus, any one character cannot have both framing and break errors.

The length of the Stop bit is not checked by CD1865. Any Stop bit long enough to be sampled in mid-bit time as a ‘1’ is interpreted as a valid Stop bit. In addition to all of the other errors, if an overrun occurs, the Overrun Error bit is set along with other error bits.

7.1.8 Types of Exceptions

7.1.8.1 Special Character Recognition

‘Special Character Recognition’ is a feature found only on the CD1865 and other Intel data communications controllers. The on-chip processor compares every good character received with user-defined special characters stored in registers on the device. Both single-character and two-character sequence recognition is possible. This capability has several applications, including In-Band Flow Control. Special-character matches are reported to the host by a Receive Exception Service Request.

Four Special Character registers are provided per channel, allowing received characters to be compared to as many as four special characters. However, these four registers are shared between Receive Special Character Detection and the Send Special Character Command, so some planning is required for using these characters.

The full set of features and options available as part of Special Character Recognition allows for Xon/Xoff flow-control to be implemented transparently to the host, and at the same time, detect either of two other special characters in the data stream and alert the host of their arrival.

The user may individually enable any CD1865 channel to recognize special characters. There are six bits used to control the various recognition and flow-control modes.

The following four registers are used to control character recognition:

Bit Name	Register	Function
SCDE	COR3	Enables detection of special characters. Must be set for In-Band Flow Control to work.
RxSC		Enables generation of service requests. Cannot be overridden by other bits. Does not need to be set for In-Band Flow Control to work.
XonCH	COR3	Controls single- versus double-character matching.
XoffCH	COR3	Controls single- versus double-character matching.

The following table shows the effects of XonCH and XoffCH:

XonCH	XoffCH	Characters matched
0	0	Match on: any of SCHR1–4
0	1	Match on: SCHR1 or SCHR3 or (SCHR2 and SCHR4)
1	0	Match on: (SCHR1 and SCHR3) or SCHR2 or SCHR4
1	1	Match on: (SCHR1 and SCHR3) or (SCHR2 and SCHR4)

Note: The two-character pairs can share a common first character; however, the same character must be programmed in both SCHR1 and SCHR2.

Single- versus double-character recognition is controlled by XonCH and XoffCH. If single-character compare is enabled, the CD1865 compares data in the data stream against the four special characters stored in the Special Character registers (SCHR1-4). If fewer than four special characters are required, the unused Special Character register(s) should be disabled by duplicating the pattern to be matched in the unneeded register. When reporting a special character, the CD1865 always reports the lowest-number Special Character register that matches.

To set up Special Character Recognition, first set the characters to be matched in registers SCHR1-4, then set XonCH and XoffCH according to the length of match wanted. Set the SCDE bit, and lastly enable service requests by setting RxSC.

Special characters are reported to the host by placing the appropriate status word in the Status FIFO and the recognized special character in the Receive Data FIFO. In the case of a two-character sequence, only the second character is stored in the Receive FIFO. This is because there is room only for one character and preserving both is not needed as these characters are user-defined.

7.1.9 Flow-Control Characters

Automatic In-Band Flow Control of the CD1865 transmitter is a subset of the Special Character Recognition capability, so to understand both these features is important. Refer to Section 7.2 on page 68 for transmitter operation. Flow-control characters and operation are programmable on a per-channel basis. This is important to operating systems that allow users to configure their own terminal settings independently.

Because the CD1865 performs flow-control functions before the data is passed to the host, the response time required of the host to avoid data overrun is greatly reduced. Additionally, the flow-control characters can be stripped from the data stream, relieving the host from processing them.

To use automatic flow-control, the Special Character Detection (SCDE) must be enabled by bit 4 of Channel Option register 3 (COR3). This causes all error-free received data to be compared for a match with the Special Character registers (SCHR1–4). In addition, flow-control must be enabled by Transmit In-Band Enable (TxIBE, bit 6) of COR2. This causes the special characters to be interpreted as flow-control characters. For single-character flow-control sequences, SCHR1 is used as Xon and SCHR2 as Xoff. SCHR3–4 are available for use as normal special-detect characters. If two-character sequences are enabled by XoffCH and XonCH (bits 6 and 7) of COR3, SCHR1 and SCHR3 form the Xon sequence, and SCHR2 and SCHR4 form the Xoff sequence.

If flow-control characters are passed to the host, they are marked as special characters 1 or 2 in the Receive Channel Status register (RCSR). If a two-character sequence is detected, it is compressed to the second character and a status indicating a match of the first character is set. A valid two-character sequence requires that both characters be received without error; if an error occurs on the second character the first character is treated as a normal character, and this does not affect non-flow control special character detection.

Bits affecting flow control are summarized below.

Bit Name	Register	Function
SCDE	COR3	Enables Special Character Recognition.
TxIBE	COR2	Enables Automatic Transmitter Flow-Control.
FCT	COR3	Sets Transparency mode of flow-control.

XonCH	XoffCH	Xon	Xoff
0	0	SCHR1	SCHR2
0	1	SCHR1	(SCHR2 and SCHR4)
1	0	(SCHR1 and SCHR3)	SCHR2
1	1	(SCHR1 and SCHR3)	(SCHR3 and SCHR4)

The FCT bit controls whether flow-control characters are passed on to the host. It has meaning only when In-Band flow control is enabled, that is, TxIBE is set. When the CD1865 receives a flow-control character or character sequence and FCT is a ‘0’, it starts or stops the transmitter, as required, and passes the character onto the host as a Receive Exception. Since there is a one-to-one correspondence between the Status and Receive FIFO, the flow-control character detected is stored in the Receive FIFO, and a status byte indicating special-character detect is stored in the Status FIFO. If FCT is a ‘0’, RxSC must be set to enable service requests to be issued to the host. Otherwise, flow-control characters cannot be passed as Receive Exceptions and is instead passed as Good Data.

If the FCT bit is a ‘1’, the CD1865 still starts or stops the transmitter, as required, but the character(s) are discarded, and no exception is posted. In either case, the flow-control status of the transmitter (on or off) is maintained by the CD1865 in the Channel Control Status register (CCSR).

The FCT bit makes it possible to support ‘escaping’ of flow-control characters. Some systems follow a convention where two identical flow-control characters in a row indicates that flow control is not to be performed, but rather one flow-control character is to be kept in the normal received-data stream, and the other ‘escape’ character is to be discarded. If the CD1865 is in such a system, set the FCT bit to a ‘0’, allowing flow-control characters to pass onto the host. When the host detects two flow-control characters in a row, it simply restores the proper flow-control state of the channel and discards one of the characters. However, for most systems the FCT bit can be set to a ‘1’, reducing loading on the host.

7.1.9.1 No New Data Received Time-Out

It is sometimes useful for the host to sense that ‘no data has arrived lately’, when managing relatively large I/O buffers. This event is used to flush the buffer up to the host for processing. One of the receive timer options, No New Data Time-out (NNDT), generates a service request for the first Receive Data Time-out following the transfer of all data from the channel to the host. This service request is a Receive Exception sub-type, and can be enabled or disabled by controlling the NNDT bit in the . Refer to Figure 23 on page 67 for the timer logic.

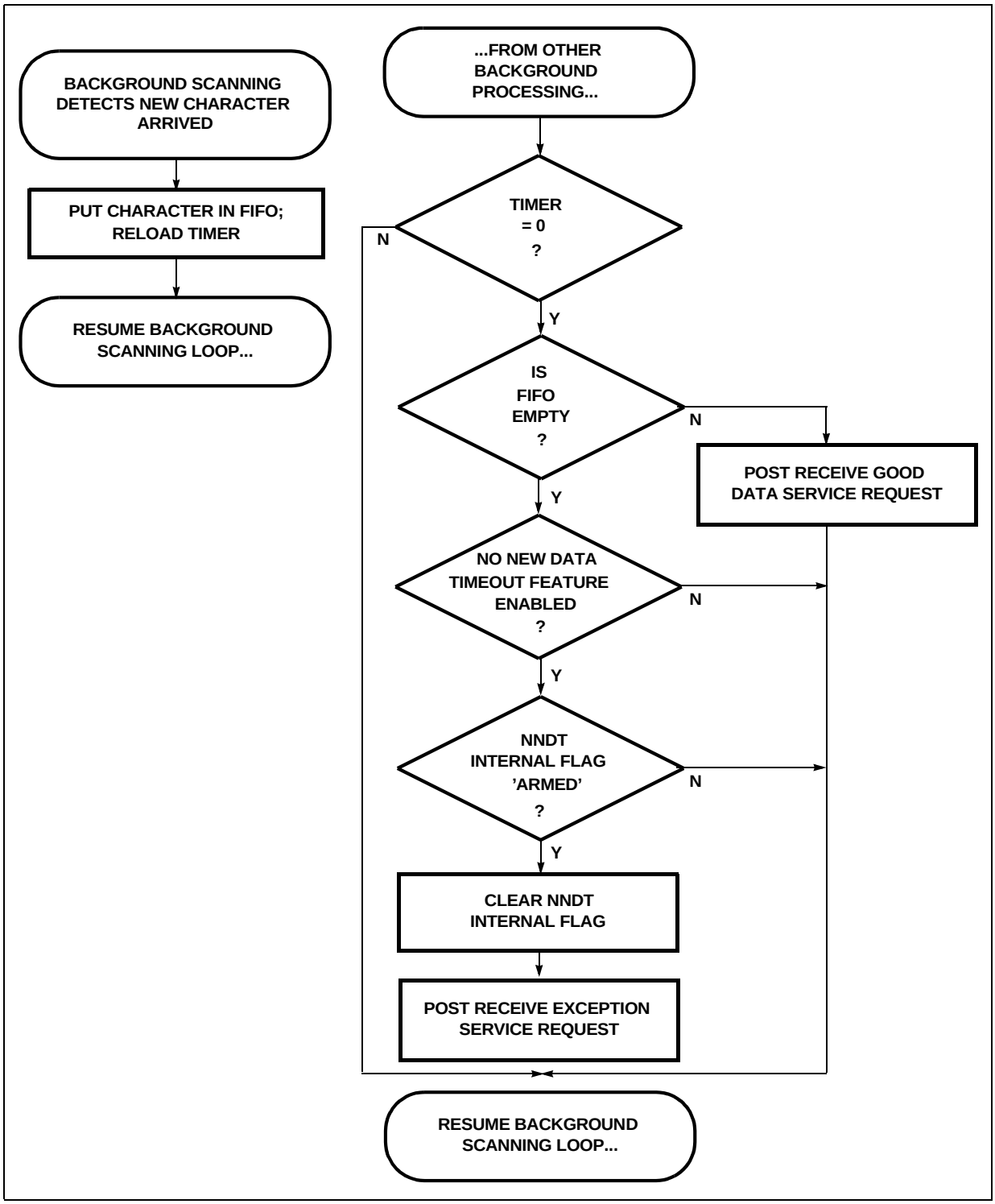
The timer is started only on data arrival. If the CD1865 processor determines that the Receive FIFO is empty, the timer has expired, and there is a previous receipt of Good Data (and the timer feature is enabled), a Receive Exception occurs with a status indicating that a time-out has occurred.

If the last Receive Exception Service Request was triggered by a time-out (to avoid ‘stale’ data) the No New Data Time-out Service Request occurs immediately after the Data Transfer Service Request completes. If the last service request was triggered by reaching the threshold, the timer

still has to expire so that some time passes before the No New Data Service Request occurs. Likewise, if the last service request was triggered by some other error, such as parity, the timer still has to expire so that some time passes before the No New Data Service Request occurs.

The No New Data function should not be confused with the time-out that occurs when there is Good Data in the FIFO but the threshold has not been reached and the timer expires. This event is a Receive Good Data Service Request, and not a Receive Exception event. Timing-out to transfer Good Data before it becomes ‘stale’ is standard, and it cannot be turned off by the user.

Figure 23. No New Data Timer Logic



7.1.10 Programming Notes

If a special condition (for example, framing or a parity error) occurred on a special character, the CD1865 does not interpret this character as matched. Flow-control characters that are processed and discarded because FCT is set never cause an overrun.

Special Character Recognition only occurs on characters that have no other problems or errors. There is one case where the CD1865 does not find a special character even though the character has been correctly received. If a good character arrives as the ninth character (for example, the FIFO is full), it stays in a Holding register. If another character arrives, the good character in the Holding register has its status marked as ‘overflow’, indicating that it is the last good character received; however, it is not recognized as a special character.

There are two cases where the CD1865 might not detect a two-character sequence. If the first character has been found, but no other character has been received for a long period of time and the Receive Time-out event occurs, no match is found because the first character is flushed up to the host. If special-character detection is disabled by clearing SCDE just when the CD1865 has received the first two-character special-character sequence, but has not received the second character yet, the first character is lost.

7.2 Transmitter Operation

7.2.1 Basic Operation

Refer to Figure 24 on page 69 for a diagram of transmitter operation. Upon power-on reset, all transmitters are disabled with their Transmit Output held in the ‘Mark’ or a logic ‘1’ condition. Other channel parameters are undefined. The minimum configuration of a channel for transmission consists of specifying the bit rate, parity, and number of Stop bits. In-band and Out-of-Band Flow Control should also be set as required. Next, set either (or both) of the service request enable bits. Then issue the Transmit Enable Command and either of two service request enable bits. For normal operation, set the TxRDY bit. This causes a service request to be issued when the FIFO is empty. Since on power-up the FIFO is empty, a service request is received (less than 1 ms.), and at that time data can be transferred to the FIFO. Data can not be transferred to the FIFO as part of channel initialization; instead one has to be in the service-request routine to do this. Refer to the Section 5.3 for details.

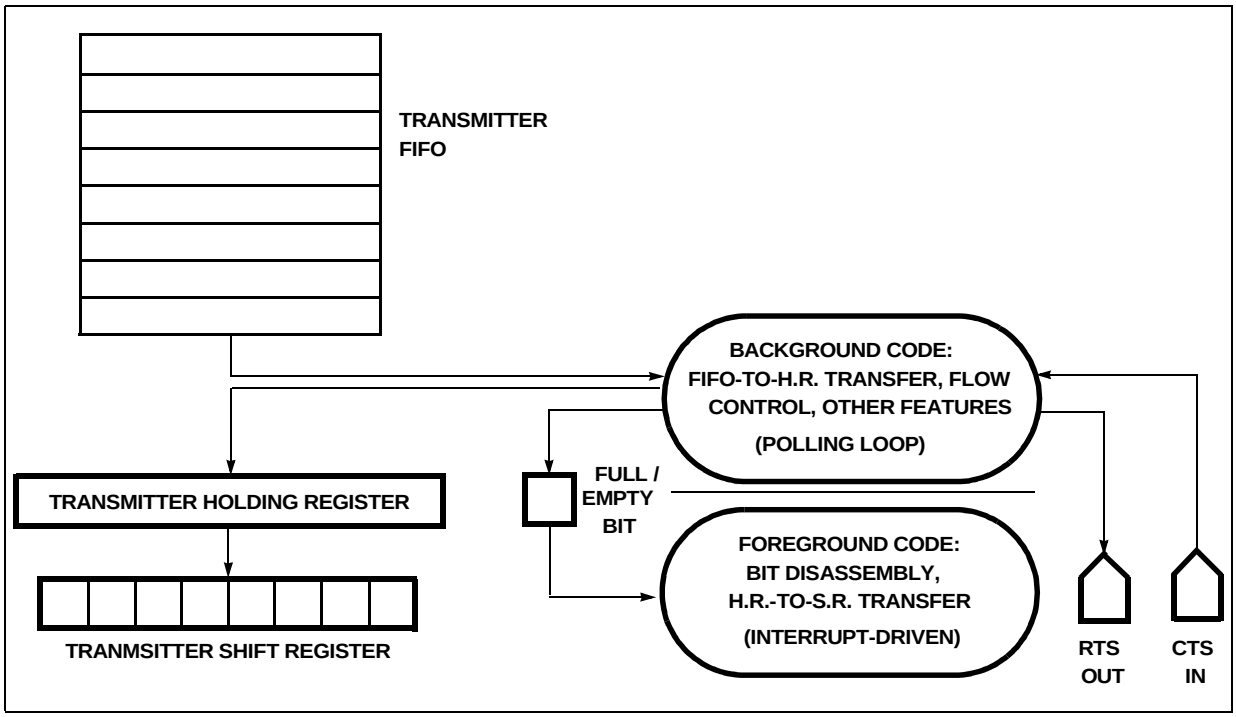
Once the channel is initialized and serviced, and a character is written into the Transmit FIFO, the transmitter starts to transmit by first sending the Start bit (space or a logic ‘0’) followed by the data character according to predefined character length, least significant bit first. An optional parity bit (none, odd, even, or forced) is appended followed by the final Stop bit (a logic ‘1’ or a ‘Mark’). The length of the Stop bit can be one, one-and-a-half, two, or two-and-a-half bit-times long.

The transmitter continues sending characters one after the other until the Transmit FIFO is empty. When the Transmit FIFO is empty and the last character is sent, the transmitter stops transmission and holds the TxD Output in the ‘Mark’ (1) condition. Transmission resumes another character is in the FIFO.

In some cases it must be determined if the channel is completely done transmitting the last bit of the last character — for instance, before changing the bit rate. In such a case, the service request is to be issued only when the last character is sent, rather than when the FIFO is empty. In this case, instead of setting the TxRDY bit, set the TxMpty bit. This causes a service request to be issued only when the transmitter is completely empty.

For details on transmitter flow-control operation, refer to the Section 7.3 on page 72.

Figure 24. Transmitter Operation



7.2.2 FIFO Operation

An 8-byte FIFO is provided for each transmit channel. In addition to the 8-byte FIFO, the CD1865 also contains a Transmit Holding register and the Transmit Shift register for each channel. However, when servicing a Transmit Service Request, only up to eight characters can be written into the Transmit Data register (TDR) consecutively.

7.2.3 Transmit Service Requests

Generating a Transmit Service Request depends on control bits in the Enable register (). Setting the TxRdy bit of the specifies that a Transmit Service Request be generated when the FIFO is empty. When this condition occurs, there is still one character in the Transmit Holding register and another character in the Transmit Shift register. The host CPU, therefore, has up to two-character times to respond before the transmitter output goes into the idle (Mark) condition.

Setting the TxMpty bit instead of the TxRdy bit of the specifies that a Transmit Service Request be generated only when the FIFO, the Transmit Holding register, and the Transmit Shift register are empty. When this condition occurs, it means that all characters are completely transmitted and the channel can now be re-configured. It is recommended that one of the two bits be set as needed, but do not set both bits at the same time.

End of a service request must be signalled to the CD1865 by writing to the End Of register ().

7.2.4 Special Transmitter Commands

The CD1865 is capable of sending special characters preemptively (bypassing the FIFO): sending break characters and inserting delays or pauses either between characters or to lengthen a break. There are two basic mechanisms the CD1865 uses for these ‘Send Special Character’ and ‘Embedded Transmit Command’ functions.

7.2.5 Special Character Transmission by Send Special Character Command

Selected special characters, or two-character sequences, may be transmitted preemptively by setting the appropriate bits in the Channel Command register (CCR). The Send Special Character (SEND SP CH) bit of the CCR, when set, initiates the Send Special Character Command. SSPC0–2 bits of the CCR then specify which character or two-character sequence is used. The choice of a single- or two-character sequence is determined by the XonCH and XoffCH bits of COR3.

When a Send Special Character Command is given, the CD1865 inserts the special character(s) into the data stream immediately following the current character in the Transmit Holding register. Thus, it is ensured that the special character begins transmitting within two-character times after the command is issued. The Send Special Character Command overrides all other flow-control modes, including the state of TXEN and CTS*. Generally this is the preferred case. However, sample CTS* or CD* in some applications to determine if it is okay to send a character before invoking the Send Special Character Command.

The CCR is reset by the CD1865 as an acknowledgment of the command. A new command must not be issued if the CCR contents are non-zero. A send special character command is recognized and cleared within 125 µs (at 15 MHz, proportionally longer at lower clock speeds), unless a break is being sent. If a break is being sent, the special character is not sent until after the break time is complete.

7.2.6 Embedded Transmit Commands

The CD1865 may be enabled to recognize certain ‘escape’ sequences as commands embedded in the Transmit Data Stream. These commands are issued to introduce a time delay between characters, to insert an idle period during the transmission, or to send a break on the line.

These capabilities are enabled on a per-channel basis by setting the Embedded Transmit Command (ETC) bit in the Channel Option register 2 (COR2). The ‘null’ (00) character is used as the controlling character to initiate the special action. To preserve data transparency, two mechanisms are provided to allow the null character to be sent as data. If the host must transmit a null character as data, either the ETC mode may be disabled, or the null character may be preceded by a null, that is, ‘00 00’ causes one-null character to be sent. If the ETC bit is not set, the ‘00’ character has no effect, and it may be sent as ordinary data. ETC mode may be enabled or disabled ‘on-the-fly’.

The CD1865 uses the Transmit Timer to generate time delays between characters in the output data stream. It is also used to extend the duration of a line-break transmit condition when the delay is inserted between the ‘Start Break’ and ‘Stop Break’ embedded-transmit commands. All of the timers count ticks are determined by the Prescaler Counter. The two eight-bit Prescaler Period registers (PPRH and PPRL) determine the real-time length of a tick. A tick is the period of the CD1865 System Clock Input (CLK) multiplied by the Period registers’ contents.

7.2.7 Sending Breaks

Line breaks may be sent by embedding the following sequences in the data stream (all values are given in Hex):

00 81 **Send Break:** Enter line-break condition for at least one character time. The line enters the break condition and stay there until one of the following conditions is met:

1. Another character needs to be sent.
2. If the Insert Delay Special Character Sequence immediately follows the Send Break Sequence, the duration of the break transmission is extended by the amount of the programmed delay. The Insert Delay Sequence is: 00 82 xx. This inserts a delay of ‘xx’ (interpreted as an unsigned binary number) times the programmed timer ‘tick’ set by the Prescaler Period registers. Multiple insert delay commands can be executed consecutively by the CD1865 to allow delays of arbitrarily long length. If ‘xx’ is a zero, no delay is inserted.
3. The Stop Break Sequence ‘00 83’ is encountered next. This sequence is optional, and exists to provide a way to terminate a break without actually sending another character. If another character is being sent anyway, no Stop Break is required.

If there is no more data to be sent, the TxD pin remains in the state it was left in by the last character. Since the Stop bit is always a ‘1’, the line is left in the idle state after any character, except for the break character. The break character leaves the line in the ‘0’ state until more data needs to be sent. Long breaks can be sent by simply sending one break and then waiting. To terminate the break, send the Stop Break Sequence or send another character.

Sending long breaks has precedence over the Send Special Character Command, for example, the time delay duration must pass before the special character is sent.

7.2.8 Sending Inter-Character Delays

In some applications it is desirable to pause between characters. For example, certain types of electro-mechanical teletype equipment cannot handle characters continuously at their specified bit rate. To accommodate this, the CD1865 allows insertion of a delay between characters.

The user embeds an escape sequence into the Output Data Stream to generate delays between characters. When the CD1865 encounters the Insert Delay Escape Sequence, it sets the Transmit Timer to the value contained in the Escape Sequence. When the timer expires, the CD1865 loads the next character into the Transmit Shift register and resumes output (unless the next character begins another Escape Sequence). The Escape Sequence for an inserted delay consists of three characters: ‘00’, ‘82’, and ‘tt’. The time-out value ‘tt’ is expressed in timer ticks.

7.2.9 Summary of Special Transmitter Commands

The ETC bit in COR2 must be set to enable the following functions:

Char. Sequence	Effect
00 00	Send one-null character

Char. Sequence	Effect
00 81h	Send one-character time of line break
00 82h xxh	Delay for ‘xx’ prescaler time ticks (for example, Transmit Timer Value is ‘xx’)
00 83h	Stop break

7.3 Flow Control

Variations in response times and system data transfer rates between systems communicating across asynchronous interfaces give rise to a need to control the flow of data between them. Systems typically are implemented with a receive buffer for temporary storage of data. When this buffer is nearly full, the receiving computer ‘flow-controls’ the remote transmitter. When, after processing the existing data, more buffer space is available for the receive process, the receiving computer signals the remote to resume transmission.

Flow control is implemented in one of two ways — ‘out-of-band’ or ‘in-band’ signaling. Out-of-band signaling is a hardware-based mechanism, performed by extra wires such as the RTS/CTS and DSR/DTR pairs. It has the advantage of complete independence from the data stream. However, it is not always possible to provide all of the wires necessary to support Out-of-Band Flow Control. Also standards for implementing Out-of-Band Flow Control vary widely.

In-Band Flow Control works by inserting special flow-control characters into the stream of data being sent. It has the advantage that only the data circuit is required, thus only two wires are needed. The disadvantage of In-Band Flow Control is that the two communicating computers must perform additional functions, specifically, they must monitor the data stream for flow-control characters and take the appropriate action. This can be quite burdensome because the host computer that receives a flow-control command must recognize this event quickly and respond in a timely manner to avoid overrun at the remote receiver.

Although there are advantages and disadvantages to each system, in general the trend is toward In-Band Flow Control. This is because it is more useful than Out-of-Band Flow Control over a wider range of applications, such as communication by modems.

The CD1865 provides significant performance advantages over conventional solutions during both the receive processing of and the transmission of flow-control characters. It does this by handling almost all flow control automatically, without host intervention. It also provides tools to make host intervention, when required, much easier. Because the CD1865 performs flow-control functions automatically, before the data is passed to the host, the response time required of the host is substantially reduced. The possibility of data overrun is also reduced. Additionally, the flow-control characters themselves can be stripped from the data stream, relieving the host from processing them. The flow-control status of the transmitter is always available to the Host as a bit in the Channel Control Status register (CCSR).

