# A Performance Comparison of the 85C30 and the CL