7.3.1 Receiver Flow Control

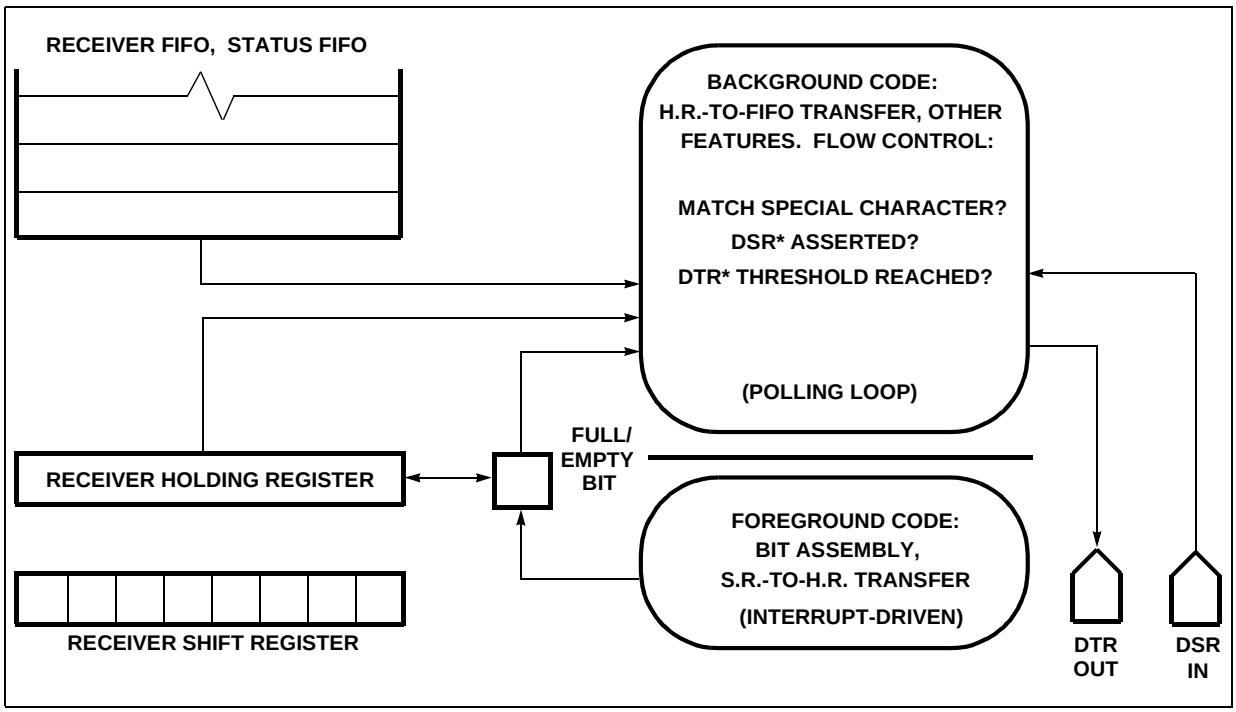
The CD1865 provides both In-Band (Xon/Xoff) and Out-of-Band Flow Control functions for ensuring that the receiver does not overflow. In-Band Flow Control is semi-automatic and helps the host manage its buffer size. Out-of-Band Flow Control is fully automatic and can be used to prevent the CD1865 Receive FIFO from overflowing. Figure 25 on page 73 diagrams the receiver flow-control logic.

When the CD1865 receiver is too busy, the transmitter can be used to send Xoff/Xon to the remote device. This Receiver Software (In-Band) Flow Control is covered in Section 7.3.3 on page 74.

The CD1865 transmitter can be controlled by the remote device. This Transmitter Software (In-Band) Flow Control is covered in Section 7.3.6 on page 76.

The current flow-control status is always available to the host. It is stored in the Channel Control Status register (CCSR). Two bits, Receive Flow-on and Receive Flow-off, show whether the last flow-control command sent by the CD1865 was on or off. As long as the receiver is enabled, the CD1865 continues to receive any data sent regardless of whether it has requested the remote to shut off.

Figure 25. Receiver Flow-Control Logic



7.3.2 Receiver Hardware (Out-of-Band) Flow Control

Out-of-Band Flow Control uses the Modem Handshake signal (DTR*) to control the flow of data. Whenever the Receive FIFO reaches a user-defined threshold, DTR* is negated. This event can be used to signal the remote to stop sending characters. The threshold is set by four bits in the Modem Control Option register 1, and can be any level from one to eight, or disabled. The DTR* pin is also negated whenever DTR* mode is set and the channel is disabled or reset. If DTR* mode is not set, the DTR* pin is not changed by the CD1865, and remains at whatever value the host sets it to.

While it is possible to set the DTR* threshold lower than the service request threshold, the part operates as though the DTR* threshold was the same as the service request threshold. If the DTR* threshold is set lower, it is ignored, and DTR* negates when the service request threshold is reached. If required, set the DTR* threshold to a ‘1’, and then it ‘tracks’ the other threshold automatically.

The receiver monitors the state of DSR* (if enabled) and ignores data on the Receive Data pin if DSR* is negated. This feature is controlled by the DsrAE bit, bit 0 of Channel Option register 2 (COR2).

7.3.3 Receiver Software (In-Band) Flow Control

Host receive buffers often cannot keep pace with data being received. The CD1865 transmitter can be used to send flow-control characters to the remote device. This avoids over-flowing the receive buffers in the host. However, transmitting flow-control characters is an additional complication and source of delay when using conventional devices. As the host's receive buffer becomes full, the transmit process must be flagged to insert a flow-control character (or sequence) in the Transmit Data Stream. Any data already in the Transmit FIFO is transmitted ahead of the flow-control character, increasing the response time at the remote end.

With the CD1865, In-Band Flow Control of the remote system is semi-automatic; two commands (Send Xon, Send Xoff) can be issued by the host whenever the host wants to flow-control the remote. These special commands make host programming and buffer management easier because it allows the flow-control character(s) to be sent as the next character, regardless of the contents of the Transmit FIFO or host transmit buffers.

Flow-control characters are transmitted by the send special character command in the Channel Control register (CCR). The lower-three bits in the command determine which of the four-special characters are to be sent. If two-character flow control sequences are enabled, requesting either SCHR1 or SCHR2 causes the appropriate two-character sequence to be transmitted. Refer to Section 7.2.5 on page 70 for Special Character Definition details. Special characters are transmitted regardless of the state of transmit enable or transmit flow control. Transmitting flow-control characters can be handled independently of the current state of the transmit channel. In sending special characters, the CD1865 bypasses any data already in the Transmit FIFO, thereby minimizing delay in transmitting flow-control characters. The maximum delay is two-character times. However, if a break is currently being transmitted, the CD1865 waits for the break transmission to terminate before the special character is transmitted, regardless of the length of the break.

The CD1865 keeps a copy of the current state of the receive flow in the CCSR. Two bits are used to indicate the current state of the channel regarding flow control: RxFloff and RxFlon. RxFloff and RxFlon are meaningful only when the CD1865 is flow-controlling the remote. Whenever an Xoff is transmitted, RxFlon is cleared and RxFloff is set. When a subsequent Xon is transmitted, RxFloff is cleared and RxFlon is set. When data is received from the remote, RxFlon is cleared.

The '00' state is provided as an aid to the programmer in determining whether there might be a problem in a communications link. If RxFlon remains set during normal operation, it could indicate that the remote did not correctly receive the last Xon.

If flow-control characters are sent by the host by embedding them in the Transmit FIFO rather than using the Send Special Character function, the CD1865 flow-control logic does not sense them, and the CCSR is not affected.

The table below summarizes the meaning of RxFloff and RxFlon.

RxFloff	RxFlon	Meaning
1	1	Illegal mode.
1	0	Xoff is last flow-control character sent (flow off).
0	1	Xon is last flow-control character sent (flow on).
0	0	Flow is on, data has been received.

7.3.4 Transmitter Flow Control

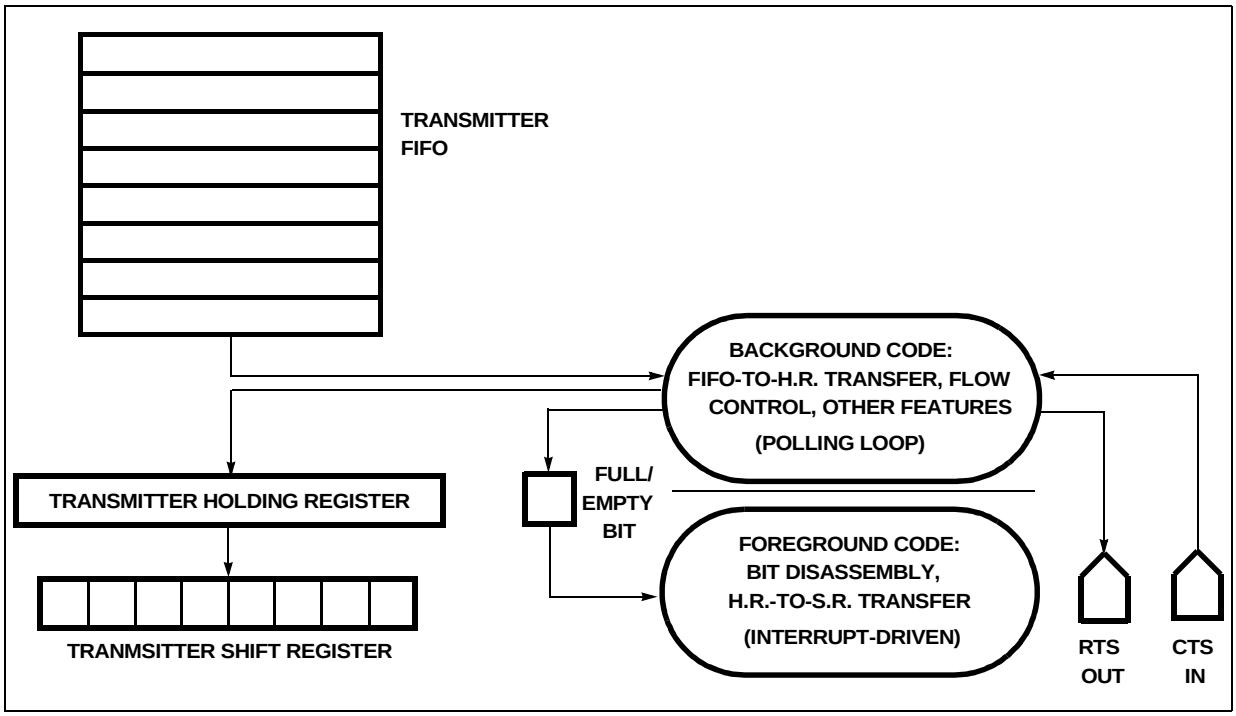
The CD1865 provides both automatic In-Band (Xon/Xoff) and Out-of-Band Flow Control functions. In-Band Flow Control recognizes special characters or character sequences for Xon and Xoff control embedded in the data stream. Out-of-Band Flow Control uses the modem handshake signals, RTS/CTS, to control the flow of data. Both types of flow control are implemented between the Transmit FIFO and the Transmit Holding register, not between the Transmit Holding register and the Transmit Shift register. Figure 26 on page 76 diagrams the transmitter flow-control logic.

All automatic flow-control functions are controlled by bits in Channel Option register 2 (COR2), except DTR threshold, which is controlled by Modem Change Option register 1 (MCOR1). Channel enable and flow-control status is stored in the Channel Control Status register (CCSR). A TxEn bit shows the enabled status of the channel's transmitter. Two bits, TxFloff and TxFlon, are used to indicate the current state of the channels' flow control.

Once the Automatic Flow-Control Modes are invoked by the host, all actions are transparent to the host. If receipt of flow-control characters by the host is not required, the Flow-Control Transparency bit of COR3 may be set to not pass received flow-control characters onto the host. If TxIBE is set, the CD1865 implements the flow-control function on the transmitter regardless of the FCT mode. The host can review the status of the channel by reading the Channel Control Status register.

If flow-control status is needed by the host, the SCDE and RxSC Control bits must be set and the FCT bit must not be set. A special character detect status and the special character is presented to the host by a Receive Exception Service Request. If the host wishes to manually flow-control the transmitter, it can do so by using the TxEn bit, which stops transmission after the current character completes.

Figure 26. Transmitter Flow-Control Logic



7.3.5 Transmitter Hardware (Out-of-Band) Flow Control

Transmit Out-of-Band Flow Control is performed automatically by the CD1865 by the CTS* pin, if the CTS Auto Enable (CtsAE) mode is enabled in bit 1 of COR2. In this mode, before a character from the FIFO is transmitted, the CTS* pin is tested, and, if inactive, transmission is delayed. Since flow control is implemented between the FIFO and the Transmit Holding register, when CTS* is negated, it is possible to get both the current character being sent and the character in the Transmit Holding register.

However, the Send Special Character Command (that is, Xon and Xoff) overrides CTS* inactive. This is generally preferred; however, in some applications sample CTS* or CD* before sending a special character.

To complete the handshake with a remote device, an RTS Automatic Output (RtsAO, bit 2) mode is also provided. This causes the RTS pin to be asserted throughout any data transmission: normal, break, and special characters. The RTS pin is activated whenever there is data in the FIFO and transmitter registers. It is held active until after the last Stop bit of the last character is transmitted.

7.3.6 Transmitter Software (In-Band) Flow Control

The CD1865 transmitter can be programmed to respond automatically to flow-control characters received by the receiver. This feature requires no host assistance and substantially reduces host processing requirements. If this Automatic mode is enabled, when the remote unit transmits an Xoff character to the CD1865 (to prompt the CD1865 to suspend transmission), the CD1865 terminates the transmission. The CD1865 may require approximately 500 microseconds (~2 character-times at 38.4 kbps) after receipt of the Stop bit to recognize that the character it has

received is a flow-control character and set its internal flag to stop transmission. Transmission actually stops as soon as the characters in the Transmit Shift register and Transmit Holding register are shifted out.

To enable In-Band Flow Control, two bits must be set. First, the Special Character Detection (SCDE) must be enabled by bit 4 of Channel Option register 3 (COR3). This causes all error-free received data to be compared for a match with the Special Character registers (SCHR1–4). Second, flow control is enabled by Transmit In-Band Enable (TxIBE, bit 6) of COR2, the special characters are interpreted as flow-control characters.

Different flow-control protocols use either single- or two-character sequences for the Xon and Xoff functions. For single-character flow-control sequences SCHR1 is used as Xon, SCHR2 as Xoff, and SCHR3-4 as normal special detect characters. If two-character sequences are enabled, by XoffCH and XonCH (bits 6 and 7) of COR3, SCHR1 and SCHR3 form the Xon sequence and SCHR2 and SCHR4 form the Xoff sequence.

Many operating systems allow users to define their own terminal's flow-control settings independently. The CD1865 allows flow-control characters to be programmed on a per-channel basis.

The FCT bit controls whether flow-control characters are passed on to the host. When the CD1865 receives a flow-control character or character sequence and FCT is a '0', it starts or stops the transmitter as required, and passes the character on to the host as a Receive Exception Service Request. Since there is a one-to-one correspondence between the Status FIFO and the Receive Data FIFO, the flow-control character detected is stored in the Receive Data FIFO, and a status byte, indicating special character detect, is stored in the Status FIFO.

If the FCT bit is a '1', the CD1865 still starts or stops the transmitter as required, but the character is discarded, and no exception is posted. In either case, the flow-control status of the transmitter (on or off) is maintained by the CD1865 in the Channel Control Status register (CCSR).

If flow-control characters are passed to the host, they are marked as special characters 1 or 2 in the Receive Channel Status register (RCSR). If a two-character sequence is detected, it is compressed to the second character and a status indicating a match of the first character is set. A valid two-character sequence requires that both characters be received without error. If an error occurs on the second character, the first character is treated as a normal character, and the second character is reported as an error by a Receive Exception Service Request.

Bits affecting flow control are summarized in the table below:

Bit Name	Register	Function
SCDE	COR3	Enables Special Character Recognition.
TxIBE	COR2	Enables Automatic-transmitter Flow Control.
FCT	COR3	Sets Transparency mode of flow control.
IXM	COR2	Sets implied Xon mode

XonCH	XoffCH	Xon	Xoff
0	0	SCHR1	SCHR2
0	1	SCHR1	(SCHR2 and SCHR4)
1	0	(SCHR1 and SCHR3)	SCHR2
1	1	(SCHR1 and SCHR3)	(SCHR2 and SCHR4)

The remote device can signal the CD1865 to resume transmission in one of two ways depending on the setting of the Implied Xon mode (IXM) option bit COR2. When the IXM bit is set, the CD1865 resumes transmission upon receipt of any character, for example, each character is an implied Xon. In Implied Xon mode it is assumed that if the remote is capable of transmitting data, it is able to receive as well. If a character is treated as an implied Xon, no special status is recorded in the RCSR, and the TxFlon bit is not set in the CCSR. An implied Xon character is not stripped if flow-control transparency is enabled.

When the IXM bit is not set, the CD1865 only resumes transmission upon receipt of an Xon character. In addition, the host may force a resumption of transmission by issuing a Transmit Enable Command, which clears the TxFloff bit. The Xon and Xoff characters or character sequences are equal in a Toggle mode. There is no special bit to enable this mode. The CD1865 detects this mode whenever the Xon character equals the Xoff character, and it implements Toggle mode automatically.

In Toggle mode, whenever the special character is received, the current state of flow control is toggled. If flow control transparency is set, the character is dropped. If not in flow-control transparency, the character is passed to the host. If it is a single character, the special character status is ‘1’ and the character is put in the Receive Data FIFO. In two-character sequence, the second character is placed in the Receive Data FIFO along with special character ‘1’ in the Status FIFO.

The TxFloff and TxFlon bits indicate channel status when the remote device is flow-controlling the CD1865 transmitter. When the remote requests the CD1865 to stop transmission, the CD1865 sets the TxFloff Status bit in the CCSR. If TxFloff is set, the last flow-control character received was a flow-off. When the remote sends an explicit flow-on character, the CD1865 clears the TxFloff bit, and set the TxFlon bit. (If flow is resumed because of implied Xon, TxFloff is cleared, but TxFlon is not set). When the CD1865 resumes transmission, the TxFlon bit is cleared. Transmit Flow Status bits is also cleared by enabling or disabling the transmitter or resetting the channel.

This is summarized in the table below:

TxFloff	TxFlon	Meaning
1	1	Illegal
1	0	Transmitter is flow-controlled off
0	1	Transmitter on, no data sent yet
0	0	Transmitter on, CD1865 has sent data, or implied Xon has occurred. This is also the ‘normal’ state of these bits when flow control is not being used

7.4 Modem Signals and General-Purpose I/O

Each channel of the CD1865 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 have been chosen to provide an easy reference for systems designers. In fact, they are all simply general purpose inputs and outputs (if automatic Out-of-Band Flow Control is not used) that can be individually controlled by the modem signal value register(s). Since they are general purpose, system designers may choose to connect the pins in any way that suits the application.

However, when the system software design chooses to make use of the automatic Out-of-Band Flow Control with the pins, then the signal naming convention no longer holds true in some cases, depending on whether the device is used as DCE or DTE. In this case, it is best to think of the pins in terms of their actual uses within the CD1865 and connect them accordingly, without regard to their names. The RTS* and CTS* pins are associated with the transmitter and the DTR* and DSR* pins are associated with the receiver. The table below shows Intel's recommended signal hook-up if automatic, Out-of-Band Flow Control is required.

DCE	DTE	CD1865 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 CD1865 is designed to be a DCE and automatic Out-of-Band Flow Control is required, the pin labeled DTR should be connected to remote CTS input. If the CD1865 is to be used as the DTE side, then the CD1865 CTS output would be connected to the remote CTS input.

Note that if automatic Out-of-Band Flow Control is implemented, the activity of DTR and DSR pins do not implement the function assigned to those signal names by the signalling conventions of the CCITT and other standards organization. These names would only apply to these pins if they are under program control and not under automatic CD1865 control. In fact, the 'DTR' function, as defined, enables the modem to go on- and off-line, depending on the state of the pin. If automatic control is used, then DTR would go inactive when the receive FIFO reached the programmed threshold thus causing the modem to drop the connection (carrier) to the remote, which would not be the correct function. Refer to Section 7.3 for details on operation of modem pins in flow-control applications.

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)

Modem pins are implemented as I/O ports accessible by either the CD1865 processor or the host. The modem pins are not connected directly to the transmit or receive hardware. When a user programs out-of-band modem functions to be active, the CD1865 processor reads from and writes to these pins. Specifically, when RTS* and CTS* are being used for transmit flow control, the CD1865 processor asserts RTS* and senses CTS*, as required. Likewise, when configured to do so, the Receive FIFO negates DTR* when full. The host should not be allowed to re-assert it inadvertently. The host is not 'locked out' of accessing these bits; care should be taken so that these bits are not written to, causing the system to malfunction.

The user has direct control over the RTS* and DTR* Outputs and can sense the state of CTS*, CD*, and DSR* Inputs through the Modem Signal Value register (MSVR). Since the host is accessing these pins directly, there is no delay in the host's ability to detect a level change. DTR* and CD* depend on the state of the DTRSEL input.

When the CD1865 is programmed to detect level changes and generate service requests when level changes occur, it does so in firmware by reading the pins and comparing 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 under worst-case conditions is approximately 2 ms. When the CD1865 is performing this function, the modem pins are periodically sampled rather than continuously monitored; as such they have very little sensitivity to noise, which is desirable in data communication applications. However, in extremely noisy applications, re-read a modem line which has caused a Modem Signal Change Service Request to verify that it has indeed changed and is not merely malfunctioning. This eliminates even the slight possibility of a noise pulse causing erratic operation.

When the CD1865 is monitoring modem pins to control transmit or receive functions, it does not rely on the previously stored value, but 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 CD1865 makes this decision when moving characters from the FIFO to the Holding register, not from the Holding register to the Shift register.) Refer to Section 7.3 on page 72 for flow-control details.

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

7.4.1 Generating Service Requests with Modem Pins

The CD1865 can generate service requests when any one of the input pins changes state. Either or both edges may be detected by setting bits in the two Modem Change Option registers (MCOR1 and MCOR2). For each pin, the user can individually enable on-to-off or off-to-on transition detection of the inputs. When the CD1865 detects such a transition, it sets the corresponding bit in the Modem Change register. If the corresponding bit in the channel’s set, the CD1865 asserts its * output. The user must clear the Modem Change register during the service request service routine before writing to the .

The CD1865 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 CD1865 scanning loop, changes may not be detected unless they are several hundred microseconds long. Modem Input pins can be used for purposes such as detecting the closing of a switch. However, the relatively slow speed of response should be taken into account when using Modem Input pins for this purpose. The CD1865 does not latch the Modem input signals.

7.4.2 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 them as any I/O port.

7.5 Testing the CD1865 — Loopback Tests

The CD1865 performs a basic internal self-test whenever it is reset. This test provides a reasonable degree of confidence that the CD1865 is functioning satisfactorily. There are two additional tests that can be performed by the user to further ensure complete functionality. These two test modes

are Local and Remote Loopback. Used together with diagnostic firmware in the host system, the Loopback Modes provide very thorough test coverage of all CD1865 functional blocks: the CD1865 processor, ROM, RAM, bus interface, transmitters/receivers, and random logic.

Local Loopback Mode

Local Loopback mode is a ‘silent’ loopback, for example, data being sent by the transmitter is internally connected to the receiver without reaching the external TxD pin. Generally, this is advantageous because it allows diagnostic software to operate without causing unwanted effects on any remote device that may be connected to the serial line. During local loopback, the TxD pin is in the ‘mark’ (a logic ‘1’) state. If non-silent loopback is also needed, it can be easily implemented externally with an AND gate or a jumper plug on the serial connector.

Local Loopback mode is invoked by setting the LLM bit in the Channel Option register 2 (COR2) and then issuing a channel command to tell the CD1865 that COR2 has changed. When in this mode, the channels TxD Output is internally looped back to the channel’s RxD Input. However, all other channel parameters including modem pins continue to work independently and normally. Receive special character recognition, overflow handling, and other options may be tested by using the Local Loopback mode and transmitting the appropriate character sequences. As shown in Figure 27 on page 82, the loopback connection is directly from the TxD signal to the RxD signal, for example, all transmit and receive logic is tested except the actual I/O buffers.

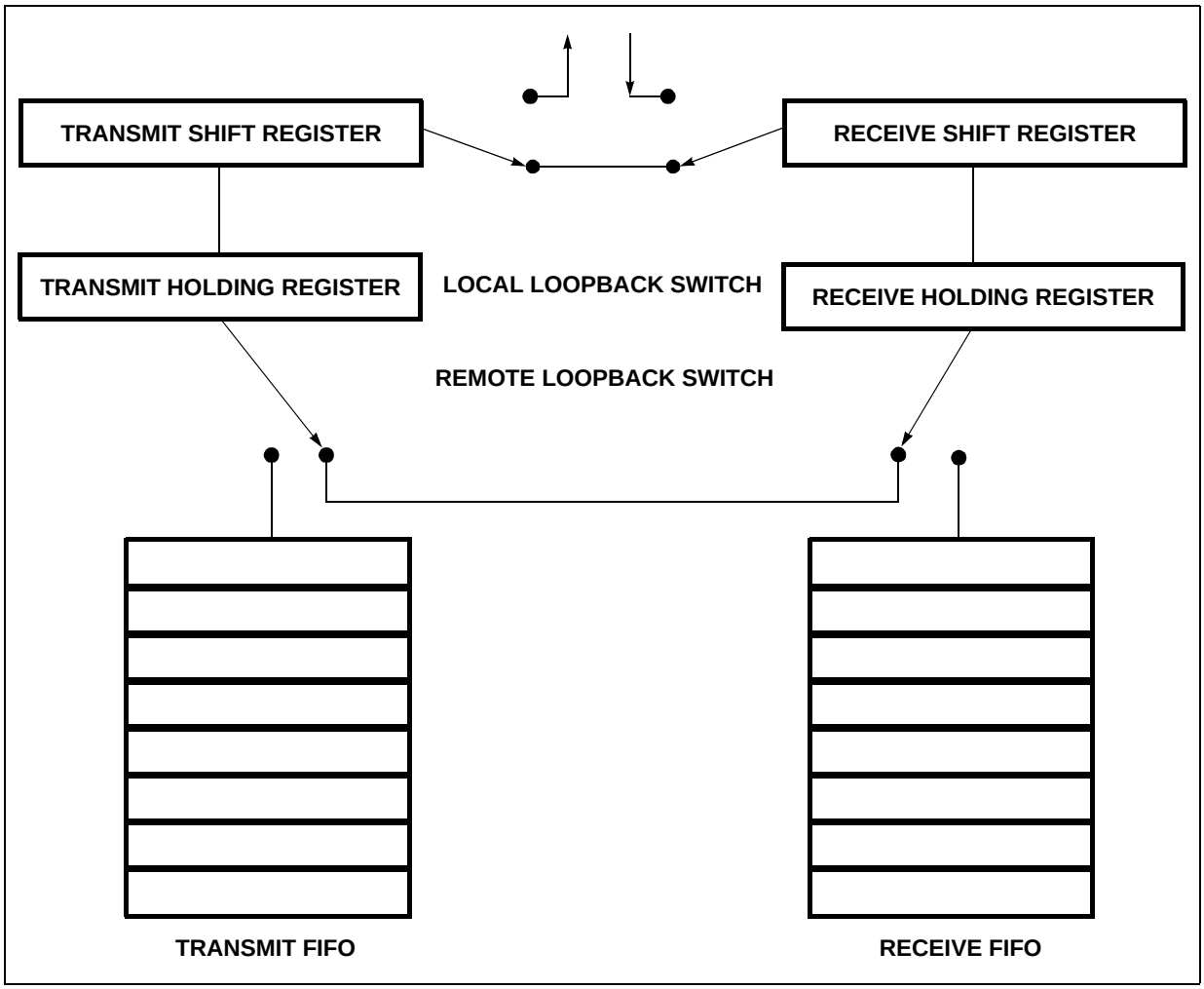
Remote Loopback Mode

Remote Loopback mode is provided to support testing of devices connected to the serial lines. Remote Loopback is invoked by setting the RLM bit in the Channel Option register 2 (COR2). When in this mode, the CD1865 echoes the received data to the transmitter for transmission back to the sender. The received data is not passed on to the host.

When in Remote Loopback mode, the transmitter continues to run as defined by its own Baud Rate registers, not the values being used by the receiver. The CD1865 receives a complete character, strips off Start, Stop, and Parity bits, and then re-transmits it with Parity, Length, and Stop bit Output options as defined in COR1. Thus, it is possible to change baud rate. However, this can result in receiver overflow. In general, when programming for Remote Loopback Operation, the Transmit bit rate should be as fast or slightly faster than the expected receive rate to avoid possible overrun and loss of data. The number of Stop bits should be set to a one, rather than one-and-a-half or two, if the application permits it. This ensures that the effective transmit rate is faster than the receive rate.

As shown in Figure 27, Remote Loopback is done at the character level and not the bit level. The Receive and Transmit FIFOs are not used in Remote Loopback. Characters are transferred directly from the Receive Holding register to the Transmit Holding register. For a diagnostic mode that tests the FIFOs, other logic is needed to be implemented by programming the host system to transfer received characters from the Receive FIFO to the Transmit FIFO. This permits full testing of FIFO thresholds, service request logic, special character operation, and so on.

Figure 27. Local and Remote Loopback Logic



8.0 Programming

8.1 Types of Registers

The CD1865 contains three types of registers:

- Global registers — registers not specific to a particular channel
- Indexed Indirect registers — special registers that are mapped to unique functions
- Channel registers — registers specific to each channel

Global Registers

Global registers contain information common to all channels. They are used primarily for passing vectors and setting-up service request handling.

Indexed Indirect Registers

Indexed Indirect registers are special registers that either point to the FIFOs or signal the end-of-service request processing. The Indexed Indirect registers are used primarily to transfer data to and from the serial channel FIFOs. Such transfers can be done only during a service request. When service requests are being serviced by the host, a context-switching technique is used by the CD1865 that reduces the number of cycles needed by the host to transfer data to and from the CD1865. The CD1865 makes available to the host all the registers pertaining to the channel requesting service by mapping them through to the Indexed Indirect register addresses. This removes the burden, of keeping different addresses according to which channel is being accessed, from the host.

FIFO information is channeled through either the Receive Data register, the Receive Character Status register, or the Transmit Data register of the Indexed Indirect register set. Use of the Indexed Indirect registers is valid only during appropriate service requests; the Transmit Data register can be accessed during Transmit Service Requests, but not during Receive or Modem Service Requests. The Channel Access register's (CAR) content is left unchanged from the value last set by the user, but it is not used in a service request context. The CAR should not be modified during a service request. During a service request, only access the channel that has caused the service request to be issued (as defined by the Global Interrupting Channel register).

Channel Registers

Channel registers are used to store parameters specific to each channel, such as bit rates, special character processing, and modem options. When not actively involved in a service request, each channel can be accessed at any time, independently of the other channels. Channel registers can be accessed by first writing the number of the channel to be accessed into the Channel Access register. The channel number in the CAR is used by the CD1865 as part of the Channel register Address.

Individual CD1865 registers are addressed by a seven-bit address contained in Address Bus bits A6-A0. Address bit A6 set to a '1' selects the Global registers, and when set to a '0' selects the Channel registers. When the CD1865 is not in a service request context, the active channel is defined in the CAR. The contents of the CAR then become part of the Address Field (along with A0–A5) needed to access the Channel register file.

Off-Limit Registers

The CD1865 communicates to the host by shared access to its on-device RAM. Of the 128-byte locations in the CD1865 address range, only 41 locations are defined as registers available to the host. The rest are used by the CD1865 for internal variable storage. Users should not access these registers since it can cause the CD1865 to malfunction.

8.2 Access Duty Cycle

The host access to the CD1865 appears to be a simple static read or write cycle, but the actual access occurs by arbitrating for the local (on-device) bus and ‘stealing’ one-bus cycle. This is completely hidden from the user in normal circumstances, and successive accesses to the CD1865 may be done ‘back-to-back’ with no delay. However, if the host were to repetitively read from (or write to) the CD1865 as fast as possible over many cycles, enough CD1865 internal bus cycles would be ‘stolen’ that the CD1865 processor might not be able to keep pace with its processing. This situation could only occur if the host was continuously testing a bit while waiting for it to change state. If there is a requirement to do something similar, insert a delay in the host code so that the net-duty cycle of accesses is less than ten percent. This limitation applies only when the CD1865 is sending and receiving data on one or more channels. When initializing or re-configuring a channel, these registers can be written to at a fast pace.

8.3 Accessing FIFOs Versus Other Registers

The FIFO storage array is under the control of the CD1865 at all times. This is necessary to ensure that the FIFO is available for the CD1865 processor to access whenever needed. During normal operation, the CD1865 processor sets the FIFO pointers to the value required to transfer data, regardless of the value placed in the Channel Access register (CAR) by the user. Therefore, the user cannot access the FIFOs in this manner.

FIFOs can only be accessed in the context of an active Service Request. At this time only the CD1865 processor causes the FIFO pointers to be set to the appropriate value for the channel being serviced. FIFOs are then accessed by the Indirect Indexed registers.

8.4 Initialization

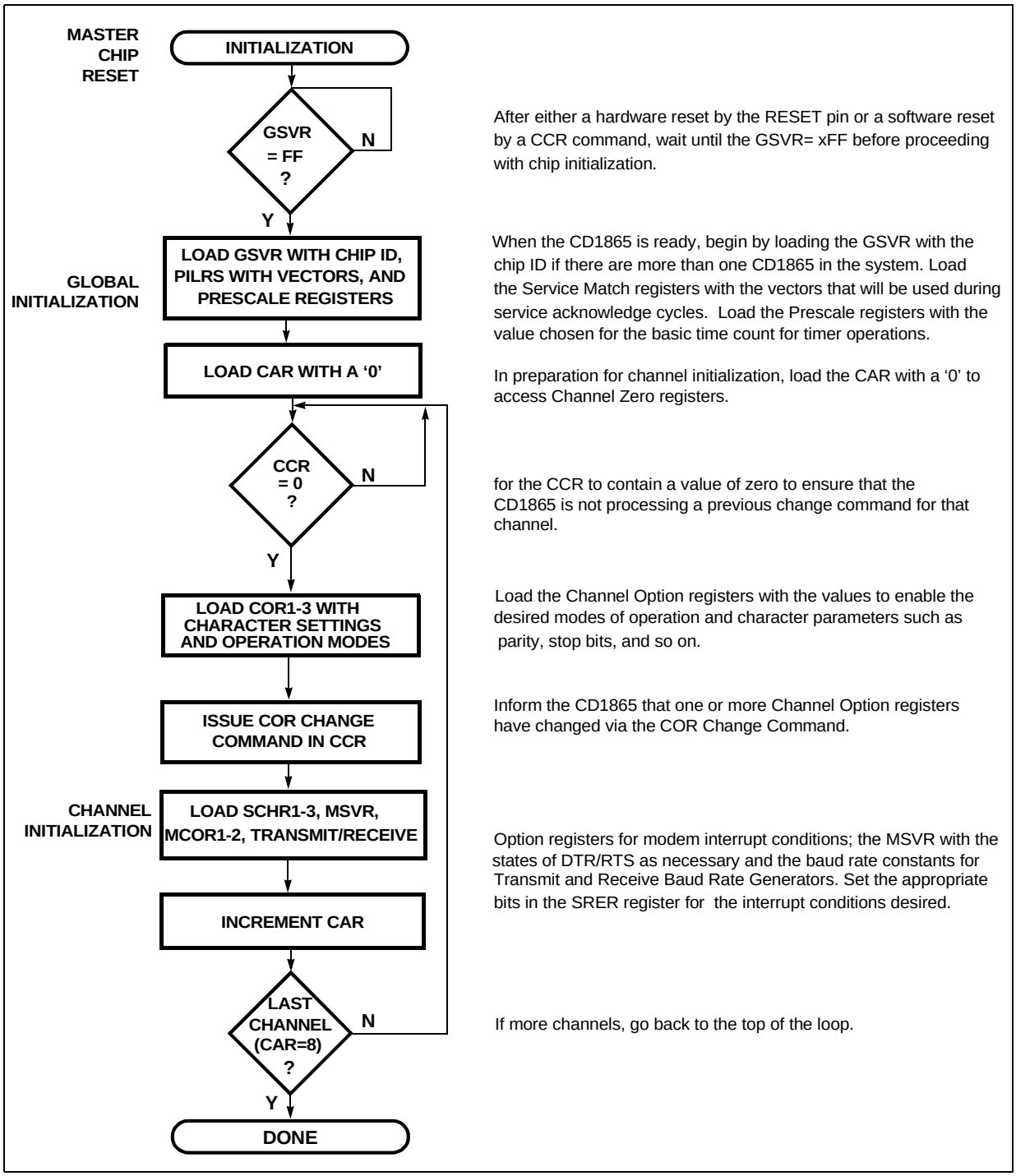
The CD1865 initialization begins with a mandatory hardware reset applied through the active-low RESET* Input. The system Clock (CLK) Input must be active during the hardware reset, and the reset duration must be at least five clock periods. It is not necessary to synchronize RESET* Input with CLK. Refer to Figure 28.

Immediately following the hardware reset, the CD1865 goes through a firmware initialization, reaching an Idle mode within 500 µs. This can be verified by the host by reading the Global Service Vector register and finding its contents to be FF Hex. Upon internal reset completion, the user can then configure the CD1865 for the required channel functions.

A software reset can be performed by setting certain bits in the Channel Command register (CCR). Setting bits 7 and 0 to a ‘1’ resets all channels. This is done by forcing the CD1865 processor to jump to the same power-up sequence that it uses upon hardware reset. Whether the reset is caused

by hardware or software, the CD1865 does not initialize every register and RAM location to a defined value. The only sure state is that all channels are inactive, no service requests are pending, and the Global Interrupt Vector register is FF Hex.

Figure 28. Initialization



8.5 Global Register Initialization

The user must initialize the CD1865 by programming the following Global registers before starting normal operations on the ports — Prescaler Period registers, the Global Interrupt Vector register, and the three Service Match registers ().

8.6 Service Request Initialization

To prepare the CD1865 for service requests the following registers must be initialized:

- Global register (GVR)
- three Service Match registers ()
- Global Channel registers (GCR)

The Global Vector register consists of five bits of user-supplied information, and three bits of CD1865–supplied service request group information. This concatenated vector supplied by the CD1865 during a service-request-acknowledgment cycle directs the host to the proper service request subroutine. The host writes the five MSBs into the GVR during initialization. These five bits can be either a device ID number or an appropriate code for handling service request. In multiple-cascaded CD1865 applications, these five bits must have a unique value for each CD1865 to identify which CD1865 is responding to a service request cycle.

Three registers in the Global register set — Modem Service Match register (), Transmit Service Match register (), and Receive Service Match register () store the service request values for the three types of service requests. These levels are used to match with the value that appears on the address bus during a service-request-acknowledgment cycle. Since these levels are system dependent, the user must initialize these registers with the proper values.

The following three registers provide the channel number of the channel requesting service — GCR1, GCR2, and GCR3. Reading any of these registers causes the CD1865 to ‘mask-in’ three bits specifying the channel number of the currently active channel. Normally these registers are read by the host when it is handling a service request. In this case, the three bits are the number of the channel requesting service. If any of the three GCR registers are read when the CD1865 is not in a service request context, the three bits are the current value in the CAR.

Bits 4:2 are masked into the contents of these registers by the CD1865 when it is read by the host. The actual contents of the register are not modified.

These three registers are provided as a convenience to the user. In most applications the user only uses one of these locations and sets the register to an arbitrary value. However, in some cases it may be useful to be able to record information about the state of the CD1865 (or the software driving it) that is associated with each of the three service request types. In this case, the user may store required information in the unused bits. When entering a service routine, the software can check these bits (a ‘sub-vector’) to read recorded states.

8.7 Prescaler

The Prescaler Period register (PPR) determines the fundamental ‘tick’ rate for all CD1865 on-device timers, the Receiver Data Time-out and Transmitter Real-time Delay Timers. The PPR counts Clock (CLK) periods, and the minimum PPR value used must guarantee a ‘tick’ length of

not less than 1.0 milliseconds. As shown in the Internal Operation Flow Chart, Figure 4 on page 25, processing timer events is in the outer (lowest priority) loop of the CD1865 firmware. A timer tick that is too short may result in two ticks occurring within one pass through the outer loop; this would result in missing one tick. This is not fatal, but it would result in inaccurate timings.

8.8 Channel Initialization and Changes

Prior to enabling the individual channels, program the Channel registers with required channel options and parameters such as character lengths, parity type, Receive FIFO thresholds, modem signal detection levels, bit rates, and so on. When ready to begin, enable service requests.

Channel initialization is accomplished by first writing to the CAR register with the number of the channel to be programmed. This channel number automatically becomes part of the address for subsequent channel register programming. The host can use the same set of register addresses for all channels, thus eliminating the need to calculate addresses.

Certain channel options are controlled by the three Channel Option registers. All changes to the Channel Option registers must be accompanied by setting the appropriate Channel Option register ‘changed’ bits in the Channel Command register (CCR). The CD1865 processor regularly samples the CCR for any value that is not a ‘0’. If the CCR is not a ‘0’, the CD1865 decodes the command or commands, acts on them, and clears the CCR to signify completion of the commands. New commands must not be issued until any existing commands have been completed.

8.9 Transmitting Data

When transmitting data, a service request is received when the Transmit FIFO is empty. The number of the channel requesting service (for example, the one with the empty FIFO) is available from the GCR. If there is more data to be sent, transfer up to 8 bytes to the FIFO. If no data is available, disable the channel. The easiest way to accomplish this is by clearing the appropriate bit in the Enable register (). When new data is available, re-enable the channel by the , and a new service request for transmit data is received. At that time, transfer the data to the FIFO. Channels can be enabled or disabled by giving enable and disable commands by the Channel Command register (CCR), but it is a slower process.

In some cases, it is necessary to know when a channel has sent the last bit of the last character rather than an empty FIFO. One example would be when changing bit rates. Two bits in the Enable register (), TxMpty and TxRdy, control the exact conditions for generating a service request. TxRdy indicates when the FIFO is empty, and TxMpty indicates when the last bit has been sent. It is acceptable to have both bits set but proper operation is achieved by switching from the FIFO empty status to the transmitter empty status when it is necessary to know that all data has been completely sent. If they are set, the FIFO Empty Service Request always occurs first. If there is no more data to be sent, the Transmitter Empty Service Request is received later, but in the mean time, FIFO empty requests may also be received. Once the last bit of the last character has been sent, a channel can be reconfigured.

8.10 Receiving Data

When receiving data, a service request is sent (for Good Data) when either the number of received bytes meets the threshold level, the Receive Time-out expires, or there is Good Data followed by a Receive Exception Condition (the CD1865 must transfer all the Good Data before giving the Exception). In all cases, the service-request routine reads the channel number requesting service (from GCR) and the number of bytes available (which can be more, the same, or less than the number set as the threshold) from the Receive Data Count register (RDCR), and proceeds to transfer that many bytes, if possible.

It is not necessary to transfer as many bytes as are available or any bytes at all. If the host's buffer is nearly or completely full, the host can accept only those bytes it has room for, disable Receive Service Requests, exit the Service Request Routine, process the buffer, enable Receive Service Requests, and wait for the next service request. If no bytes are transferred during a Receive Service Request, and Receive Service Requests are still enabled, the CD1865 immediately re-requests service because the internal conditions that caused the request to be issued are still true. The host may either disable service requests or suspend host service request processing; however, both of these options should be implemented carefully as suspending service requests may result in an overflow condition if the suspension lasts too long.

8.11 Programming Examples

When writing programs for the CD1865 evaluation board, a few guidelines should be followed to keep the programs from getting lost or error conditions to be encountered. This section discusses some programming errors and ways to avoid them.

8.11.1 Programming the Service Match Registers

One common programming error is made when using the CD1865 in the area of Service Match registers (SMR). The value placed in these three registers during chip initialization must **exactly** match the value that is present on the address inputs A0–A6 during the service acknowledge cycle. (When the ACKIN* control signal is activated.) If this condition is not met, the CD1865 does not respond with a DTACK* to terminate the bus cycle. This causes the system to hang.

8.11.2 CD1865 Initialization

Initializing the CD1865 is simple and quite straight forward. This section presents some guidelines for the sequence to write to the various registers to correctly complete the initialization process. Refer to Section 8.4 on page 84 for a flow-chart style description of the process.

The first step in the initialization process is to issue a master reset command to the CD1865 internal logic. This can be done in one of the two ways: throughout the use of the RESET* control signal at the hardware level, or via the chip reset command at the software level. The software reset command is issued by placing a value of x'81 in the CCR register. Internally the chip reset command does the same thing as activating the RESET* control input. The internal micro-code enters the exact same routines to setup the chip for operation. When the reset command has been issued, the program must wait until the GSVR has a value of x'FF. Until this value is placed in the GSVR by the micro-code, the CD1865 initialization procedures are not complete.

The next function to be performed is global initialization. These steps set up the chip ID for the CD1865, the Service Match registers (SMR) for the three service request levels and the Prescaler Period register. The SMR should be programmed as discussed previously (Section 8.1). The Prescaler Period register value (high and low) sets the basic time scale for internal timer operation, such as the receiver time-out period. A value must be chosen that yields a timer period of no less than 0.1 ms.

Following global initialization, each channel must be programmed for the desired mode of operation, including the transmit and receive baud rate divider constants, the individual character settings such as parity, bits per character, and number of Stop bits. Receive FIFO threshold levels, special character values, modem output signal levels and interrupt conditions. Before beginning the process of channel initialization, the CAR register must be loaded with the number of the register to be worked on. One important point to remember is that before placing a new value in any of the COR registers or issuing the COR change command, the CCR must be checked to be sure that it has a value of zero. If it is not zero, then the CD1865 may be processing a previous CCR command and the CCR and the CORs must be changed.

If the program is ready at this point to respond to interrupts then the appropriate interrupt condition bits for transmit and receive can be set in the SRER, and the transmitter and receiver can be enabled by command to the CCR.

The following sections explain the initialization sequence.

Global Initialization

Use Set_Byte to write to the register, and Read_Byte to read the register content. For details on the two functions, refer to the following basic I/O operations.

```
    Set_Byte(GSVR, 0x00);            // Clear GSVR for chip reset
    Wait_CCR();                      // Confirm CCR is clear
    Set_Byte(CCR, 0x81);             // Reset all Command
    Wait_GSVR();                     // wait to be FF
    Set_Byte(PPRH, 0x80 );           // Set up Timer Prescaler (High)
    Set_Byte(PPRL, 0xe8 );           // Set up Timer Prescaler (Low)
/*-------------------------------------------------------------------------
/* set up the Service Match Register according to the decoding ACKIN value
/* In this case, we decoded to be 0x8x.
/*-------------------------------------------------------------------------*/
    Set_Byte(RX_SMR, 0x8a );          // Set Service Match reg.
    Set_Byte(TX_SMR, 0x85);
    Set_Byte(MDM_SMR, 0x81);
                                     // comment out the following line if is using
the hardware acknowlegement.
    //Set_Byte(SRCR, 0x40);          // Set up the software interrupt acknowledge
```

Channel Initialization

```
    Set_Byte(CAR, chan);             // Setup channel access register to be the
                                     // Specified channel number.
    Set_Byte(COR1, 0x03);            // No parity, 8-bit char, 1-stop bit
    Set_Byte(COR2, 0x00);            // Disable all COR2 functions
    Set_Byte(COR3,0x35);             // Special char detection, FCT
    Wait_CCR();
    Set_Byte(CCR, 0x4e);

    Set_Byte(SCHR1, 0x11);            // XON defined (cntl-Q 0x11)
    Set_Byte(SCHR2, 0x13);            // XOFF defined (cntl-S 0x13)
    Set_Byte(SCHR3, 0x11);
```

```
    Set_Byte(SCHR4, 0x13);
    Set_Byte(RTPR, 0x05);            // Set timeout value

    Set_Byte(TBPRH,0x00);            // Set Tx baud rate 115200 (devisor 0x12) at
33 MHz
    Set_Byte(TBPRL,0x12);
    Set_Byte(RBPRH,0x00);            // Set Rx baud rate 115200 (devisor 0x12) at
33 MHz
    Set_Byte(RBPRL,0x12);
```

8.11.3 Basic I/O Operations

All the example routines for accessing and programming the board resources are written in Borland® C++, but can be easily ported to other languages. Specific device programming is not included in this document; refer to the device data book for general programming information.

Read_Byte routine is used to read a register within the CD1865. The sequence is shown below:

```
    // Read the content of assigned register
    unsigned char Read_Byte( unsigned char addr )
    {
    return ( inportb( BASE_ADDR+addr ) );
    }
```

Set_Byte routine is used to write to the register and is a similar operation as the Read_Byte.

```
    // Set the content of assigned register
    void Set_Byte( unsigned char addr, unsigned char data)
    {
    outportb(BASE_ADDR+addr, data);
    }
```

8.11.4 Interrupt Response Operations

The CD1865 evaluation board generates a single interrupt to the ISA bus in response to an interrupt from any of the three possible sources within the CD1865 device. The interrupt sources are from the receive system, the transmit system, or the modem functions from any of the available channels. This single interrupt can be user selected to any of the three ISA -bus interrupt sources as described in the configuration section. Note that due to all the interrupt signals being OR'ed together, the PC motherboard 8259A PIC must be programmed in level sensitive rather than edge-sensitive mode.

Determining the Interrupt Source

The host's response to an interrupt from the board is to call an interrupt service routine that determines the type of interrupt pending and services the request. The example below shows one way to perform the interrupt determination using the Interrupt Status register.

```
    while (  (int_status = Read_Byte( SRSR )) & 0x15  ){

                                     //   Case of Receiving interrupt
                                     if ( int_status & 0x10 )
                                     Service_Rx(chan);



                                     // Case of transmitting interrupt
                                     if ( int_status & 0x04 )
                                     Service_Tx(chan);
```

```
                                    if (int_status & 0x02)
                                    Service_Mdm();

     } //while

     outportb(S8259, EOI);          /* End of Interrupt */
     outportb(S8259, RDISTAT);      /* Next access read the IS Reg. */
     if ( !inportb(S8259) )         /* while the slave is not serving any int. */
    outportb(M8259, EOI);           /* issue End of Int. (EOI) to master */
```

Once the interrupt source has been determined, the request must be serviced by issuing an IACKIN signal to the device with the preprogrammed PILR match value supplied as the address. A receive interrupt acknowledge cycle might be written as shown. Immediately following the write to the EOIR register to terminate the current interrupt context, the 8259A must be informed that the service is over. This is done by the simple procedure shown at the end of the interrupt source determination routine above.

Receive Interrupt Service

```
Service_Rx(unsigned char chan)
{
unsigned charchannel, vector, RxCount;
int            i;

    vector= Read_Byte(0x8a);          // perform hardware acknowledge
    channel = Read_Byte(GSCR1) >>     2;
    RxCount=Read_Byte(RDCR);          // RDCR contains the number of byte to be
transfered.
    if ((RxCount>0)&&(RxCount<=8))
        {
              if  ((exception_data = Read_Byte(RCSR)) != 0)
                                                  // Receive Exception: in this
                                                  // Example, we disable receive
                                                  // Operation if detected a
receive
                                                  // Exception.
              {
                    Set_Byte(CAR, chan);
                    Set_Byte(SRER, 0x00);         // Disable receive
              }

else
                                                  // Normal Receive Operation:
no
                                                  // Receive exception.
              {
                    if (channel==chan){           // Correct Receiving Channel
                    for (i=RxCount; i>0; i--){
                         rx_str[rx_ptr]=Read_Byte(RDR);
                         rx_ptr++;
                    }//for
                    }//if
                    else                          // Incorrect Receiving Channel
                         Rx_chan_err = 1;
              } // else
        }  //if

Set_Byte(EOSRR, 0x00);                  // Set Transmit End of Int Reg. The
Transmit End of                               // Interrupt Register must be
written to by the
                                        // Corresponding host  interrupt service
routine to
```

```
                                        // Signal to the CD1865 that the current
interrupt
                                        // Service is concluded.
        return;

}
```

Transmit Interrupt Service

```
Service_Tx(unsigned char chan)
{
unsigned char      channel, vector,c;
int i;

        vector= Read_Byte(0x85);          // Perform hardware acknowledge
        channel = Read_Byte(GSCR1) >> 2;

        if (channel==chan){               // Make sure correct channel
              for (i=1; i<= 8 && !quit_tx ; i++){

                    Set_Byte(TDR, txm_str[tx_ptr[chan]++] );
                    if (tx_ptr[chan] >= strlen(txm_str) ){
                          tx_ptr[chan] = 0; // Reset the pointer back to the
                          quit_tx = 1;
                    }
              }//for
        }//if
        else
              Tx_chan_err = 1;       // Wrong transmitting channel.

        Set_Byte(EOSRR, 0x00);       // Set Transmit End of Int Reg.
                                     // The Transmit End of Interrupt Register must
                                     // Be written to by the corresponding host
                                     // Interrupt service routine to signal to the
                                     // CD1865 that the current interrupt service
                                     // Is concluded.

        return;
}
```

Modem Interrupt Service

```
Service_Mdm()
{

        unsigned char channel, vector;

        vector = Read_Byte(MRAR);         // Software acknowledge using MRAR
        //vector = Read_Byte(0x81);       // Comment out the previous line, if using
// hardware acknowledge.

        channel = Read_Byte(GSCR1) >> 2;
        switch(Read_Byte(MCR)&0xe0)
        {
              case 32:                          // case of CTS change interrupt
              {
                    printf(" CTR has a changed state. \n");
                    break;

              }
              case 64:                          // case of CD change interrupt
              {
                    printf(" CD has a changed state.\n");
                    break;
```

```
                }
                case 128:                       // case of DSR change interrupt
                {
                      printf(" DSR has a changed state.\n");
                      break;
                }

                default:
                {
                      printf(" Invalid Modem interrupt detected.......\n");
                      break;
                }

        }

        Set_Byte(MCR, 0x00);              // Clear the Modem Change register after
                                          // service the modem request.
        Set_Byte(EOSRR, 0x00);            // Clear the EOSRR register at the end
                                          // modem service routine.
        return;
}
```

8.11.5 Polled Mode Operation

The Polled-mode operation can be used with any type of host CPU, or it can be used in combination with interrupts to provide a Mixed-mode system optimized for a particular operation. For details refer to Section 5.5 on page 35.

In the Polled-mode operation, the Service Match registers need to be setup first (TSMR, RSMR, MSMR) in the channel initialization routine. Once an interrupt is detected, it is acknowledged by reading the corresponding register depending on the type of interrupt.

```
while ( !((int_status = Read_Byte( SRSR )) & 0x15 ) )
        {
                  // waiting for interrupt.
                  break;
        }

// Case of Receiving interrupt
if ( int_status & 0x10 )
{
              Read_Byte(RSMR)
              Service_Rx();
}
// Case of transmitting interrupt
if ( int_status & 0x04 )
{
              Read_Byte(TSMR)
              Service_Tx();
}
// case of modem interrupt
if (int_status & 0x02)
{
              Read_Byte(MSMR)
              Service_Mdm();
}
```

9.0 Detailed Register Descriptions

9.1 Register Map Quick Reference

Name	Description	Access	Binary Address	Default Value	Hex Address (8 bit)[1]	Hex Address (Intel®)[2]	Hex Address (Motorola®)[3]	Page
Global Registers								
GFRCR	Global Firmware Revision Code Register	R/W	110 1011	84	$6B[4]	$D6	$D7	98
SRCR	Service Request Configuration Register	R/W	110 0110	0	$66	$CC	$CD	98
PPRH	Prescaler Period Register High	R/W	111 0000	FF	$70	$E0	$E1	100
PPRL	Prescaler Period Register Low	R/W	111 000	FF	$71	$E2	$E3	100
MSMR	Modem Service Match Register	R/W	110 0001	0	$61	$C2	$C3	100
TSMR	Transmit Service Match Register	R/W	110 0010	0	$62	$C4	$C5	101
SSVR	Receive Service Match Register	R/W	110 0011	0	$63	$C6	$C7	101
	Global Vector Register	R/W	100 0000	FF	$40	$80	$81	102
SRSR	Service Request Status Register	R	110 0101	0	$65	$CA	$CB	103
MRAR	Modem Request Acknowledge Register	R	111 0101	80	$75	$EA	$EB	105
TRAR	Transmit Request Acknowledge Register	R	111 0110	80	$76	$EC	$ED	105
RRAR	Receive Request Acknowledge Register	R	111 0111	80	$77	$EE	$EF	105
GSCR1	Global Channel Register 1	R/W	100 0001	0	$41	$82	$83	106
GSCR2	Global Channel Register 2	R/W	100 0010	0	$42	$84	$85	106
GSCR3	Global Channel Register 3	R/W	100 0011	0	$43	$86	$87	106
CAR	Channel Access Register	R/W	110 0100	0	$64	$C8	$C9	107
Indexed Indirect Registers								
RDCR	Receive Data Count Register	R	000 0111	0	$07	$0E	$0F	108
RDR	Receiver Data Register	R	111 1000	0	$78	$F0	$F1	109
RCSR	Receiver Character Status Register	R	111 1010	0	$7A	$F4	$F5	110
TDR	Transmit Data Register	W	111 1011	0	$7B	$F6	$F7	111
EOSSR	End Of Register	W	111 1111	0	$7F	$FE	$FF	111

NOTES:
1. Hex address for 8-bit processor.
2. Address for Intel-style processor, see the following description.
3. Address for Motorola-style processor, see the following description.
4. $ denotes address value.

Name	Description	Access	Binary Address	Default Value	Hex Address (8 bit)[1]	Hex Address (Intel®)[2]	Hex Address (Motorola®)[3]	Page
Channel Registers								
SRER	Enable Register	R/W	000 0010	0	$02	$04	$05	112
CCR	Channel Command Register	R/W	000 0001	0	$01	$02	$03	112
COR1	Channel Option Register 1	R/W	000 0011	0	$03	$06	$07	116
COR2	Channel Option Register 2	R/W	000 0100	0	$04	$08	$09	116
COR3	Channel Option Register 3	R/W	000 0101	0	$05	$0A	$0B	117
CCSR	Channel Control Status Register	R	000 0110	0	$06	$0C	$0D	118
RBR	Receiver Bit Register	R	011 0011	21	$33	$66	$67	119
RTPR	Receive Time-Out Period Register	R/W	001 1000	5	$18	$30	$31	120
RBPRH	Receive Bit Rate Period Register High	R/W	011 0001	0	$31	$62	$63	120
RBPRL	Receive Bit Rate Period Register Low	R/W	011 0010	0	$32	$64	$65	120
TBPRH	Transmit Bit Rate Period Register High	R/W	011 1001	0	$39	$72	$73	121
TBPRL	Transmit Bit Rate Period Register Low	R/W	011 1010	0	$3A	$74	$75	121
SCHR1	Special Character Register 1	R/W	000 1001	0	$09	$12	$13	121
SCHR2	Special Character Register 2	R/W	000 1010	0	$0A	$14	$15	122
SCHR3	Special Character Register 3	R/W	000 1011	0	$0B	$16	$17	122
SCHR4	Special Character Register 4	R/W	000 1100	0	$0C	$18	$19	123
MCR	Modem Change Register	R/W	001 0010	0	$12	$24	$25	123
MCOR1	Modem Change Option Register 1	R/W	001 0000	0	$10	$20	$21	124
MCOR2	Modem Change Option Register 2	R/W	001 0001	0	$11	$22	$23	125
MSVR	Modem Signal Value Register	R/W	010 1000	0	$28	$50	$51	125
MSVRS	Modem Signal Value Request To Send	W	010 1000	0	$29	$52	$53	126
MSVDR	Modem Signal Value Data Terminal Ready	W	010 1010	0	$2A	$54	$55	126

NOTES:
1. Hex address for 8-bit processor.
2. Address for Intel-style processor, see the following description.
3. Address for Motorola-style processor, see the following description.
4. $ denotes address value.

Even though not all of the CD1865 registers are intended to be read/write, there is no hardware mechanism to prevent the user from writing to them. The registers should, in some cases, not be written to by the host. See the individual register descriptions for details.

In the register map, the binary addresses are shown relative to the CD1865 address lines. In 16- and 32-bit systems, it is a common practice to connect 8-bit peripherals to only 1-byte lane. In 16-bit systems, the CD1865 appears at every other address, that is, A0 in the CD1865 is connected to A1 in the host. In 32-bit systems, the CD1865 appears at every fourth address, that is, A0 in the CD1865 is connected to A2 in the host. In both of these cases, the addresses used by a programmer are different than what is shown.

For instance, in a 16-bit Motorola 68000-based system (or other ‘big-endian’ processors), the CD1865 is placed on data lines D0–D7 that are at odd addresses in the Motorola manner of addressing. The A0 in the CD1865 is connected to A1 of the 68000. Thus, the CD1865 address $40 becomes $81 to a programmer. It is ‘left-shifted’ one bit, and A0 must be ‘1’ for low-byte (D0–D7) accesses.

In a 16-bit Intel system (or other ‘little-endian’ processors), the CD1865 is again placed on data lines D0–D7, but these are at even addresses. The A0 in the CD1865 is connected to the A1 in the host, but the host’s A0 must be a ‘0’ to access data lines D0–D7.

Many 32-bit processors have internal logic to ‘steer’ the data to the correct pins regardless of address value. However, if the processor employed does not, a scheme similar to the one described for 16-bit machines can be used, except that the CD1865 addresses are shifted 2 bits instead of one.

Table 9. Register Summary (Sheet 1 of 2)

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Global Registers								
GFRCR	Firmware Revision Code							
SRCR	PkgTyp	RegAckEn	DaisyEn	GlobPri	UnFair	Reserved	AutoPri	PriSel
PPRH	Binary Value							
PPRL	Binary Value							
MSMR	Binary Value							
TSMR	Binary Value							
RSMR	Binary Value							
GSVR	User Defined	User Defined	User Defined	User Defined	User Defined	IT2	IT1	IT0
SRSR	ILV[1]	ILV[0]	RREQext	RREQint	TREQext	TREQint	MREQext	MREQint
MRAR	Modified Interrupt Vector Provided On Read							
TRAR	Modified Interrupt Vector Provided On Read							
RRAR	Modified Interrupt Vector Provided On Read							
GSCR1	User Defined	User Defined	User Defined	C2	C1	C0	User Defined	User Defined
GSCR2	User Defined	User Defined	User Defined	C2	C1	C0	User Defined	User Defined
GSCR3	User Defined	User Defined	User Defined	C2	C1	C0	User Defined	User Defined
CAR	Reserved	Reserved	Reserved	Reserved	A7(0)	C2	C1	C0

Table 9. Register Summary (Sheet 2 of 2)

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Indexed Indirect Registers								
RDCR	0	0	0	0	CT3	CT2	CT1	CT0
RDR	D7	D6	D5	D4	D3	D2	D1	D0
RCSR	Time-Out	SC Det2	SC Det1	SC Det0	Break	PE	FE	OE
TDR	D7	D6	D5	D4	D3	D2	D1	D0
EOSRR	Irrelevant Value							
Channel Registers								
SRER	DSR	CD	CTS	RxD	RxSC	TxRdy	TxMpty	NNDT
CCR	RESET CHAN	COR CHNG	SEND SP CH	CHAN CTL	D3	D2	D1	D0
COR1	Parity	PArM1	ParM0	Ignore	Stop 1	Stop 0	CHL 1	CHL 0
COR2	IXM	TxIBE	ETC	LLM	RLM	RtsAO	CtsAE	DsrAE
COR3	Xon CH	Xoff CH	FCT	SCDE	RxTH3	RxTH2	RxTH1	RxTH0
CCSR	RxEN	RxFloff	RxFlon	Not Used	TxEn	TxFloff	TxFlon	Not Used
RBR	Reserved	RxD	Start Hunt	Reserved	Reserved	Reserved	Reserved	Reserved
RTPR	Receiver Data Time Out Period							
RBPRH	Receive Bit Rate Divisor Byte High							
RBPRL	Receive Bit Rate Divisor Byte Low							
TBPRH	Transmit Bit Rate Divisor Byte High							
TBPRL	Transmit Bit Rate Divisor Byte Low							
SCHR1	Special Character 1							
SCHR2	Special Character 2							
SCHR3	Special Character 3							
SCHR4	Special Character 4							
MCR	DSRchg	Cdchg	CTSchg	0	0	0	0	0
MCOR1	DSRzd	Cdzd	CTSzd	0	DTRth3	DTRth2	DTRth1	DTRth0
MCOR2	DSRod	Cdod	CTSod	0	0	0	0	0
MSVR	DSR	CD	CTS	Not Used	Not Used	Not Used	DTR	RTS
MSVRTS	0	0	0	0	0	0	0	RTS
MSVDTR	0	0	0	0	0	0	DTR	0

9.2 Global Registers

Global registers provide a function common to all channels. There are two groups of Global registers: those that control the configuration of the CD1865 and those that control service requests/interrupts.

9.2.1 Miscellaneous Registers

9.2.1.1 Global Firmware Revision Code Register

Register Name: GFRCR							8-Bit Hex Address: $6B
Register Description: Global Firmware Revision Code Register							Intel Hex Address: $D6
Default Value: 84							Motorola Hex Address: $D7
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Firmware Revision Code							

This register is initialized by the firmware during the power-on reset initialization routine to contain the current firmware version code of the CD1865.

This register is a RAM location and may be modified by the user. The CD1865 sets it to the defined value only when a hardware or software reset is performed, and its contents are otherwise ignored. This value can be modified to indicate the configuration status of the CD1865, or to indicate any other requirement.

9.2.2 Configuration Registers

9.2.2.1 Service Request Configuration Register

Register Name: SRCR							8-Bit Hex Address: $66
Register Description: Service Request Configuration Register							Intel Hex Address: $CC
Default Value: 0							Motorola Hex Address: $CD
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
PkgTyp	RegAckEn	DaisyEn	GlobPri	UnFair	Reserved	AutoPri	PriSel

This register configures the CD1865 depending on the method chosen for handling service requests. In addition to the ‘traditional’ interrupt-based host interface, writing the appropriate bits in this register provides for software- rather than hardware-based service request acknowledgments, fixes service request priorities in either of two ways, and controls Fair Share Interrupt operation. This register preserves compatibility with existing CD1865 software. For this reason, this register defaults to all zeroes and must be enabled for each new feature as required.

RegAckEn and DaisyEn Bits are related to each other, and perform service-request acknowledgments by accessing registers within the CD1865 instead of asserting hardware signals.

Service requests are prioritized by four other bits. AutoPri enables the priority scheme; PriSel, GlobPri, and UnFair determine the specific priority to be used.

Bit	Description
Bit 7	**PkgTyp:** This read-only bit indicates the CD1865 package type. This bit always reads back as .
Bit 6	**RegAckEn:** Enables register-based service-request acknowledgments. If this bit is a ‘0’, register-based acknowledgments are not accepted. In this case, the results of a read of any of the service-acknowledgment registers are undefined. This is the default state of RegAckEn, and it ensures compatibility with earlier versions of the CD1865. When RegAckEn is enabled, register-based acknowledges allow the user’s software to acknowledge a service request by reading from a register rather than by driving the external ACKIN* signal. This is convenient in applications where interrupts are not supported or where polling is preferred. Setting this bit does not disable the function of the ACKIN* signal.
Bit 5	**DaisyEn:** Enables daisy-chaining of register-based service acknowledgments. When DaisyEn is a ‘1’, a CD1865 being addressed with a register-based service acknowledgment (a read occurs from a register-acknowledgment address) for which it has a pending request, places the contents of the Global Interrupt Vector register modified by the service type on the data bus. When DaisyEn is a ‘1’, a CD1865 being addressed with a register-based service acknowledgment, for which it does not have a pending service request, asserts ACKOUT* to pass the acknowledgment down the daisy chain. The next CD1865 in the chain monitors the acknowledgment as an ACKIN* acknowledgment. The Service Request Acknowledge register addresses must be placed in the corresponding Service Match registers (, , and) as part of the user setup for daisy-chaining of register-based service acknowledgments. If daisy-chaining of register-based service acknowledgments is not used, the Service Match registers may be programmed with any address codes that the user finds convenient for use with the ‘normal’ ACKIN* service-acknowledge mechanism. If DaisyEn is a ‘0’ and a CD1865 is addressed with a register-based service acknowledgment for which it does not have a pending service request, it responds by providing an interrupt vector with a modification code of ‘000’. The addressed CD1865 treats this as an interrupt acknowledge cycle, but with passing inhibited, it must ‘take’ the acknowledge with an ACK level of ‘00’ (none of the interrupt types). RegAckEn must be a ‘1’ to enable register-based service acknowledgments. DaisyEn has no effect on daisy-chain operation of the regular ACKIN*/ACKOUT* chain.
Bit 4	**GlobPri:** When AutoPri is used, if GlobPri is set to a ‘1’, the CD1865 prioritizes across multiple CD1865s sharing REQ () lines. If GlobPri is set to a ‘0’, the CD1865 accepts the acknowledge for the highest priority on-device interrupt. In both cases, automatic prioritizing is only done on type 1 (normally the modem signal change type) interrupt acknowledgments through the ACKIN* mechanism or the register-based acknowledge mechanism. When using GlobPri and AutoPri, it is possible to use the CD1865 with the three REQ lines wire-OR’ed together. In this configuration, with any interrupt request asserted, the global values of all requests appears asserted. GlobPri should be a ‘0’ to force prioritization among the interrupt sources on-device. When no on-device interrupts are pending, the acknowledgment is subject to daisy-chaining. See DaisyEn description.
Bit 3	**UnFair:** Fairness Override bit. If UnFair is a ‘0’, normal Fair Share Interrupt control is performed. If UnFair is a ‘1’, the fair bits are all forced to a ‘1’, disabling the Fair Share mechanism. This is useful when the AutoPriority Option is used, and the different REQ lines are wire-OR’ed together.
Bit 2	Reserved. Must be a ‘0’.
Bit 1	**AutoPri:** When set, indicates that the CD1865 should prioritize service requests in the manner selected by the PriSel bit. In conjunction with the GlobPri bit, either local (within the device) or global (across daisy-chained devices) prioritization is done. With AutoPri set, auto-prioritization is performed only when a type 1 (modem) interrupt acknowledgment is recognized. Acknowledgments of type 2 (transmit) and 3 (receive) interrupts continue to be unique and specific even with AutoPri set. This offers a form of local override to Auto-prioritization for Transmit or Receive Service Request when continuing a second-priority service routine. If not set, the user must indicate the service request being acknowledged by the choice of service request acknowledge register. AutoPri x GlobPri => look at REQext to prioritize globally. AutoPri x GlobPri* => look at REQ to prioritize locally.
Bit 0	**PriSel:** Prioritized interrupt order option. If AutoPri is set, PriSel selects the highest-priority service request. If PriSel is a ‘0’, receive requests have the highest priority. If PriSel is a ‘1’, transmit requests have the highest priority. Modem signal change request priority is fixed at the lowest priority.

9.2.2.2 Prescaler Period Registers (High/Low)

Register Name: PPRH							8-Bit Hex Address: $70
Register Description: Prescaler Period Register (High)							Intel Hex Address: $E0
Default Value: FF							Motorola Hex Address: $E1
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Binary Value							

Register Name: PPRL							8-Bit Hex Address: $71
Register Description: Prescaler Period Register (Low)							Intel Hex Address: $E2
Default Value: FF							Motorola Hex Address: $E3
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Binary Value							

These two registers provide the initialization value for the Timer Prescaler that is clocked by the system clock. This establishes the clock for the various on-device timers.

The value loaded into these registers must establish a clock period of at least 1.0 msec. For a clock speed of 33 MHz, the value must be 33,000 (decimal) or larger. The values in these registers are programmed to be FF (Hex) automatically upon a hardware reset.

9.2.2.3 Modem Service Match Register

Register Name:							8-Bit Hex Address: $61
Register Description: Modem Service Match Register							Intel Hex Address: $C2
Default Value: 0							Motorola Hex Address: $C3
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Binary Value							

This register must contain the value for Modem Signal Change Service Requests that are presented on the Address Bus A0-A6 by the host to indicate the type of service request being acknowledged when ACKIN* is asserted. This register, along with the other two Match registers, is compared to the value on the Address Bus during acknowledgment cycles so that the CD1865 can determine the service request being acknowledged by the host.

Bit 7 must be programmed to a ‘1’. The CD1865 compares all eight bits internally, but there are only seven address lines. Bits 6:0 of the register are compared to A6:A0 of the Address Bus. Bit 7 of the register is compared with a logic ‘1’.

Within any one CD1865, the three Match registers must have unique values. In multiple CD1865 designs where service acknowledgments are cascaded, all Match registers of the same type (for example, Modem) must have the same value.

In designs using register-based service acknowledgments (RRAR, TRAR, and MRAR), the addresses of these registers must be placed in the equivalent Match register so that contains $75.

9.2.2.4 Transmit Service Match Register

Register Name:							8-Bit Hex Address: $62
Register Description: Transmit Service Match Register							Intel Hex Address: $C4
Default Value: 0							Motorola Hex Address: $C5
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
1	Binary Value						

This register must contain the value for Transmit Data Service Requests that are presented on the Address Bus A0-A6 by the host to indicate the type of service request being acknowledged when ACKIN* is asserted. This register, along with the other two Match registers, is compared to the value on the Address Bus during acknowledgment cycles so that the CD1865 can determine the service request being acknowledged by the host.

Bit 7 must be programmed to a ‘1’. The CD1865 compares all eight bits internally, but there are only seven address lines. Bits 6:0 of the register are compared to A6:A0 of the Address Bus. Bit 7 of the register is compared with a logic ‘1’.

Within any one CD1865, the three Match registers must have unique values. In multiple-CD1865 designs where service acknowledgments are cascaded, all Match registers of the same type (for example, Transmit) must have the same value.

In designs using register-based service acknowledgments (RRAR, TRAR, and MRAR), the addresses of these registers must be placed in the equivalent Match register so that contains $76.

9.2.2.5 Receive Service Match Register

Register Name:							8-Bit Hex Address: $63
Register Description: Receive Service Match Register							Intel Hex Address: $C6
Default Value: 0							Motorola Hex Address: $C7
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
1	Binary Value						

This register must contain the value for Receive Data Service Requests that are presented on the Address Bus A0-A6 by the host to indicate the type of service request being acknowledged when ACKIN* is asserted. This register, along with the other two Match registers, is compared to the value on the Address Bus during acknowledgment cycles so that the CD1865 can determine the service request being acknowledged by the host.

Bit 7 must be programmed to a ‘1’. The CD1865 compares all eight bits internally, but there are only seven address lines. Bits 6:0 of the register are compared to A6:A0 of the Address Bus. Bit 7 of the register is compared with a logic ‘1’.

Within any one CD1865, the three Match registers must have unique values. In multiple CD1865 designs where service acknowledgments are cascaded, all Match registers of the same type (for example, Receive) must have the same value.

In designs using register-based service acknowledgments (RRAR, TRAR, and MRAR), the addresses of these registers must be placed in the equivalent Match register so that contains $77.

9.2.2.6 Global Vector Register

Register Name:						8-Bit Hex Address: $40	
Register Description: Global Service Vector Register						Intel Hex Address: $80	
Default Value: FF						Motorola Hex Address: $81	
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Binary Value					IT2	IT1	IT0

Bit	Description
Bits 7:3	These bits are user-defined. However, in a multiple-device design, these five bits must have a unique value in each CD1865 to identify which CD1865 is returning a vector during service acknowledgments. When writing to this register, write eight bits at once; the CD1865 modifies the low-three bits automatically. Note that if this register is read in a normal manner, the original eight bits are read and the modified bits from the last acknowledgment cycle is not preserved.
Bits 2:0	These three bits indicate the group/type of service request occurring. These bit are supplied by the CD1865 during an acknowledgment cycle.

IT2	IT1	IT0	Value	Group/Type
0	0	0	0	No Request Pending*
0	0	1	1	Modem Signal Change Service Request
0	1	0	2	Transmit Data Service Request
0	1	1	3	Receive Good Data Service Request
1	0	0	4	Reserved
1	0	1	5	Reserved
1	1	0	6	Reserved
1	1	1	7	Receive Exception Service Request

NOTE: * This code is returned by the CD1865 only when RegAckEn is set, and DaisyEn is not set. In this condition, the CD1865 must provide a vector when acknowledged. If the CD1865 receives an acknowledgment for which it does not have a request pending, it returns ‘000’.

9.2.3 Service Request/Interrupt Control Registers

9.2.3.1 Service Request Status Register

Register Name:							8-Bit Hex Address: $65
Register Description: Service Request Status Register							Intel Hex Address: $CA
Default Value: 0							Motorola Hex Address: $CB
Access: Read only							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
ILV[1]	ILV[0]	ext	int	ext	int	ext	int

The i-level bits, ILV[1] and ILV[0], are the current context code from the service request context stack. They are encoded as follows:

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>Bits 7:6</td><td>
<table>
<tr><th>ILV1</th><th>ILV0</th><th>Context</th></tr>
<tr><td>0</td><td>0</td><td>Not in a service request context</td></tr>
<tr><td>0</td><td>1</td><td>CD1865 is in a Receive Service Request context</td></tr>
<tr><td>1</td><td>1</td><td>CD1865 is in a Transmit Service Request context</td></tr>
<tr><td>1</td><td>0</td><td>CD1865 is in a Modem Service Request context</td></tr>
</table>

An accepted interrupt acknowledge cycle pushes a new context onto the stack.
NOTE: The Status bits are positive true, and the * Pins are negative true. The ‘...int’ (internal) values are local to the device being read, and the ‘...ext’ (external) values are the current external status on the pin, that is, the result of the wire-OR’ed function.
</td></tr>
<tr><td>Bits 5:4</td><td>
<table>
<tr><th>RREQext</th><th>RREQint</th><th>Context</th></tr>
<tr><td>0</td><td>0</td><td>No interrupts</td></tr>
<tr><td>0</td><td>1</td><td>Invalid state</td></tr>
<tr><td>1</td><td>1</td><td>The location device requests a receive interrupt.</td></tr>
<tr><td>1</td><td>0</td><td>External interrupt pending. The local device has no receive interrupts.</td></tr>
</table>
</td></tr>
<tr><td>Bits 3:2</td><td>
<table>
<tr><th>TREQext</th><th>TREQint</th><th>Context</th></tr>
<tr><td>0</td><td>0</td><td>No interrupts</td></tr>
<tr><td>0</td><td>1</td><td>Invalid state</td></tr>
<tr><td>1</td><td>1</td><td>The location device requests a transmit interrupt.</td></tr>
<tr><td>1</td><td>0</td><td>External interrupt pending. The local device has no transmit interrupts.</td></tr>
</table>
</td></tr>
<tr><td>Bits 1:0</td><td>
<table>
<tr><th>MREQext</th><th>MREQint</th><th>Context</th></tr>
<tr><td>0</td><td>0</td><td>No interrupts</td></tr>
<tr><td>0</td><td>1</td><td>Invalid state</td></tr>
<tr><td>1</td><td>1</td><td>The location device requests a modem interrupt.</td></tr>
<tr><td>1</td><td>0</td><td>External interrupt pending. The local device has no modem interrupts.</td></tr>
</table>
</td></tr>
</table>

9.2.3.2 Modem Request Acknowledge Register\

Register Name:							8-Bit Hex Address: $75
Register Description: Modem Request Acknowledge Register							Intel Hex Address: $EA
Default Value: 80							Motorola Hex Address: $EB
Access: Read only							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Modified Interrupt Vector provided on read							

9.2.3.3 Transmit Request Acknowledge Register

Register Name:							8-Bit Hex Address: $76
Register Description: Transmit Request Acknowledge Register							Intel Hex Address: $EC
Default Value: 80							Motorola Hex Address: $ED
Access: Read only							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Modified Interrupt Vector provided on read							

9.2.3.4 Receive Request Acknowledge Register

Register Name:							8-Bit Hex Address: $77
Register Description: Receive Request Acknowledge Register							Intel Hex Address: $EE
Default Value: 80							Motorola Hex Address: $EF
Access: Read only							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Modified Interrupt Vector provided on read							

The Service Request Acknowledge registers are read-only registers that return an appropriate interrupt vector when read. Reading one of these registers has the effect of a service acknowledgment cycle in the CD1865 (not necessarily the one addressed; it may be one further down the daisy chain). The vector supplied on the data bus during the cycle is described under the Global Service Vector register description. RegAckEn must be set for these registers to operate properly.

9.2.3.5 Global Channel Registers 1

Register Name:						8-Bit Hex Address: $41	
Register Description: Global Service Channel Register 1						Intel Hex Address: $82	
Default Value: 0						Motorola Hex Address: $83	
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Binary Value			C2	C1	C0	Binary Value	

9.2.3.6 Global Channel Registers 2

Register Name:						8-Bit Hex Address: $42	
Register Description: Global Service Channel Register 2						Intel Hex Address: $84	
Default Value: 0						Motorola Hex Address: $85	
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Binary Value			C2	C1	C0	Binary Value	

9.2.3.7 Global Channel Registers 3

Register Name:						8-Bit Hex Address: $43	
Register Description: Global Service Channel Register 3						Intel Hex Address: $86	
Default Value: 0						Motorola Hex Address: $87	
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Binary Value			C2	C1	C0	Binary Value	

There are three registers used to provide the channel number of the channel requesting service. Reading any of these registers causes the CD1865 to ‘mask-in’ three bits, specifying the channel number of the currently active channel. Normally these registers are read by the host when it is handling a service request. In this case, the three bits are the number of the channel requesting service. If any of the three registers are read when the CD1865 is not in a service request context, the three bits are the current value in the CAR. Bits 4:2 are masked into the contents of this register by the CD1865 when it is read by the host. The actual contents of the register are not modified.

These three registers are provided as a convenience to the user. In most applications, the user uses one of these locations, and sets the register to an arbitrary value. All types of service routines would use this register. However, in some cases it may be useful to be able to record information about the state of the CD1865 (or the software driving it) that is associated with each of the three service request types. In this case, the user may associate an individual register with each level of service request, and store whatever information is required in the unused bits. When entering a service routine, the software can check these bits (a sub-vector) to read recorded states.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>Bits 7:5</td><td>User-defined</td></tr>
<tr><td>Bits 4:2</td><td>Defines the service requesting channel number.
<table>
<tr><th>C2</th><th>C1</th><th>C0</th><th>Channel Number</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Channel 0</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Channel 1</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Channel 2</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Channel 3</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Channel 4</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>Channel 5</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Channel 6</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Channel 7</td></tr>
</table></td></tr>
<tr><td>Bits 1:0</td><td>User-defined</td></tr>
</table>

9.2.3.8 Channel Access Register

Register Name:
Register Description: Channel Access Register
Default Value: 0
Access: Read/Write

8-Bit Hex Address: $64
Intel Hex Address: $C8
Motorola Hex Address: $C9

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Reserved				A7(0)	C2	C1	C0

This register contains the channel number used for channel-oriented host read or write operations when the host is not in a service request service routine. When the CD1865 and the host are in a service request routine, the CD1865 supplies the service-requesting channel number by the Global Interrupting Channel register. The Channel Access register contents are not used during service request. The host service request routine is restricted to accessing only the register set of the service-requesting channel and the Global registers.

The Channel Access register is used by the host when the host is setting up or modifying the configuration of the channel. It is also used to issue certain channel-specific commands such as sending a flow-control character.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>Bits 7:4</td><td>Reserved, must be a ‘0’.</td></tr>
<tr><td>Bit 3</td><td>Internally, to the CD1865, this is Address bit 7. This bit completes the external to internal CD1865 register address mapping, but it is only to be used for test purposes. In normal operation, this bit should always be a ‘0’.</td></tr>
<tr><td>Bits 2:0</td><td>Channel number
<table>
<tr><th>C2</th><th>C1</th><th>C0</th><th>Channel Number</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Channel 0</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Channel 1</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Channel 2</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Channel 3</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Channel 4</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>Channel 5</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Channel 6</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Channel 7</td></tr>
</table></td></tr>
</table>

9.3 Indexed Indirect Registers

Certain registers are specially designed to facilitate service-request handling. These registers do not exist as distinct registers, and can be thought of as pointers. These registers provide functions that are valid only during service-request service routines, and they must not be accessed at other times.

Three of the registers are actually pointers to the Transmit and Receive FIFOs, that is, when referenced they cause the appropriate FIFO to be accessed. These registers are: Receive Data register, Receive Character Status register, and Transmit Data register.

The CD1865 maintains all channel-specific information. During data transfer between the host and the CD1865, the CD1865 uses a context-switching technique to switch the proper channel-specific information into the Global registers for use by the host. This reduces the processing burden on the host by eliminating the need to calculate address offsets.

9.3.1 Receive Data Count Register

Register Name: 8-Bit Hex Address: $07
Register Description: Receive Data Count Register Intel Hex Address: $0E
Default Value: 0 Motorola Hex Address: $0F
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

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>Bits 7:4</td><td>Reserved, must be a ‘0’.</td></tr>
<tr><td>Bits 3:0</td><td>Specifies the number of Good Data bytes for transfer from the Receive FIFO at the time of service request. This may be larger or smaller than the threshold level set by the user. This register reflects the actual amount of data available, which can be greater than the threshold level if service-request response is slow, or less than the threshold if some other event (such as an error condition) has caused the Receive Good Data Interrupt. This register need only be read when receiving Good Data; by default all exceptions are one character, and the value in this register during a Receive Exception is not defined or meaningful. The RDCR contains a zero if the current service request is for the NNDT case.
<table>
<tr><th>C3</th><th>C2</th><th>C1</th><th>C0</th><th>Number of Good Bytes</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>Does not occur</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>2</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>3</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>4</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>5</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>6</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>7</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>8</td></tr>
<tr><td colspan="4">1001 to 1111</td><td>Does not occur</td></tr>
</table>
</td></tr>
</table>

9.3.2 Receive Data Register

Register Name:
Register Description: Receive Data Register
Default Value: 0
Access: Read Only

8-Bit Hex Address: $78
Intel Hex Address: $F0
Motorola Hex Address: $F1

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
D7	D6	D5	D4	D3	D2	D1	D0

This register accesses the Receive Data FIFO for the channel. It is used by all channels to transfer Receive FIFO data to the host. Successive reads transfer bytes from the FIFO to the host. Reading this register increments an internal pointer to the Data and Status FIFOs. During service-request routines for Good Data, this is the only register that must be read. During service-request routines for Receive Exception, the Receive Status register must be read first, then this register may be read. If both the RCSR and this register are to be read, the RCSR must be read first because reading this register causes the FIFOs to ‘pop’.

Any attempt to write to this register causes unpredictable results.

9.3.3 Receive Character Status Register

Register Name:						8-Bit Hex Address: $7A	
Register Description: Receive Character Status Register						Intel Hex Address: $F4	
Default Value: 0						Motorola Hex Address: $F5	
Access: Read Only							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Time-out	SC Det2	SC Det1	SC Det0	Break	PE	FE	OE

This register accesses the status information for the current receive character.

Bit	Description
Bit 7	**Time-out:** Indicates that the Receive FIFO is empty, and no data has been received within the receive time-out period. There is no data character associated with this status and no other status bits are valid if the Time-out bit is set. Must be ‘armed’ by the NNDT bit in .
Bits 6:4	**Special Character Detect (SCD0-2):** SCD2 / SCD1 / SCD0 / Status 0 / 0 / 0 / None detected 0 / 0 / 1 / Special Character 1 or Special Character 1 and 3 sequence matched (only if Special Character 1 and 3 sequence is enabled). 0 / 1 / 0 / Special Character 2 or Special Character 2 and 4 sequence matched (only if Special Character 1 and 3 sequence is enabled). 0 / 1 / 1 / Special Character 3 (only if Special Character 1 and 3 sequence is not enabled). 1 / 0 / 0 / Special Character 4 (only if Special Character 2 and 4 sequence is not enabled). **NOTE:** No special-character match is performed if any type of error occurs. The second character of a two-character sequence cannot cause a receiver overrun.
Bit 3	**Break:** Indicates that a break has been detected.
Bit 2	**Parity Error:** Indicates that a parity error has been detected.
Bit 1	**Framing Error:** Indicates that a bad Stop bit has been detected.
Bit 0	**Overrun Error:** Indicates that new data has arrived but the CD1865 FIFO and Holding registers are full. The new data is lost and the overrun indication is flagged on the last character received before the overrun occurred.

Multiple errors in 1 byte are possible because the CD1865 evaluates the character bit-by-bit as it receives it. For example, a parity error is detected and flagged before a framing error. If a character is received with every bit (including the stop bit) equal to a ‘0’, it is marked as a line-break. If some bits are a ‘1’, but the Stop bit is ‘missing’ a ‘0’, it is marked as a framing error. If odd parity is set and the bits received are all zeroes, it is marked as both a break character and a parity error. In addition to any other bits, the Overrun bit is set if an overrun has occurred. Any attempt to write to this register causes unpredictable results.

9.3.4 Transmit Data Register

Register Name:							8-Bit Hex Address: $7B
Register Description: Transmit Data Register							Intel Hex Address: $F6
Default Value: 0							Motorola Hex Address: $F7
Access: Write Only							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
D7	D6	D5	D4	D3	D2	D1	D0

When servicing a Transmit Data Service Request, the Transmit Data register accesses the Transmit FIFO of the service-requesting channel. Data is written to the Transmit Data register by the host; the CD1865 automatic FIFO pointer mechanism places the data into the service-requesting channel's Transmit Character FIFO. Up to 8 bytes of data may be written into the TDR during Transmit Data Service Request.

Any attempt to read from this register causes unpredictable results.

9.3.5 End-of-Service Request Register

Register Name:							8-Bit Hex Address: $7F
Register Description: End-of-Service Request Register							Intel Hex Address: $FE
Default Value: 0							Motorola Hex Address: $FF
Access: Write Only							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Irrelevant Value							

This is a dummy register, and must be written to by the host's service request routine to signal to the CD1865 that the current service-request service is concluded. This must be the last access to the CD1865 during a service-request routine. Writing to this register generates an internal End Of Service signal, which 'pops' the CD1865's service-request-context stack, allowing the CD1865 to resume normal processing and also service other channels. Service-request contexts may be nested, as explained in Section 5.4; that is, one can respond to and service a higher-priority event while in the middle of a lower-priority service request routine (nesting subroutine calls within other subroutines).

Any attempt to read from this register causes unpredictable results.

9.4 Channel Registers

There are eight sets of Channel registers, but only one set is available at any given time. This offers the software-simplifying advantage that a given register is at the same address regardless of the channel number. To access a given channel's registers, first point to them by writing the channel number to the Channel Access register.

9.4.1 Enable Register

Register Name:	8-Bit Hex Address: $02
Register Description: Service Request Enable Register	Intel Hex Address: $04
Default Value: 0	Motorola Hex Address: $05
Access: Read/Write	

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
DSR	CD	CTS	RxD	RxSC	TxRdy	TxMpty	NNDT

A ‘1’ in each bit position enables service request generation for the associated cause.

Bit	Description
Bit 7	**Data-Set-Ready (DSR) Service Request:** When enabled, generates a Modem-Change Service Request on the selected level changes of the DSR Input.
Bit 6	**Carrier Detect (CD) Service Request:** When enabled, generates a Modem-Change Service Request on the selected level changes of the CD Input.
Bit 5	**Clear-To-Send (CTS) Service Request:** When enabled, generates a Modem-Change Service Request on the selected level changes of the CTS Input.
Bit 4	**Receive Data Service Request:** When enabled, the Receive Data Service Request is generated for receive data and Receive Exceptions.
Bit 3	**Receive Special Character (RxSC) Service Request:** When enabled, the Receive Data Exception Service Request is generated when a received character matches one of the four user-defined special characters. When disabled, Receive Exceptions are generated for error conditions and time-outs only. If flow-control transparency is set, flow-control characters are stripped, and no Receive Special Character Exceptions occurs.
Bit 2	**Transmit Ready (TxRdy) Service Request:** When enabled, the transmitter generates a service request when the Transmit FIFO becomes empty. Set this bit when first beginning transmission on a channel, and before attempting to write data to the Transmit FIFO. Enabling the service request causes an immediate Transmit Service Request, allowing it to write data into the Transmit FIFO in the usual manner. This bit may be set and cleared as needed to regulate the assertion of Transmit Data Service Requests on each channel. This technique is preferred over disabling the transmitter.
Bit 1	**Transmitter Empty (TxMpty) Service Request:** When enabled, a service request is generated when the Transmit FIFO, the Transmit Holding register, and the Transmit Shift register are all empty. This mode is provided to allow the host to determine when all bits are sent and it is safe to alter a channel’s configuration.
Bit 0	**No New Data Time-out (NNDT) Service Request:** When enabled, a Receive Exception Service Request is generated after the completion of data transfer from the CD1865 to the host. This feature assists in buffer management by providing a notice of a gap in the Receive Data Stream longer than the time-out period.

9.4.2 Channel Command Register

Register Name:	8-Bit Hex Address: $01
Register Description: Channel Command Register	Intel Hex Address: $02
Default Value: 0	Motorola Hex Address: $03
Access: Read/Write	

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
RESET CHAN	COR CHNG	SEND SP CH	CHAN CTL	D3	D2	D1	D0

The CCR is a special register used to prompt the CD1865 processor to indicate if any channel parameters have changed. Bits are set in the CCR to indicate which of several commands to carry out. The CD1865 processor notes changes in these bits and makes the required adjustments to the hardware; this process can take from microseconds to milliseconds. Therefore, it is important that the host CPU waits until the CD1865 processor has finished the current command before issuing any more commands, or continuing with any operation that the command affects. For example, after setting the Local Loopback bit in COR2, the host must wait until the command is complete before resuming transmission. If the host does not wait, characters may not be properly looped back.

Reset Channel, Channel Option, Send Special Character, and Channel Control commands can be set through the CCR register. One of the four commands can be selected by setting the appropriate bit (7:4). The commands can be defined in detail by setting the bit fields (3:0) accordingly. Bit fields (3:0) are defined differently by each command. The CD1865 indicates completion by clearing the CCR.

Bit	Description
Bit 7	Reset Channel Command.
Bit 6	Channel Option Register Command.
Bit 5	Send Special Character(s) Command.
Bit 4	Channel Control Command.
Bits 3:0	Defined by the type of command being issued; see the following descriptions.

The tables on the following pages define the appropriate setting of the bits according to the command.

Reset Channel Command

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
RESET CHAN	0	0	0	0	0	0	TYPE

This is a software reset command. There are two types of reset — Channel Reset (type 0), which resets only the current channel, and Global Reset (type 1), which resets the entire part to its power-up condition. When the channel reset command is issued, the CD1865 disables the transmitter and the receiver and clears the Data and Status FIFOs of the channel. Channel parameters are not affected by a Channel Reset.

Bit	Description
Bit 7	Reset Channel Command, must be a ‘1’.
Bits 6:1	Not used. Must be a ‘0’.
Bit 0	**Reset Type:** If the Reset Type bit is a ‘0’, a software reset of the channel is performed. The transmitter and receiver are disabled, and all FIFOs are cleared (flushed). If the Reset Type bit is a ‘1’, an on-device firmware initialization of all channels is performed. All channel and global parameters are reset to their power-on reset condition.

Channel Option Register Change Command

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	COR CHNG	0	0	COR3	COR2	COR1	N/U

Changes made to some Channel Option register bits must be signalled to the CD1865 by this command. Any combination of COR changes may be indicated by one command. All of the bits in COR3 take effect immediately, and all of the bits in COR2 (except LLM) take effect immediately. In other words, when changing COR3 or any of COR2 (except LLM), it is not necessary to issue a Channel Option register Change Command. However, to preserve compatibility with older CD1865 designs, it is acceptable to set these bits.

Bit	Description
Bit 7	Must be a ‘0’.
Bit 6	Channel Option Register Change Command, must be a ‘1’.
Bits 5:4	Must be a ‘0’.
Bit 3	Channel Option Register 3 changed (no longer required).
Bit 2	Channel Option Register 2 changed (required only for Local Loopback mode change).
Bit 1	Channel Option Register 1 changed.
Bit 0	Not used.

Send Special Character(s) Command

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	SEND SP CH	0	0	SSPC2	SSPC1	SSPC0

Bit	Description
Bits 7:6	Must be a ‘0’.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>Bit 5</td><td>Send Special Character(s) Command, must be a ‘1’.</td></tr>
<tr><td>Bits 4:3</td><td>Must be a ‘0’.</td></tr>
<tr><td>Bits 2:0</td><td>Special Character Select
<table>
<tr><th>SSPC2</th><th>SSPC1</th><th>SSPC0</th><th>Function</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Do not use</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Send Special Character 1, or characters 1 and 3 in sequence if COR3 [XonCH] defines a two-character sequence.</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Send Special Character 2, or characters 2 and 4 in sequence if COR3 [XoffCH] defines a two-character sequence.</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>Do not use</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Do not use</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Do not use</td></tr>
</table></td></tr>
</table>

Channel Control Command

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
0	0	0	CHAN CTL	XMTR EN	XMTR DIS	RCVR EN	RCVR DIS

Bit	Description
Bits 7:5	Must be a ‘0’.
Bit 4	Channel Control Command, must be a ‘1’.
Bit 3	Transmitter Enable
Bit 2	Transmitter Disable
Bit 1	Receiver Enable
Bit 0	Receiver Disable

When turning the receiver or transmitter on or off, it is faster to simply enable and disable service requests () rather than using the Channel Control Command.

9.4.3 Channel Option Register 1

Register Name: COR1	8-Bit Hex Address: $03
Register Description: Channel Option Register 1	Intel Hex Address: $06
Default Value: 0	Motorola Hex Address: $07
Access: Read/Write	

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Parity	ParM1	ParM0	Ignore	Stop 1	Stop 0	CHL 1	CHL 0

Changes to this register must be signalled by the Channel Command register

Bit	Description
Bit 7	**Parity:** 1 = odd parity. 0 = even parity.
Bits 6:5	**Parity Mode 1 and 0:** Defines Parity mode for both the transmitter and the receiver. **ParM1** **ParM0** **Parity** 0 0 No parity 0 1 Force parity (odd parity = force 1, even = force 0) 1 0 Normal parity 1 1 Not used
Bit 4	**Ignore:** Ignore parity 0 = Evaluate parity on received characters. 1 = Do not evaluate parity on received characters.
Bits 3:2	**Stop Bit Length:** Specifies the length of the Stop bit. **Stop1** **Stop0** **Stop Bit** 0 0 1 Stop bit 0 1 1 1/2 Stop bits 1 0 2 Stop bits 1 1 2 1/2 Stop bits
Bits 1:0	**Character Length:** **CHL1** **CHL0** **Character Length** 0 0 5 bits 0 1 6 bits 1 0 7 bits 1 1 8 bits

9.4.4 Channel Option Register 2

Register Name: COR2	8-Bit Hex Address: $04
Register Description: Channel Option Register 2	Intel Hex Address: $08
Default Value: 0	Motorola Hex Address: $09
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

Changes only to bit 4 (LLM) of this register must be signalled by the Channel Command register.

Bit	Description
Bit 7	**Implied Xon Mode (IXM):** This bit has meaning only when in the automatic Transmit In-Band Flow-control mode. During Transmit In-Band Flow-control mode, the CD1865 stops transmission upon detection of an Xoff character or character sequence. The IXM bit determines whether the CD1865 should restart transmission based on receipt of an Xon character or any character. When IXM bit is set, the CD1865 restarts transmission upon detection of any character. When IXM bit is not set, the CD1865 waits for the Xon character or character sequence to restart the transmission.
Bit 6	**Transmit In-Band (Xon/Xoff) Flow Control Automatic Enable (TxIBE):** The CD1865 in the Transmitting mode is flow-controlled by the remote. Upon receipt of the Xoff character, the CD1865 terminates transmission after the current character in the Transmit Shift register, and the character in the Transmit Holding register is sent. The CD1865 resumes transmission upon receipt of the Xon character, or any character, depending on the state of the IXM bit.
Bit 5	**Embedded Transmitter Command Enable (ETC):** If set, the embedded special transmitter command functions are enabled.
Bit 4	**Local Loopback Mode (LLM):** 1 = Enables the Local Loopback mode. 0 = Disables the Local Loopback mode.
Bit 3	**Remote Loopback Mode (RLM):** 1 = Enables the Remote Loopback mode. 0 = Disables the Remote Loopback mode.
Bit 2	**RTS Automatic Output Enable (RtsAO):** When set, if the channel is enabled, the CD1865 automatically asserts the RTS* Output when it has characters to send. If CtsAE is also set, it waits for CTS* to respond prior to transmission.
Bit 1	**CTS Automatic Enable (CtsAE):** Enables the CTS* Input to be used as automatic transmitter enable or disable.
Bit 0	**DSR Automatic Enable (DsrAE):** Enables the DSR* Input as automatic receiver enable or disable.

9.4.5 Channel Option Register 3

Register Name: COR3 — 8-Bit Hex Address: $05
Register Description: Channel Option Register 3 — Intel Hex Address: $0A
Default Value: 0 — Motorola Hex Address: $0B
Access: Read/Write

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Xon CH	Xoff CH	FCT	SCDE	RxTH3	RxTH2	RxTH1	RxTH0

Changes to this register do not have to be signalled by the CCR.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>Bit 7</td><td>Xon Character Definition:
0 = Xon Character is a single-character code, and it is defined by Special Character.
1 = Xon Character is a double-character sequence, and it is defined by Special Characters 1 and 3.</td></tr>
<tr><td>Bit 6</td><td>Xoff Character Definition:
0 = Xoff Character is a single-character code, and it is defined by Special Character 2.
1 = Xoff Character is a double-character sequence, and it is defined by Special Characters 2 and 4.</td></tr>
<tr><td>Bit 5</td><td>Flow-Control Transparency (FCT) Mode:
0 = Flow-control characters received are given to the host by Receive Exception Service Requests.
1 = Flow-control characters received are not given to the host by Receive Exception Service Requests.</td></tr>
<tr><td>Bit 4</td><td>Special-Character Detection Enable:
0 = Special-Character Status detection is disabled.
1 = Special-Character Status detection is enabled.</td></tr>
<tr><td>Bits 3:0</td><td>RxFIFO Threshold:
<table>
<tr><th>RxTh3</th><th>RxTh2</th><th>RxTh1</th><th>RxTH0</th><th>Status</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>Do not use</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>0</td><td>0</td><td>1</td><td>1</td><td>3 characters</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>4 characters</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>5 characters</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>6 characters</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>7 characters</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>8 characters</td></tr>
<tr><td colspan="4">1001 to 1111</td><td>Reserved, do not use.</td></tr>
</table></td></tr>
</table>

9.4.6 Channel Control Status Register

Register Name: CCSR
Register Description: Channel Control Status Register
Default Value: 0
Access: Read Only

8-Bit Hex Address: $06
Intel Hex Address: $0C
Motorola Hex Address: $0D

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

This Status register stores the current state of the channel. It may be read by the host at any time. If the host determines that a flow-control state is inappropriate, it may be cleared by enabling or disabling the transmitter or receiver by CCR command.

Bit	Description
Bit 7	**RxEn Receiver Enable:** 0 = Receiver is disabled. 1 = Receiver is enabled.
Bit 6	**RxFloff Receive Flow-off:** 0 = Normal 1 = The CD1865 has requested the remote to stop transmission (Send Xoff Command has been given to the channel). This bit is reset when the CD1865 has requested the remote to restart transmission, or when the receiver is enabled or disabled, or when the channel is reset.
Bit 5	**RxFlon Receive Flow-on:** 0 = Normal 1 = The CD1865 has requested the remote to restart character transmission (Send Xon Command has been given to the channel). This bit is reset when the next (non-flow control) character is received, or when the receiver is enabled or disabled, or when the channel is reset.
Bit 4	Not used
Bit 3	**TxEn Transmitter Enable:** 0 = Transmitter is disabled. 1 = Transmitter is enabled.
Bit 2	**TxFloff Transmit Flow-off:** 0 = Normal 1 = The CD1865 has been requested by the remote to stop transmission. This bit is reset when the CD1865 receives a request to resume transmission, or when the transmitter is enabled or disabled, or when the channel is reset.
Bit 1	**TxFlon Transmit Flow-on:** 0 = Normal 1 = The CD1865 has been requested by the remote to resume transmission. This bit is reset once character transmission is resumed, or when the transmitter is enabled or disabled, or when the channel is reset.
Bit 0	Not used

9.4.7 Receiver Bit Register

Register Name: RBR — 8-Bit Hex Address: $33
Register Description: Receiver Bit Register — Intel Hex Address: $66
Default Value: 21 — Motorola Hex Address: $67
Access: Read Only

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Reserved	RxD	Start Hunt	Reserved				

This register monitors certain functions of the actual receive hardware. It should *never* be written to as this causes the CD1865 to fail. Only two of the bits are defined herein; however, the other bit positions can change value, so these bits should be ‘masked-out’ before testing.

Bit 6 is the sampled state of the RxD pin, as sampled at the last bit-rate clock edge. This is not the actual RxD Input, as RxD cannot be sampled in real time. If no data has been received for a period of time, this bit still reflects the last sampled state of the line at the end of the last character. This is because the line is not sampled when the CD1865 is looking for the Start bit of a new character.

Bit 5 indicates whether the CD1865 is looking for a Start bit. If bit 5 is a ‘1’, it is looking. If bit 5 is a ‘0’, it is receiving a character.

9.4.8 Receive Time-Out Period Register

Register Name: RTPR
Register Description: Receive Time-Out Period Register
Default Value: 5
Access: Read/Write
8-Bit Hex Address: $18
Intel Hex Address: $30
Motorola Hex Address: $31

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Receiver Data Time-out Period							

This register defines the time period for two functions related to the Receive FIFO. As each character is moved to the Receive FIFO, the Receive Timer is reloaded with the Receive Data Time-out Period. The Receive Timer is then decremental on each tick of the Prescaler Counter. If the Receive Timer reaches a ‘0’, it causes a Receive Good Data Service Request.

There is another optional feature called No New Data Time-out. When enabled, the Receive Timer generates a Receive Exception if the timer expires after the last data is transferred from the FIFO to the host. This is intended to tell the host that no more data is arriving, and to go ahead and process the buffer.

The Receive Time-out Period register defines the time-out period for both of these functions. It counts in time increments defined by the prescaler.

9.4.9 Receive Bit Rate Period Registers (High/Low)

Register Name: RBPRH
Register Description: Receive Bit Rate Period Register (High)
Default Value: 0
Access: Read/Write
8-Bit Hex Address: $31
Intel Hex Address: $62
Motorola Hex Address: $63

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Receiver Bit Rate Divisor Byte							

Register Name: RBPRL
Register Description: Receive Bit Rate Period Register (Low)
Default Value: 0
Access: Read/Write
8-Bit Hex Address: $32
Intel Hex Address: $64
Motorola Hex Address: $65

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Receiver Bit Rate Divisor Byte							

These two registers contain the 16-bit pre-load value for the Receive Bit Rate Counter. This count establishes the basic Receiver Clock Rate, which must be 16 times the required Receiver Bit Rate. These registers are reset to a '0' by RESET*. The period established for the 16 times Receiver Clock Rate is equal to the RBPR 16-bit binary value times the System Clock (CLK) Period.

9.4.10 Transmit Bit Rate Period Registers (High/Low)

Register Name: TBPRH
Register Description: Transmit Bit Rate Period Register (High)
Default Value: 0
Access: Read/Write
8-Bit Hex Address: $39
Intel Hex Address: $72
Motorola Hex Address: $73

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Transmit Bit Rate Divisor Byte							

Register Name: TBPRL
Register Description: Transmit Bit Rate Period Register (Low)
Default Value: 0
Access: Read/Write
8-Bit Hex Address: $3A
Intel Hex Address: $74
Motorola Hex Address: $75

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Transmit Bit Rate Divisor Byte							

These two registers contain the 16-bit pre-load value for the Transmit Bit Rate Counter. This count establishes the Transmitter Clock Rate, which must be 16 times the required Transmitter Bit Rate. The precise period established for the 16 times Transmitter Clock is equal to the RBPR 16-bit binary value times the System Clock (CLK) Period. These registers are reset to a '0' by RESET*.

9.4.11 Special Character Register 1

Register Name: SCHR1
Register Description: Special Character Register 1
Default Value: 0
Access: Read/Write
8-Bit Hex Address: $09
Intel Hex Address: $12
Motorola Hex Address: $13

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Special Character 1							

This register stores the right-justified bit pattern for Special Character 1. Unused bits must be a '0'. During receive, this character is one of the four characters compared with the received data for special-character recognition. If a match occurs with one of these four characters, it is noted in the Receiver Status FIFO entry accompanying the received character unless a double-character compare is enabled. In this case, the Receive Status FIFO entry is not made until both characters are compared and matched.

During transmit, this register contains the characters that are sent as a result of the Send Special Character 1 command. If two-character sequences are enabled, Characters 1 and 3 are sent.

Special Character 1 defines the Xon character or the first-half of the Xon-character sequence. The second half is Special Character register 3.

9.4.12 Special Character Register 2

Register Name: SCHR2
Register Description: Special Character Register 2
Default Value: 0
Access: Read/Write
8-Bit Hex Address: $0A
Intel Hex Address: $14
Motorola Hex Address: $15

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Special Character 2							

This register stores the right-justified bit pattern for Special Character 2. Unused bits must be a ‘0’. During receive, this character is one of the four characters compared with the received data for special-character recognition. If a match occurs with one of these four characters, it is noted in the Receiver Status FIFO entry accompanying the received character unless a double-character compare is enabled. In this case, the Receive Status FIFO entry is not made until both characters are compared.

During transmit, this register contains the characters that are sent as a result of the Send Special Character 2 command. If two-character sequences are enabled, Characters 2 and 4 are sent.

Special Character 2 defines the Xoff character or the first-half of the Xoff-character sequence.

9.4.13 Special Character Register 3

Register Name: SCHR3
Register Description: Special Character Register 3
Default Value: 0
Access: Read/Write
8-Bit Hex Address: $0B
Intel Hex Address: $16
Motorola Hex Address: $17

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Special Character 3							

This register stores the right-justified bit pattern for Special Character 3. Unused bits must be a ‘0’. During receive, this character is one of the four characters compared with the received data for special character recognition. If a match occurs with one of these four characters, it is noted in the Receiver Status FIFO entry accompanying the received character unless a double-character compare is enabled. In this case, the Receive Status FIFO entry is not made until both characters are compared.

During transmit, this register contains the characters that are sent as a result of the Send Special Character 3 command.

Special Character 3 may be the second-half of the Xon-character sequence.

9.4.14 Special Character Register 4

Register Name: SCHR4							8-Bit Hex Address: $0C
Register Description: Special Character Register 4							Intel Hex Address: $18
Default Value: 0							Motorola Hex Address: $19
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Special Character 4							

This register stores the right-justified bit pattern for Special Character 4. Unused bits must be a ‘0’. During receive, this character is one of the four characters compared with the received data for special character recognition. If a match occurs with one of these four characters, it is noted in the Receiver Status FIFO entry accompanying the received character unless a double-character compare is enabled. In this case, the Receive Status FIFO entry is not made until both characters are compared.

During transmit, this register contains the characters that are sent as a result of the Send Special Character 4 command.

Special Character 4 may be the second-half of the Xoff-character sequence.

9.4.15 Modem Change Register

Register Name: MCR							8-Bit Hex Address: $12
Register Description: Modem Change Register							Intel Hex Address: $24
Default Value: 0							Motorola Hex Address: $25
Access: Read/Write							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
DSRchg	CDchg	CTSchg	0	0	0	0	0

The CD1865 sets bits in this register when it recognizes a level change on a modem pin, as programmed by the Modem Change Option registers. Changes detected are a cause for asserting the Modem Service Request if corresponding Service Request Enable bits are set. Once the service request is asserted, updates to this register are inhibited until End Of register () is written at the end of the Modem Service Request Routine. The host must clear these register bits during the service routine.

Bit	Description
Bit 7	**DSR Changed:** A logic ‘1’ denotes that the Data-Set-Ready Input has changed state.
Bit 6	**CD Changed:** A logic ‘1’ denotes that the Carrier Detect Input has changed state.
Bit 5	**CTS Changed:** A logic ‘1’ denotes that the Clear-to-Send Input has changed state.
Bits 4:0	Must be a ‘0’.

9.4.16 Modem Change Option Register 1

Register Name: MCOR1 Register Description: Modem Change Option Register 1 Default Value: 0 Access: Read/Write						8-Bit Hex Address: $10 Intel Hex Address: $20 Motorola Hex Address: $21	
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
DSRzd	CDzd	CTSzd	0	DTRth3	DTRth2	DTRth1	DTRth0

This register is used to define the current state change options to be monitored.

Bit	Description
Bit 7	**DSRzd is a '1':** Detect high-to-low voltage transition on DSR* Input (zero-to-one transition of DSR (MSVR) bit).
Bit 6	**CDzd is a '1':** Detect high-to-low voltage transition on CD* Input (zero-to-one transition of CD (MSVR) bit).
Bit 5	**CTSzd is a '1':** Detect high-to-low voltage transition on CTS* Input (zero-to-one transition of CTS (MSVR) bit).
Bit 4	Must be a '0'.
Bits 3:0	Defines the threshold level that causes negation of DTR* when this flow-control option is specified. Normally, this level should be equal to or higher than the service-request level threshold as set in COR3. If it is set lower than the service-request threshold, it defaults to the service-request threshold level.

DTRth3	DTRth2	DTRth1	DTRth0	Function
0	0	0	0	Automatic DTR mode disabled
0	0	0	1	1 character
0	0	1	0	2 character
0	0	1	1	3 character
0	1	0	0	4 character
0	1	0	1	5 character
0	1	1	0	6 character
0	1	1	1	7 character
1	0	0	0	8 character

9.4.17 Modem Change Option Register 2

Register Name: MCOR2	8-Bit Hex Address: $11
Register Description: Modem Change Option Register 2	Intel Hex Address: $22
Default Value: 0	Motorola Hex Address: $23
Access: Read/Write	

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

This register is used to define the current state change options to be monitored.

Bit	Description
Bit 7	**DSRod is a '1':** Detect low-to-high transition on DSR* Input (one-to-zero transition DSR (MSVR) bit).
Bit 6	**CDod is a '1':** Detect low-to-high transition on CD* Input (one-to-zero transition of CD (MSVR) bit).
Bit 5	**CTSod is a '1':** Detect low-to-high transition on CTS* Input (one-to-zero transition of CTS (MSVR) bit).
Bits 4:0	Must be a '0'.

9.4.18 Modem Signal Value Register

Register Name: MSVR	8-Bit Hex Address: $28
Register Description: Modem Signal Value Register	Intel Hex Address: $50
Default Value: 0	Motorola Hex Address: $51
Access: Read/Write	

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
DSR	CD	CTS	N/U	N/U	N/U	DTR	RTS

This register is read to determine the current input levels on the Modem Input pins. It is written to supply an output value for the RTS* and DTR* pins. The register bits have the opposite polarities from the actual states on the individual pins. Writing a '1' causes the pin to go to nominal zero volts.

Bit	Description
Bit 7	**DSR:** Current state of Data-Set-Ready Input.
Bit 6	**CD:** Current state of Carrier Detect Input.
Bit 5	**CTS:** Current state of Clear-to-Send Input.
Bits 4:2	Not used.
Bit 1	**DTR:** Current state of Data-Terminal-Ready Output.
Bit 0	**RTS:** Current state of Request-to-Send Output.

9.4.19 Modem Signal Value Request-to-Send Register

Register Name: MSVRTS
Register Description: Modem Signal Value Request-to-Send Register
Default Value: 0
Access: Write Only
8-Bit Hex Address: $29
Intel Hex Address: $52
Motorola Hex Address: $53

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

In the Modem Signal Value register, a write to either RTS or DTR affects the state of the other one. This can be a problem when the CD1865 is using one of these signals for flow control and the other one needs to be used under host control. This register writes to RTS without affecting the state of any other bits. RTS is at bit 0.

9.4.20 Modem Signal Value Data-Terminal-Ready Register

Register Name: MSVDTR
Register Description: Modem Signal Value Data Terminal Ready
Default Value: 0
Access: Write Only
8-Bit Hex Address: $2A
Intel Hex Address: $54
Motorola Hex Address: $55

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

In the Modem Signal Value register, a write to either RTS or DTR affects the state of the other one. This can be a problem when the CD1865 is using one of these signals for flow control and the other one needs to be used under host control. This register writes to DTR without affecting the state of any other bits. DTR is at bit 1.

Note: Before beginning any new design with this device, please contact Intel for the latest errata information. See the back cover of this document for sales office locations and phone numbers.

10.0 Electrical Specifications

10.1 Absolute Maximum Ratings

- Operating ambient temperature 0°C to 70°C
- Storage temperature –65°C to 150°C
- All voltages, with respect to ground –0.5 volts to V_{cc} + 0.5 volts
- Supply voltage (V_{cc}) +7.0 volts
- Power dissipation 0.5 watt

Note: Stress above those listed under Absolute Maximum Ratings may 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 operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

10.2 Recommended Operating Conditions

- Supply voltage (V_{cc}) 5 volts ± 5%
- Operating free-air ambient temperature 0°C < T_A < 70°C
- System clock 33 MHz

10.3 DC Electrical Characteristics

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

Symbol	Parameter	MIN	MAX	Units	Conditions
V_{IL}	Input low voltage	–0.5	0.8	V	
V_{IH}	Input high voltage	2.0	V_{cc}	V	
V_{OL}	Output low voltage		0.4	V	I_{OL} = 8 mA
V_{OH}	Output high voltage	2.4	V_{cc}	V	I_{OH} = –8 mA
I_{IL}	Input leakage current	–10	10	µA	0 < V_{in} < V_{cc}
I_{LL}	Data bus three-state leakage current	–10	10	µA	0 < V_{out} < V_{cc}
I_{OC}	Open drain output leakage	–10	10	µA	0 < V_{out} < V_{cc}
I_{CC}	Power supply current		90	mA	CLK = 33 MHz
C_{in}	Input capacitance		10	pF	
C_{out}	Output capacitance		10	pF	

10.4 Index of Timing Information

10.5 AC Electrical Characteristics

Internally, the CD1865 is a fully clocked design; however, the hardware interface to the CD1865 may be either unclocked or clocked. An unclocked interface is generally easier to implement, especially if the CD1865 and its host are operating at different clock speeds. A clocked interface may be faster in some applications.

10.5.1 Clocked Bus Interface

Data transfers to or from the device occur in two steps. The first step occurs during the clock-low time. If the read/write state machine detects that it is time to do a cycle, it acquires the internal bus. The second step, that of actually transferring the data, occurs during the clock-high time. The cycle is complete at the end of the clock-high time.

The read/write state machine determines that it is time to do a cycle when there is a falling edge on the clock and both CS* and DS* are low. There is a specified setup time which must be met to guarantee that the cycle begins. If this setup is not met, the cycle occurs one clock later. If the cycle is recognized, arbitration for the internal bus is done during the clock-low time. Addresses (and data, if a write cycle) must meet another setup time specification to the rising edge of the clock for the actual data transfer to occur properly during the clock-high time. In addition, the addresses must remain valid throughout the clock-high time, as specified. If the cycle is a write cycle, data must remain valid as specified. If the cycle is a read cycle, data is guaranteed valid for a specified time after the rising edge of the clock.

Service Acknowledge Cycles are a special case of read cycles. The service acknowledge ‘read’ (which returns the Global Service Request Vector value to the host) is started when the read/write state machine detects both DS* and another internal signal derived from both ACKIN* and DS*. There are two possible worst-case paths to consider when determining whether DS* and ACKIN* meet the necessary setup times to guarantee recognition on a particular clock edge. The longest path is DS*; it must propagate through a gate, an 8-bit comparator, a state machine, and another gate before arriving at the read/write state machine. The setup time for this is given in Table 10.

The other critical path is ACKIN*; it must pass through a state machine and a gate before arriving at the read/write state machine. The setup time to guarantee recognition on a particular clock edge is given in Table 10. Intel-style pin names are shown in {brackets}. All times are in nanoseconds, unless otherwise specified.

Table 10. Clocked Timings (Sheet 1 of 2)

Number in Figures	Description	MIN (1)	MAX (1)	Notes
t_1	Setup, DS*{RD*} and CS* low to CLK low, for read or write cycle to start (‘ordinary’ reads and all writes)	10		2
t_2	Setup, DS* {RD*} low to CLK low, for service acknowledge cycle to start (ACKIN* cycles and read cycles from acknowledge registers)	15		3
t_3	Setup, ACKIN* low to CLK low for cycle to start	10		
t_4	Setup, Address valid to CS* and DS* low	3		
t_5	Setup, Address valid to DS* (service acknowledge cycles)	4		4
t_6	Setup, Write Data valid to CLK high	0		
t_7	Setup, R/W* {RD*, WR*} stable to DS* and CS* low (read, write cycles)	0		2, 5
t_8	(DS* and CS*), or (RD* and CS*), or (WR* and CS*), high	5		6, 7
t_9	Hold time, CS* low after CLK high (read, write cycles)	5		8
t_{10}	Hold time, DS* {RD*} after valid data	0	Infinity	8
t_{11}	Hold time, Address valid after CLK high	15		8
t_{12}	Hold time, Write Data valid after CLK high	10		
t_{13}	Hold time, ACKIN* low after next CLK low	4		9
t_{14}	Clock Period (T_{CLK})	30	200	10
t_{15}	Clock low time	12		10
t_{16}	Clock high time	12		10
	Clock duty cycle (50% ± 10%)			
t_{17}	Clock rise/fall time		3	11
t_{18}	RESET pulse width (after power is good and clock is stable)	5 clock periods		
t_{19}	Data Bus out of Hi-Z after CLK low	0		12
t_{20}	Read Data valid after CLK high		35	
t_{21}	ACKIN* to ACKOUT* propagation delay		12	
t_{22}	ACKOUT* high after ACKIN* high		12	
t_{23}	DS* {RD*} high to data bus three-state	0	10	
t_{24}	DTACK* assert after CLK high (DTACKDLY = 0)		25	
t_{25}	DTACK* assert after CLK low (DTACKDLY = 1)		20	

Table 10. Clocked Timings (Sheet 2 of 2)

Number in Figures	Description	MIN (1)	MAX (1)	Notes
t_{26}	DTACK* negate after DS* {RD* or WR*} negation		10	
t_{27}	ACKOUT* assert after CS* and DS* active on register acknowledge cycle with no match		22	13
t_{28}	DTACK* active pull-up time			14
t_{29}	ACKOUT* high after end of cycle		22	

NOTES:
1. Unless otherwise noted, all values are in nanoseconds (ns).
2. The reference to DS* and CS* refers to whichever one goes active last; that is, both signals must meet the setup time requirement.
3. Enabling the Register Acknowledge (‘regack’) feature changes the timing somewhat, even on cycles where ‘regack’ is not being used.
4. Calculated value; guaranteed by design, but not tested.
5. For Motorola-style interface, refers to R/W*.For Intel-style interface, refers to RD* or WR* (whichever is inactive for that cycle).
6. A cycle must positively end before another begins; that is, control signals shall return to states such that no cycle is pending or active.
7. Guaranteed by design, but not tested.
8. During Register Based Acknowledge cycles, these signals must be held in the correct state until valid data is presented by the device, as indicated by DTACK* going active. Note that in daisy-chain applications, the response from the chain may be quite long due to the ACKIN*-ACKOUT* propagation delay required for the actual interrupting device to receive the select (ACKIN*). Waiting for the active DTACK* from the chain eliminates any timing problems relating to these parameters.
9. ACKIN* must be low for at least one clock period plus setup and hold times if there is only one CD1865 in the daisy chain. If there is more than one CD1865 in a daisy chain, ACKIN* must be low until it has rippled all the way down the chain.
10. When using the clock out (CKOUT) of one CD1865 to drive subsequent CD1865s (such as in daisy-chain environments), CKOUT is skewed (delayed) by 3 ns from the internal clock. Therefore, on subsequent CD1865s, setup times are improved by 3 ns and hold times are derated by 3 ns.
11. For clock periods greater than 100 ns (10 MHz or less clock), rise and fall time may be 5 ns maximum.
12. Greater than a ‘0’ by design, but not tested.
13. This is the time for ACKOUT* to assert on register acknowledge cycles. ACKOUT* asserts if the device determines the acknowledgment is not intended for that part. If ACKOUT* asserts, the device does not drive the data bus or assert DTACK*. These functions are left to a device further down the daisy chain that accepts the acknowledge cycle.
14. DTACK* sources current (drives ‘high’) until the voltage on the DTACK* line is approximately 1.5 volts. Then DTACK* goes to an ‘open-drain’ (high-impedance) state.

Figure 29. Clocked Bus Interface Reset

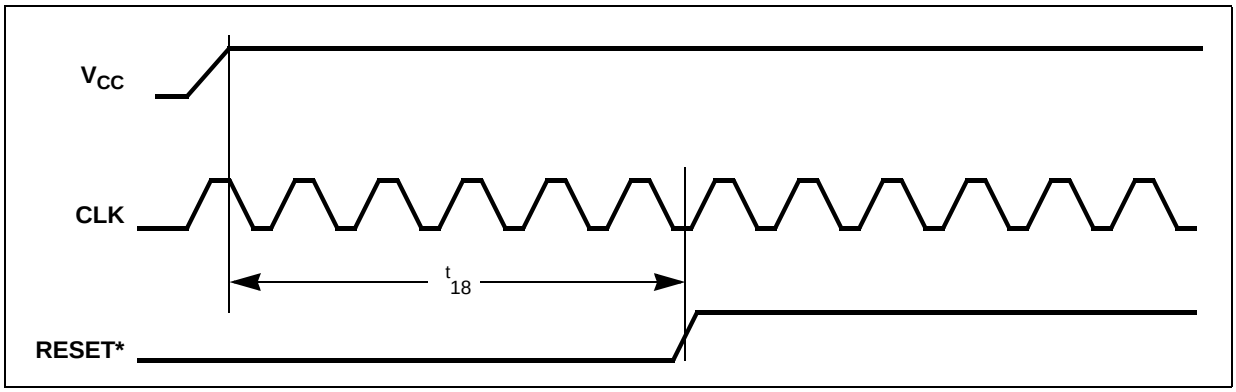


Figure 30. Clocked Bus Interface Clocks

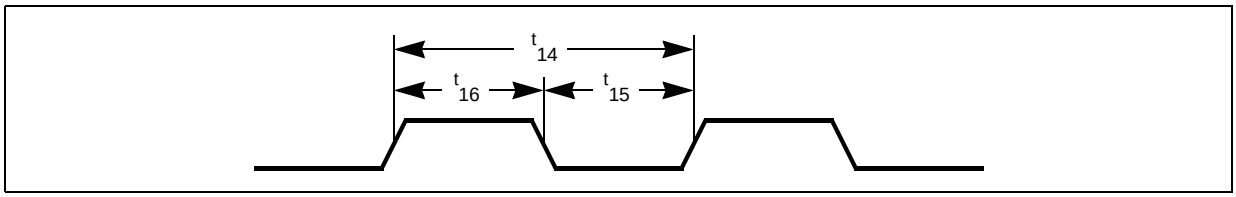


Figure 31. Clocked Bus Interface Read Cycle, Motorola®-Style Handshake

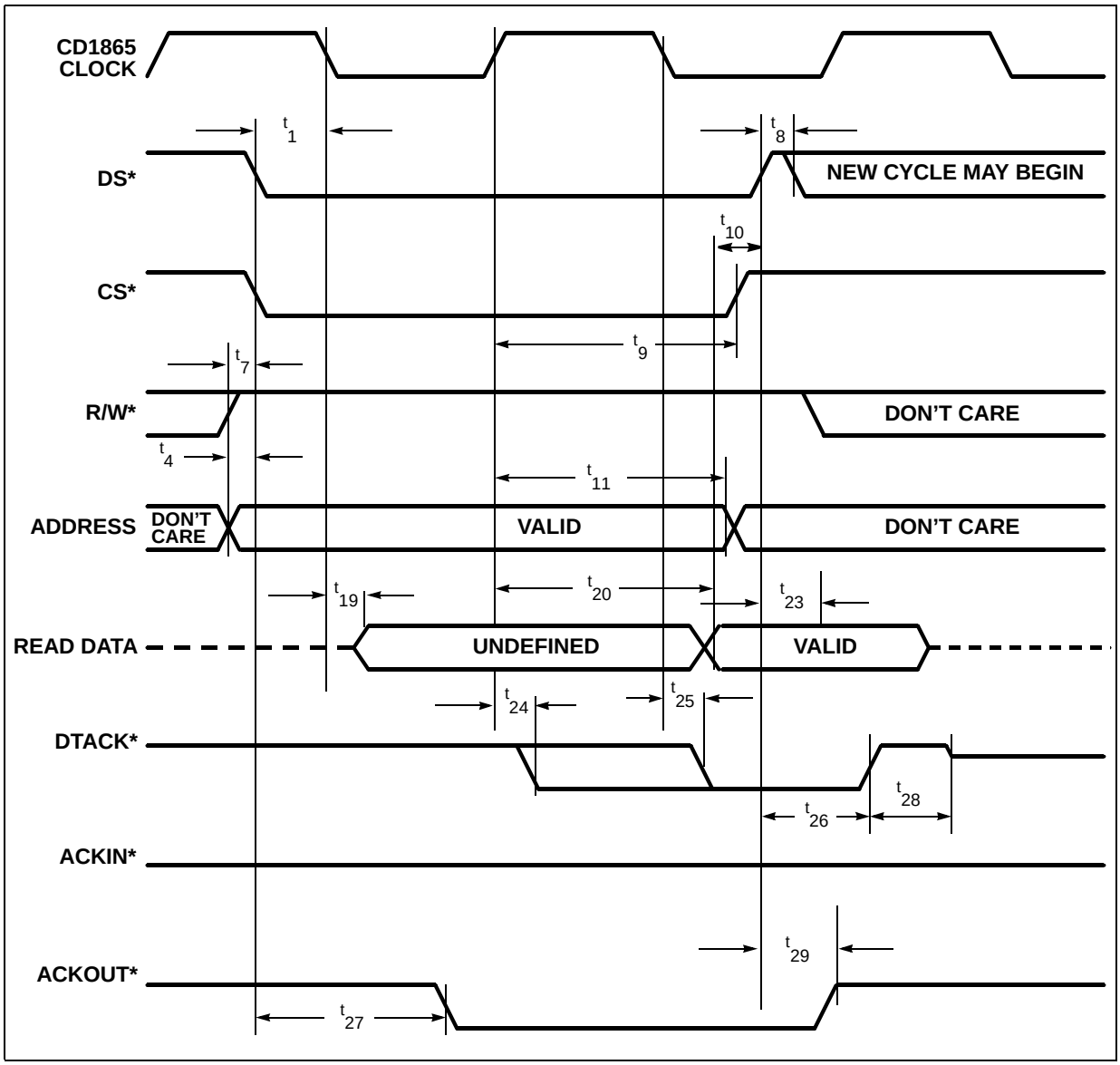


Figure 32. Clocked Bus Interface Service Acknowledgment Cycle, Motorola®-Style Handshake

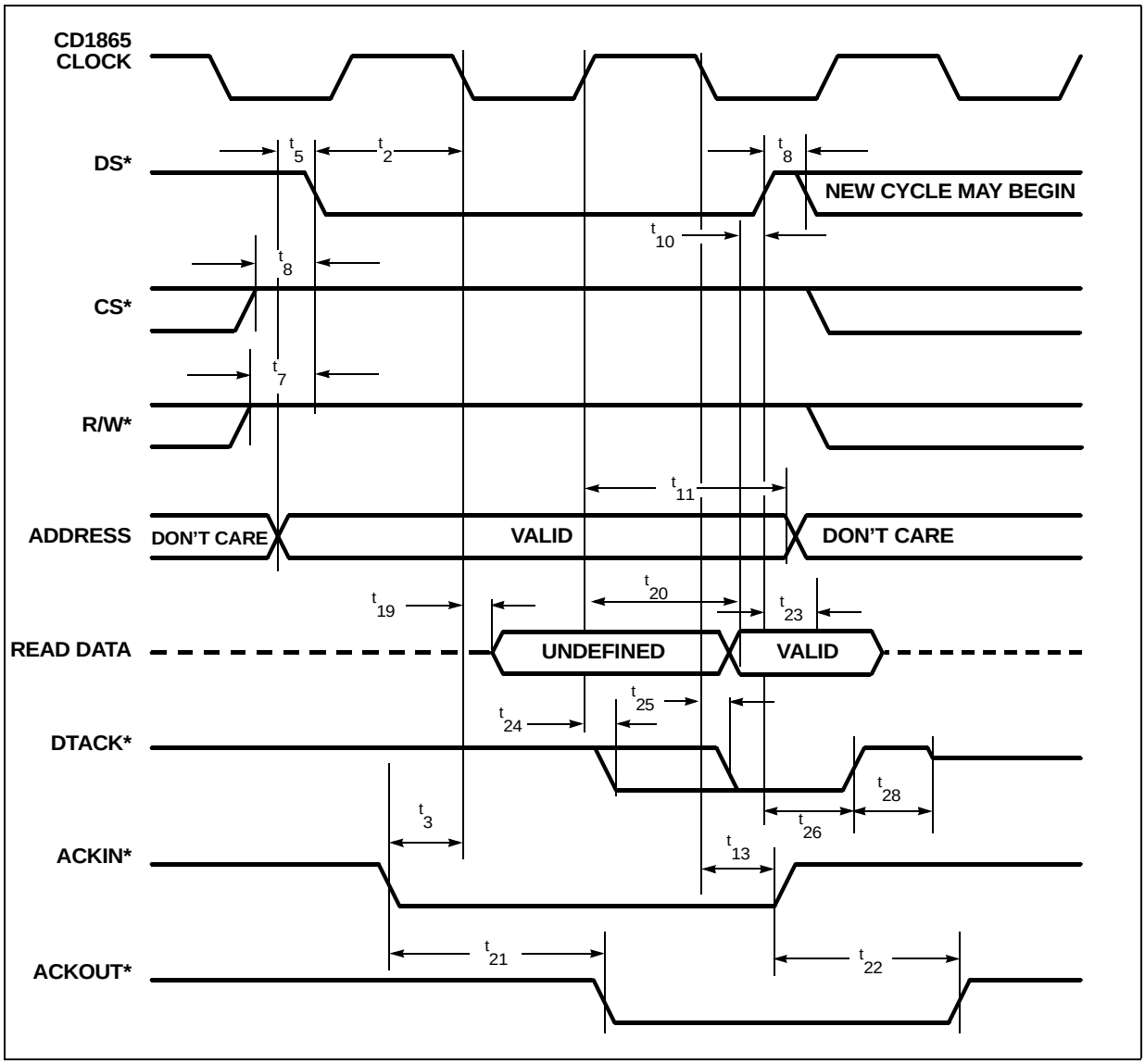


Figure 33. Clocked Bus Interface Write Cycle, Motorola®-Style Handshake

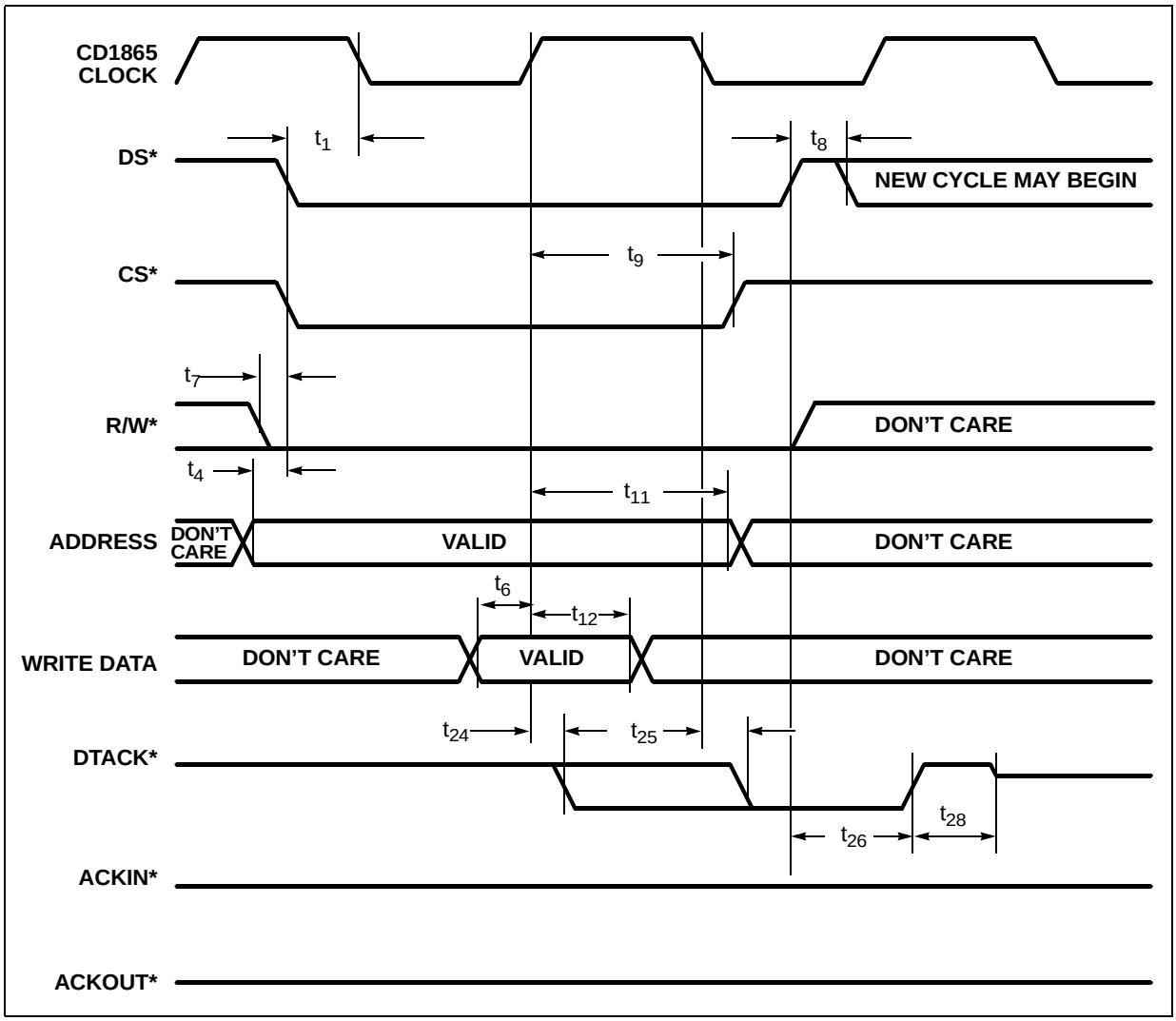


Figure 34. Clocked Bus Interface Read Cycle, Intel®-Style Handshake

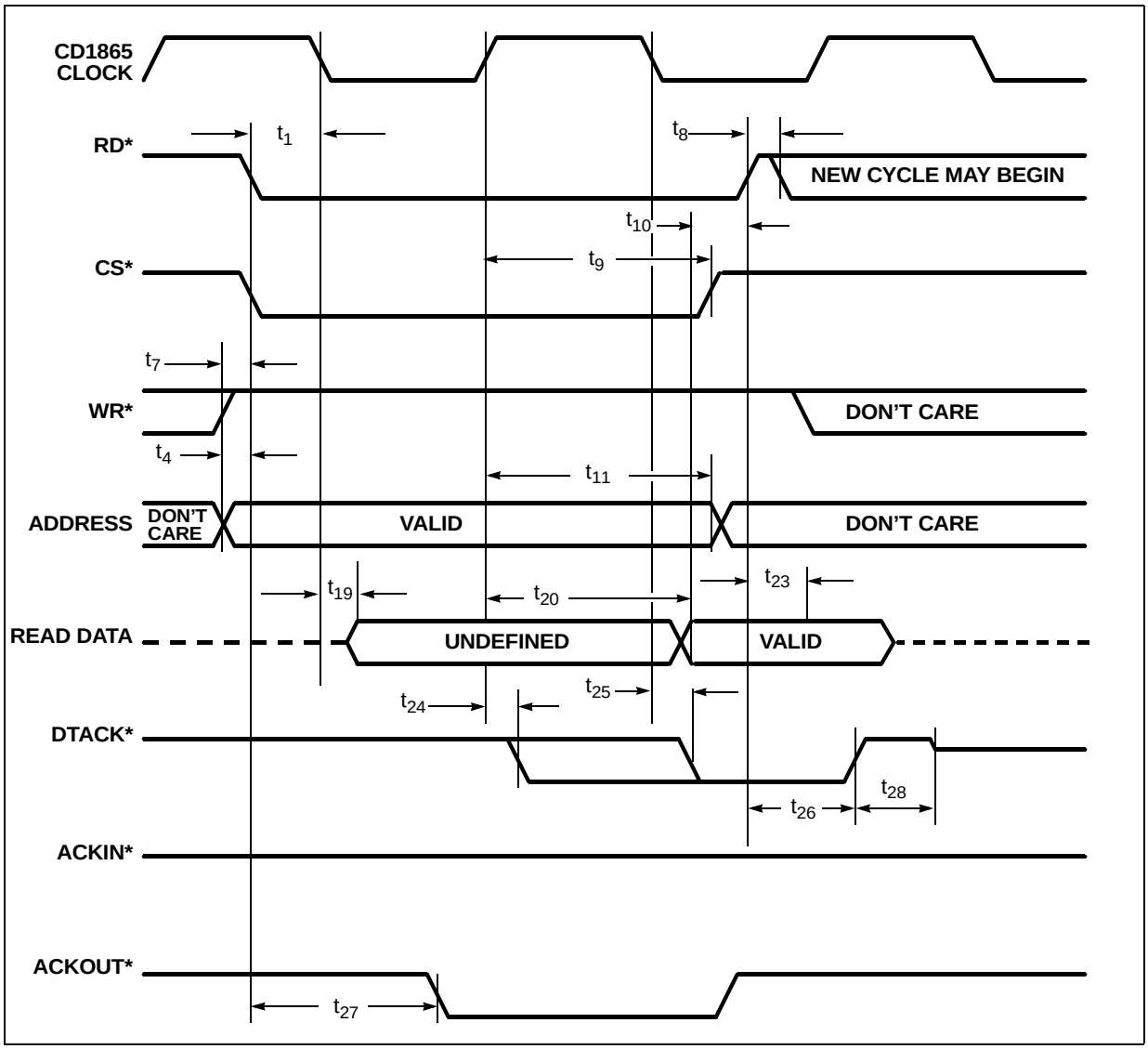


Figure 35. Clocked Bus Interface Service Acknowledgment Cycle, Intel®-Style Handshake

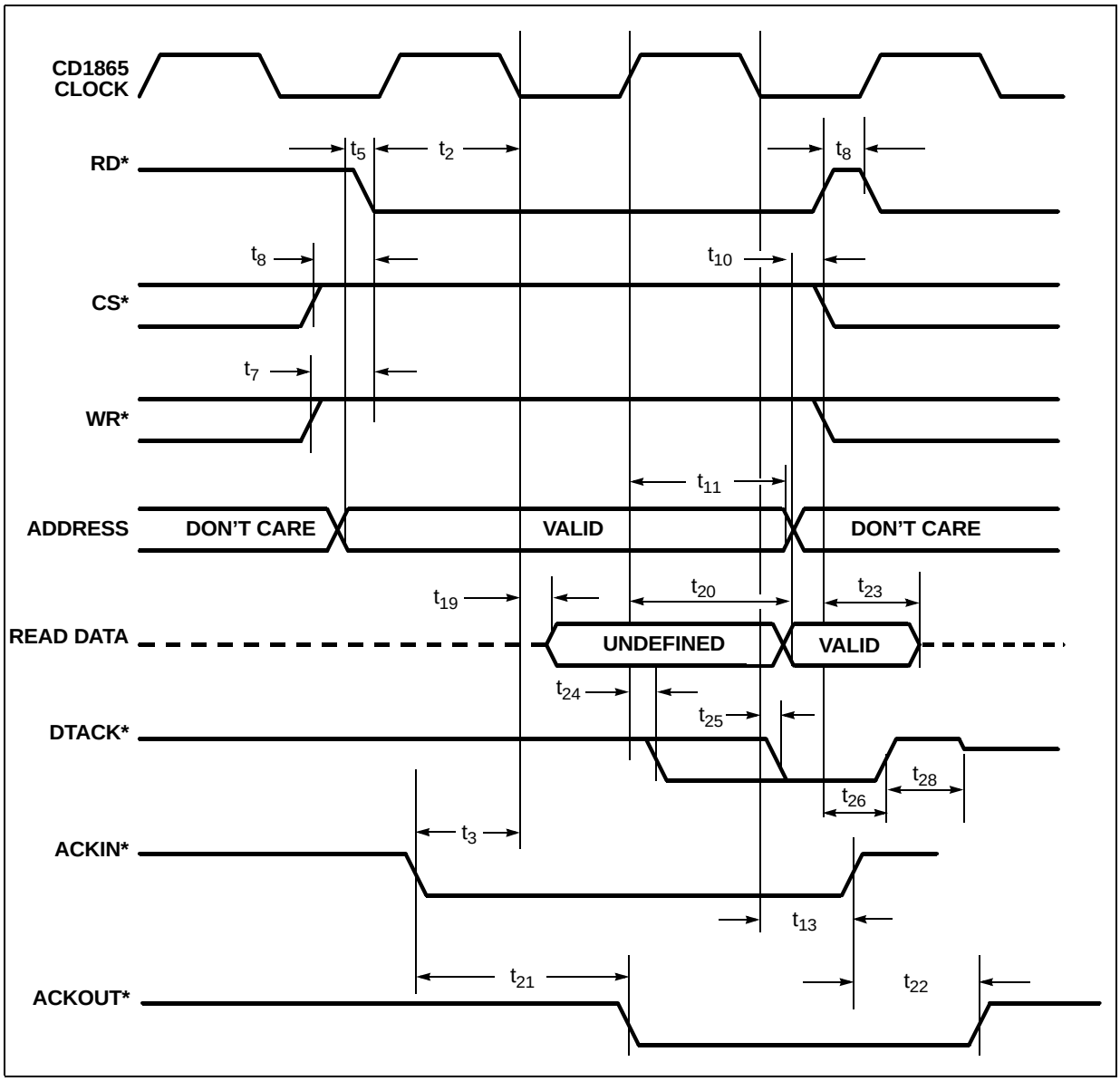
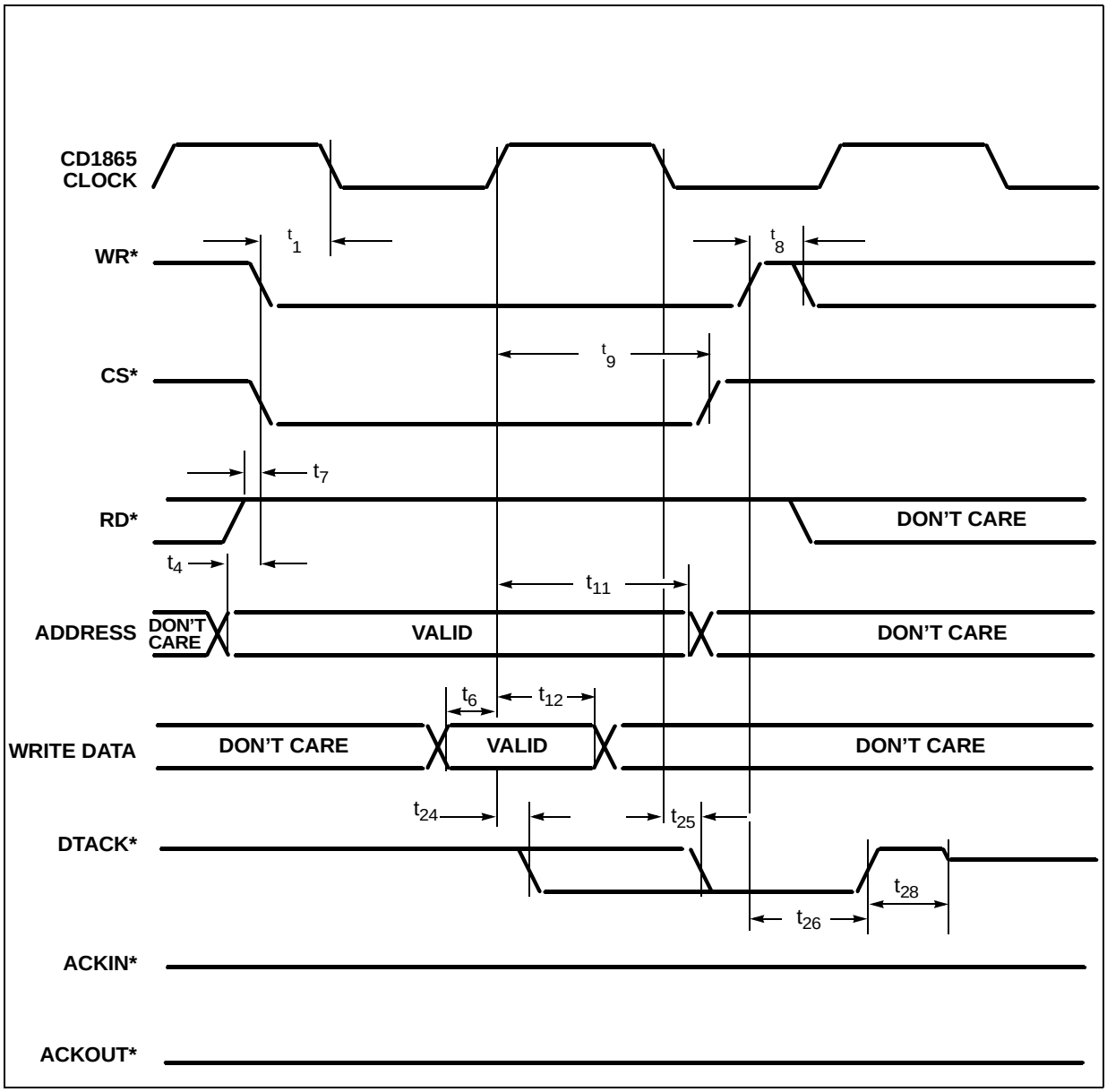


Figure 36. Clocked Bus Interface Write Cycle, Intel®-Style Handshake



10.5.2 Unclocked Bus Interface

Unclocked timing diagrams represent worst-case synchronization delays. That is, they reflect the maximum number of clock cycles required to complete the operation.

Internally, the CD1865 fully synchronizes all signals; thus, the user need not be concerned with setup times or metastability. The vast majority of CD1865 designs employ an unclocked Bus Interface.

All times are based on a master clock (CLK) of 15 MHz. All times are measured in nanoseconds. Intel-style handshake signals (where appropriate) are shown in {curly brackets}.

Table 11. Unclocked Timings (Sheet 1 of 2)

Number	Description	MIN1	MAX1	Notes
t_1	Setup time, Address to CS*, DS* {CS*, RD* or WR*}	3		2
t_2	Setup time, R/W* to CS* or DS*	0		2
t_3	Hold time, Address after CS* or DS* {CS* or RD* or WR*}	0		3, 4
t_4	R/W* hold time after CS* and DS*	3		3, 4
t_5	Delay time, DTACK* assert to valid Read Data: If DTACKDLY = 0 If DTACKDLY = 1		 25 -12	
t_6	DTACK* assert after CS* or DS* {RD*} or ACKIN* If DTACKDLY = 0 If DTACKDLY = 1		 75 85	2, 5
t_7	Hold time, Read Data after CS* and DS*{RD*} high	1	12	3, 6, 7
t_8	CS* or DS* {RD*} high from DTACK* low If DTACKDLY = 0 If DTACKDLY = 1	 25 1		 4, 4., 8, 7
t_9	DTACK* inactive from (CS* or ACKIN*) or DS* high		12	3, 9, 4
t_{10}	DS* {RD*} high pulse width	5		4
t_{11}	Setup time, Address to ACKIN*	10		10, 11
t_{12}	Setup time, Write Data to DS* {or WR*} low		0	
t_{13}	Hold time, Write Data after DS* {or WR*} high	0		
t_{14}	x_REQ* deassert after DTACK* asserted		2^{T}clk+30	12
t_{15}	Setup time, R/W* {WR*} and CS* to ACKIN* low	0		13
t_{16}	x_REQ* reassert delay after write to EOSRR		2^{T}clk+30	14, 15
t_{17}	ACKIN* assert/deassert to ACKOUT* assert/deassert prop delay		15	
t_{18}	Data bus out of high-impedance after DS* {RD*} low	3		16
t_{19}	Setup time, Address to DS* {RD*} during acknowledge cycles	4		

Table 11. Unclocked Timings (Sheet 2 of 2)

Number	Description	MIN1	MAX1	Notes
t_{20}	ACKOUT* assert after CS* and DS* {RD*} active on register acknowledge cycles with no match		25	17
t_{21}	DTACK* active pull-up time			18

NOTES:
1. Unless otherwise noted, all values are in nanoseconds (ns).
2. During read cycles, CS* and DS* {RD*} are gated together internally. This specification is with respect to whichever goes active (low) last.
3. During read cycles, CS* and DS* {RD*} are gated together internally. This specification is with respect to whichever goes inactive (high) last.
4. This specification is with respect to whichever goes inactive (high) last.
5. The values given is for 15-MHz operation. The time depends on system clock rate and the chosen DTACKDLY option. The actual time in any case can be determined by the formula:
   If DTACKDLY = 0, then the time is 1.5(Tclk) + 30 ns
   If DTACKDLY = 1, then the time is 2.0(Tclk) + 35 ns
6. This specification is with respect to whichever of ACKIN* and DS* {RD*} goes active (low) last.
7. The data bus is three-stated immediately after removal of DS* {RD*}. The device is guaranteed to be off the bus by the specified maximum time. The time can be as short as the minimum time. The hardware design should ensure that the data has been read before DS* {RD*} is removed.
4. In multiple CD1865 designs, the Interrupt Acknowledge cycle must be long enough to accommodate the ACKIN* to ACKOUT* daisy-chain propagation delay from the first to the last CD1865. ACKIN* must remain low until after DTACK* asserts.
8. For Acknowledge cycles, this specification refers to ACKIN* instead of CS*.
9. During Interrupt Acknowledge cycles, ACKIN* is asserted instead of CS*; CS* should remain high. Note that ACKIN* timing is not always the same as CS*.
10. During acknowledge cycles, addresses must propagate through the Service Match Registers. If a service request is pending on this CD1865, the match must finish before ACKIN* asserts. This is ensured by the specifications.
11. This specification is with respect to ACKIN* only.
12. This specification refers to one of Receive, Transfer, or Modem Service Request Outputs (RREQ*, TREQ*, MREQ*).
13. This specification is with respect to DS*. CS* and R/W* must be high before the assertion of DS* to avoid the possibility of the CD1865 misinterpreting the cycle as a read or write.
14. This is the time required to reassert a service request if the internal conditions of the CD1865 are such that the request should be asserted.
15. This specification refers to one of Receive, Transfer, or Modem Service Request Outputs (RREQ*, TREQ*, MREQ*).
16. The data bus is guaranteed to become active after DS* {RD*} low and before data is valid.
17. This is the time for ACKOUT* to assert on register acknowledge cycles. ACKOUT* asserts if the part determines the acknowledgment is not intended for that part. If ACKOUT* asserts, the part does not drive the data bus or assert DTACK*. These functions are left to a device further down the daisy chain that accepts the acknowledge cycle.
18. DTACK* sources current (drives ‘high’) until the voltage on the DTACK* line reaches 1.5V. At that time, DTACK* switches to an ‘open-drain’ (high-impedance) state.

Figure 37. Unclocked Bus Interface Read Cycle, Motorola®-Style Handshake

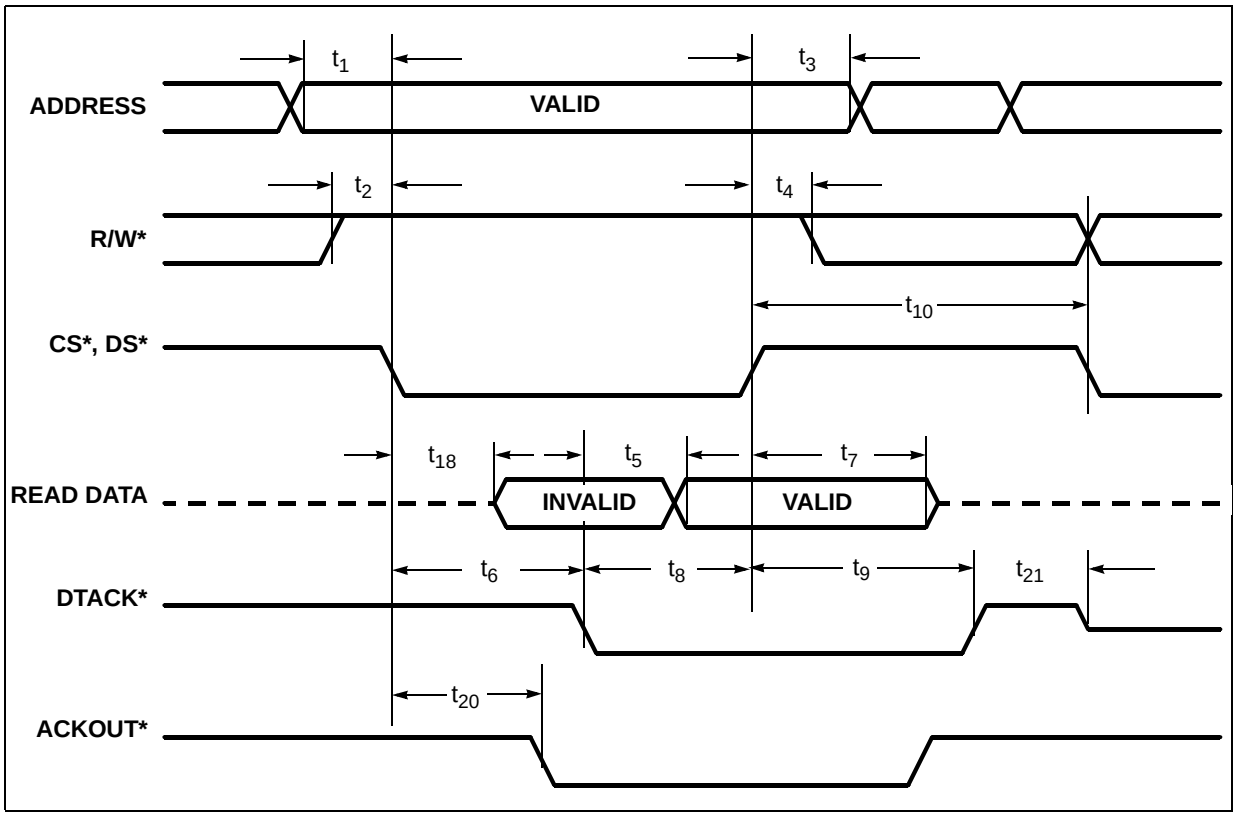


Figure 38. Unclocked Bus Interface Service Acknowledgment Cycle, Motorola®-Style Handshake

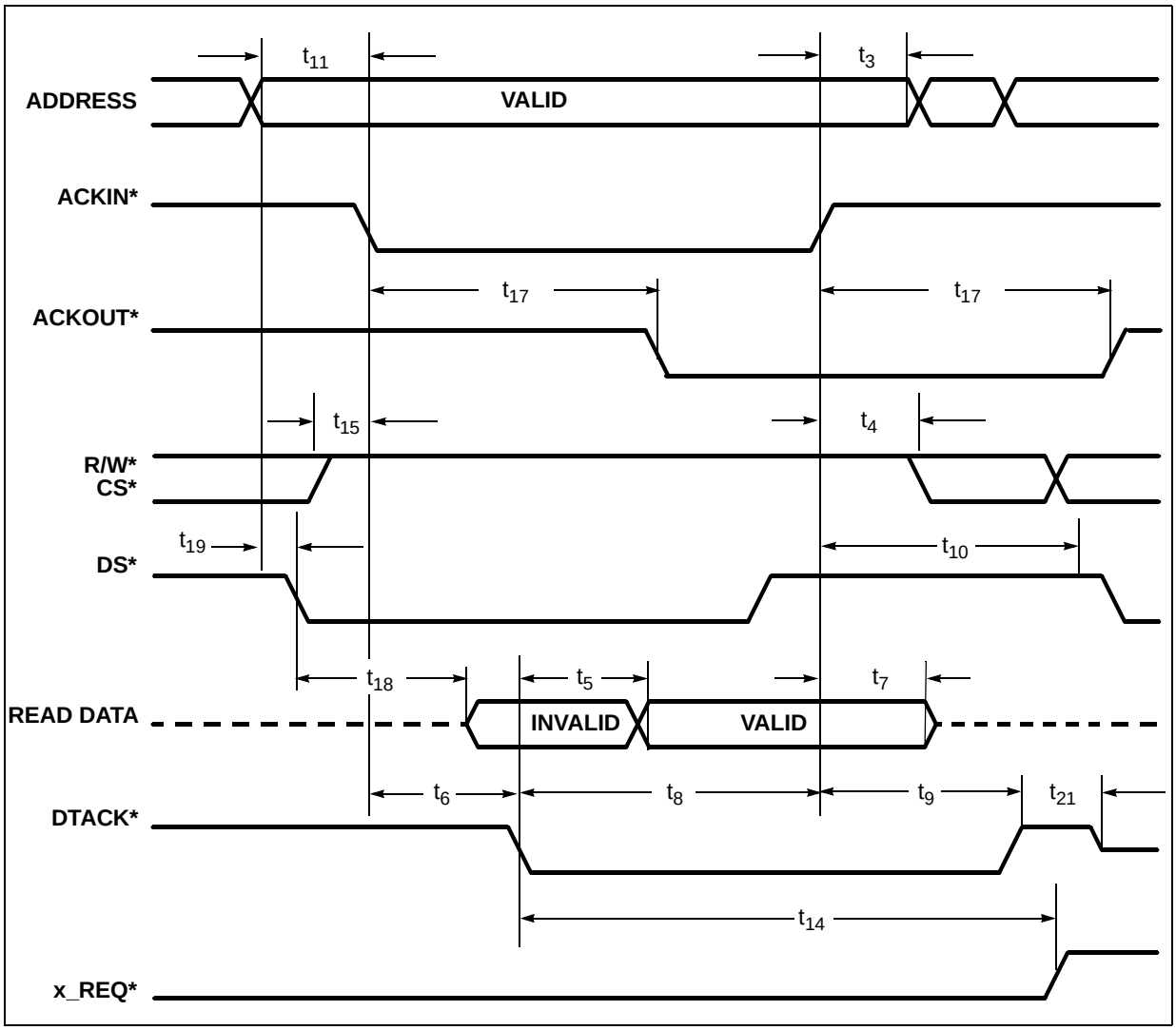


Figure 39. Unclocked Bus Interface Write Cycle, Motorola®-Style Handshake

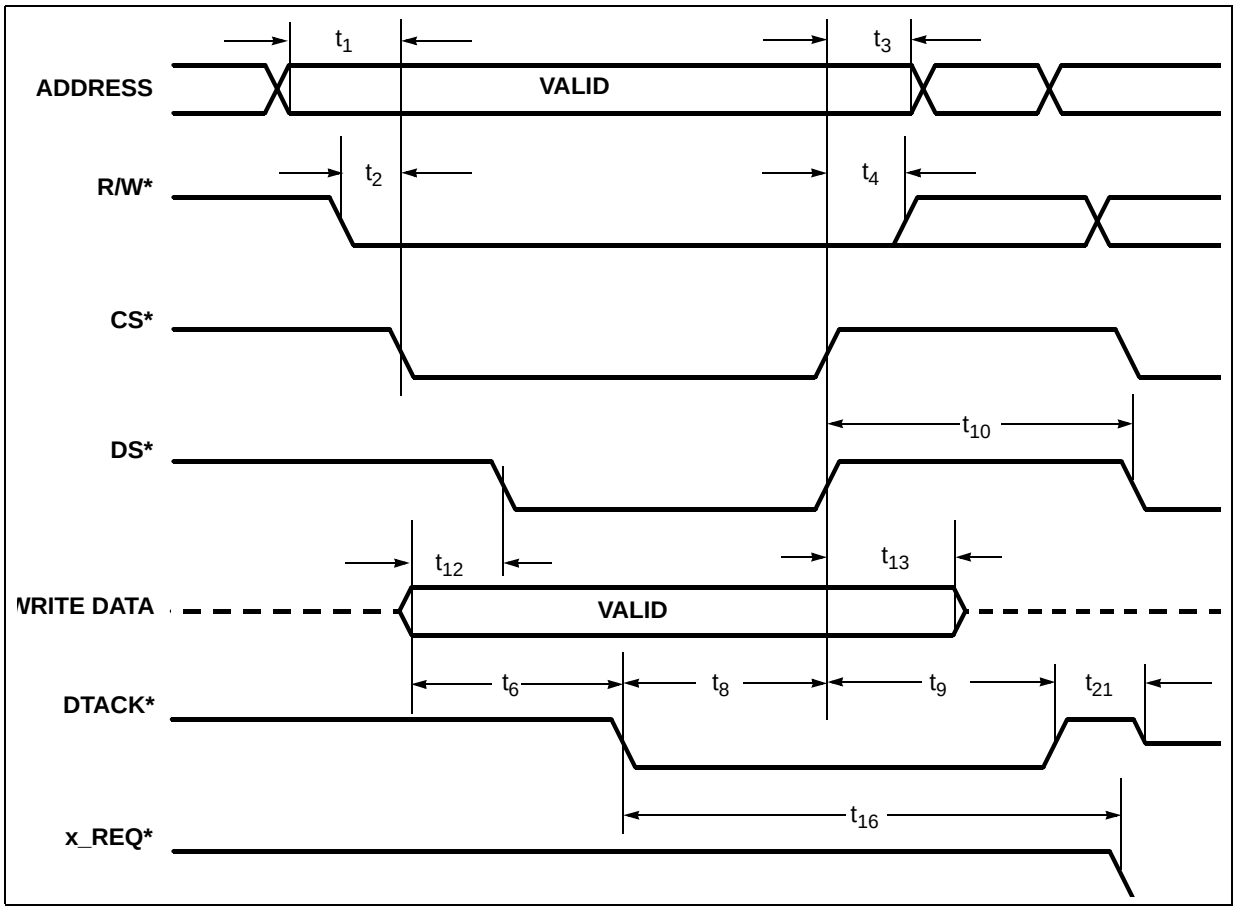


Figure 40. Unclocked Bus Interface Read Cycle, Intel®-Style Handshake

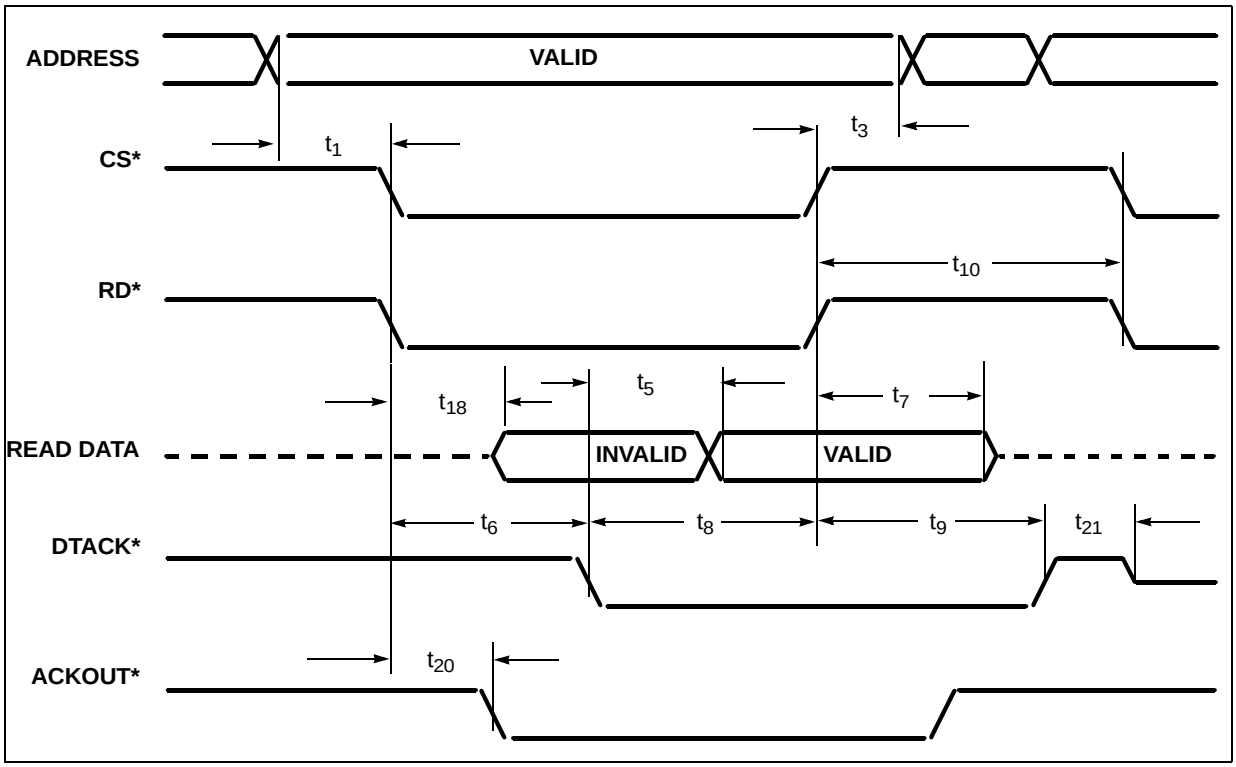


Figure 41. Unclocked Bus Interface Service Acknowledgment Cycle, Intel®-Style Handshake

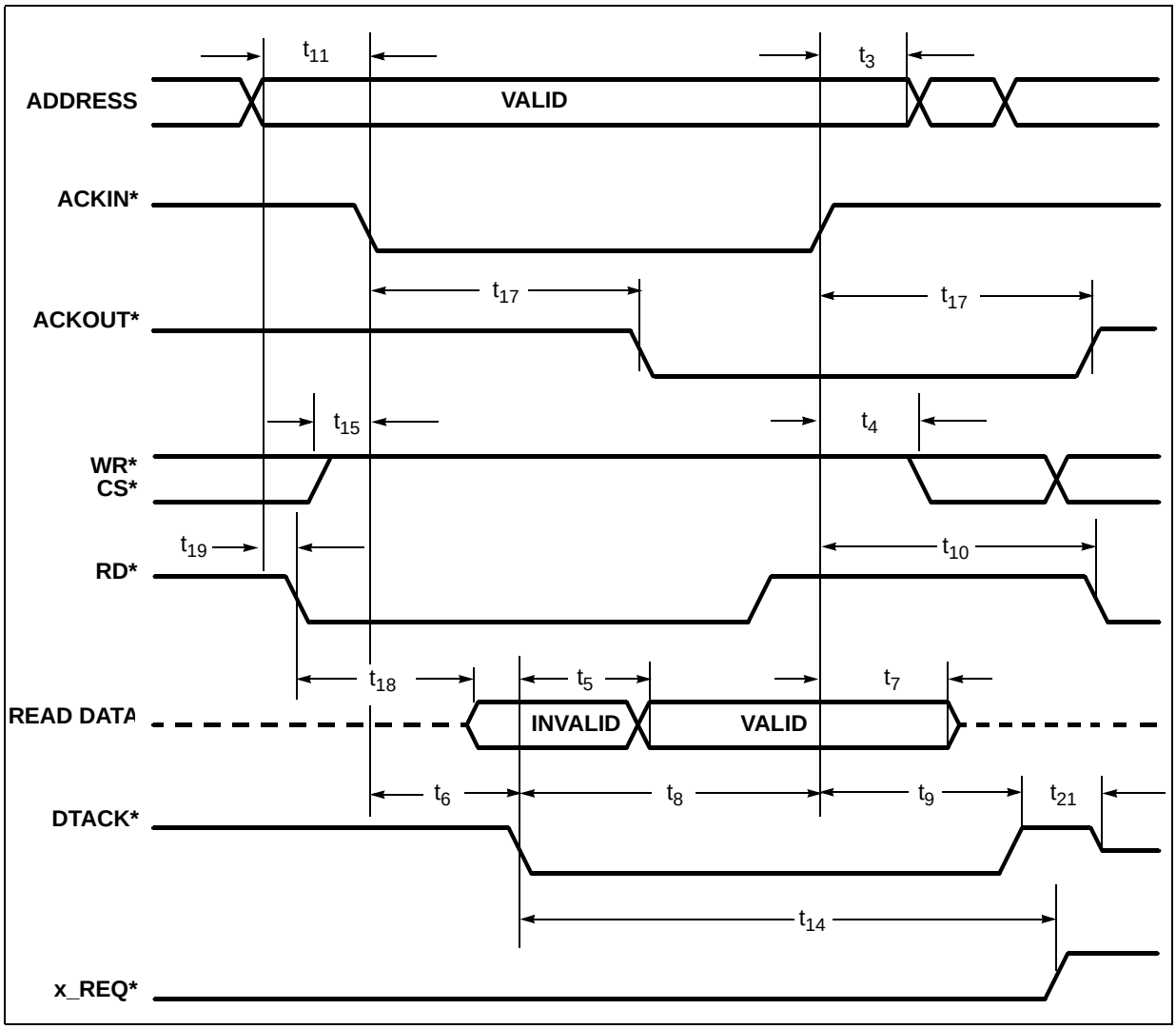
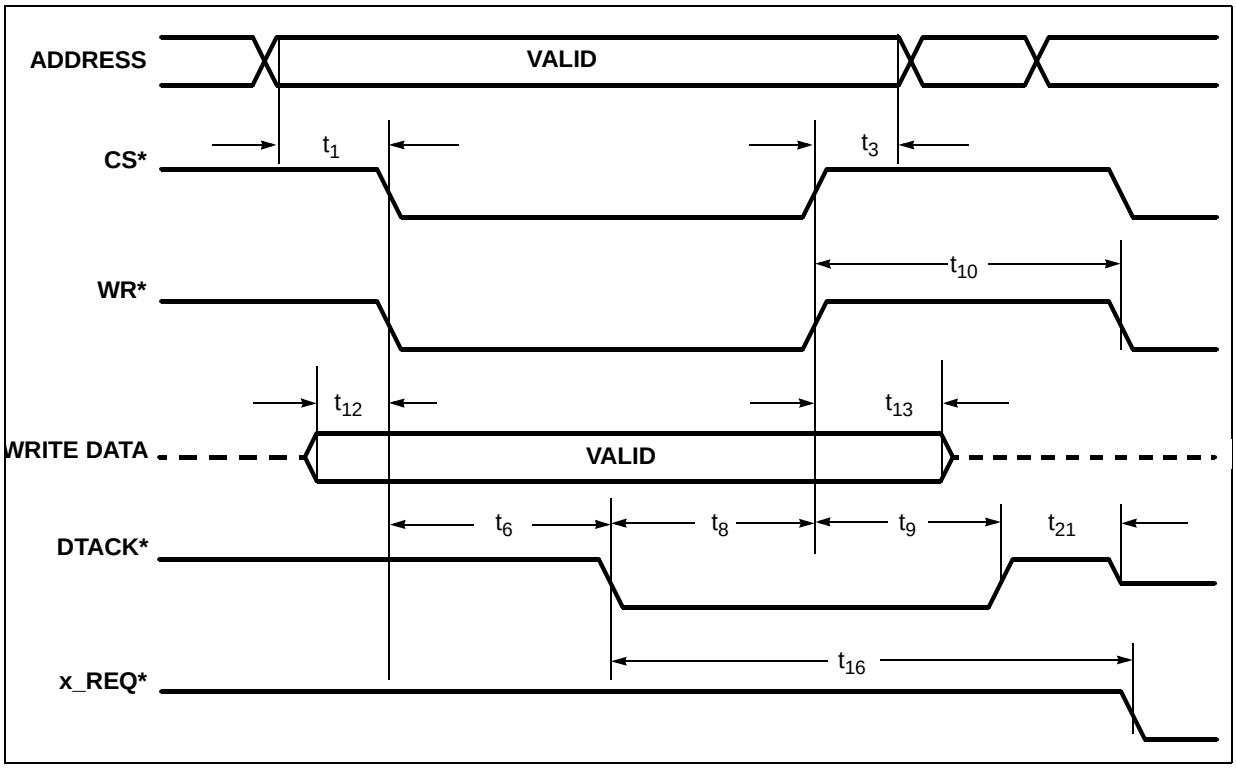
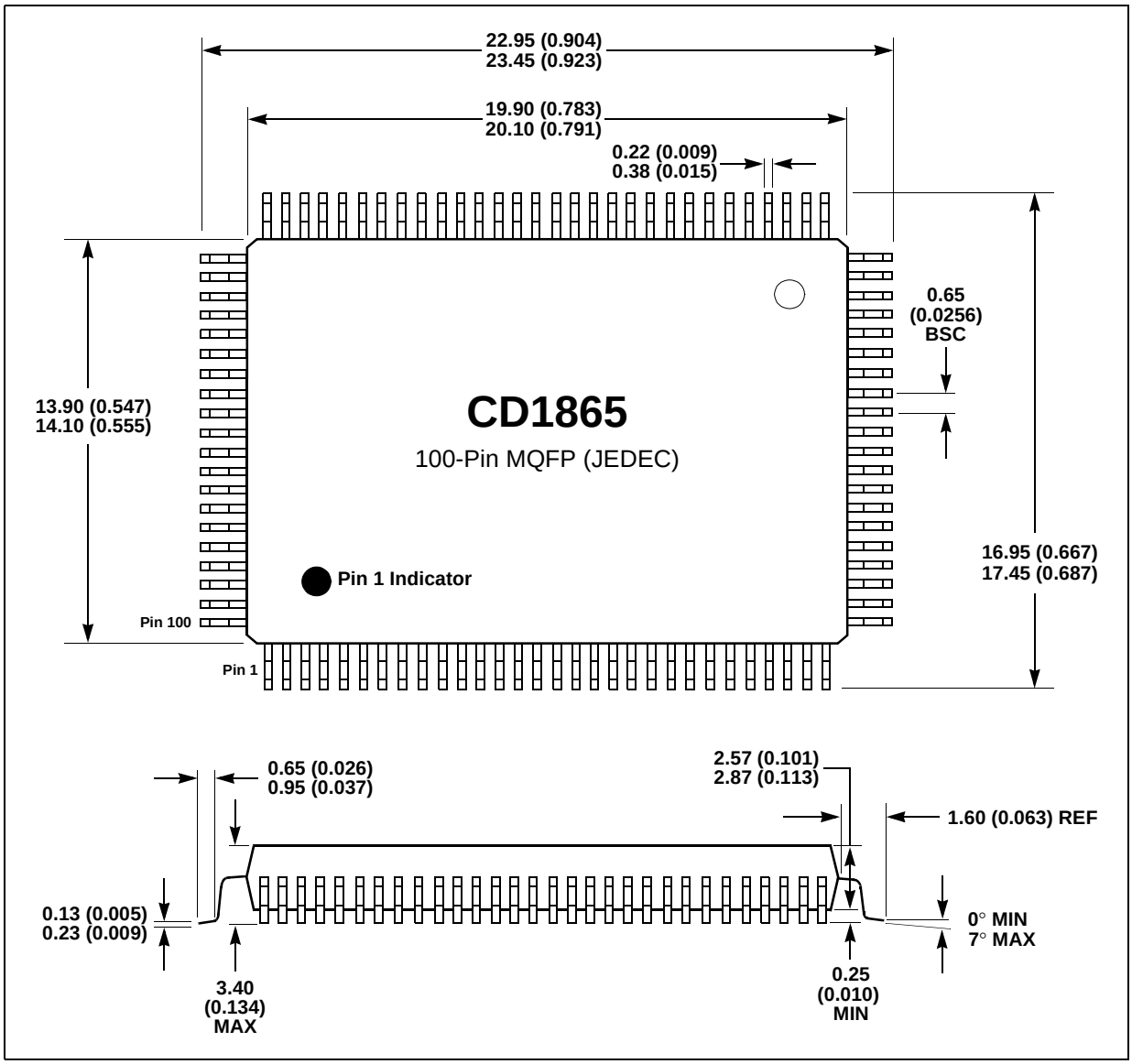


Figure 42. Unclocked Bus Interface Write Cycle, Intel®-Style Handshake



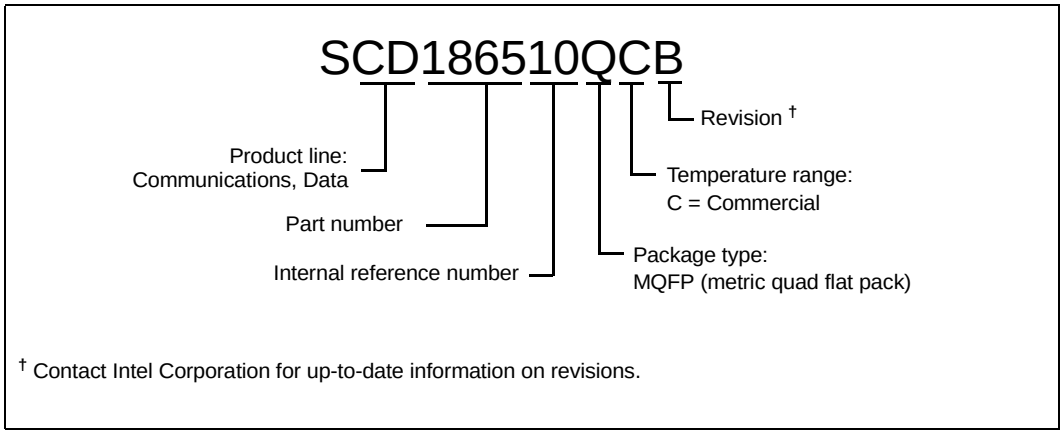
11.0 Package Specifications



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.

12.0 Ordering Information

The order number for the -pin device is:



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

Index

L

M

O

P

Q

R

S

T

U

V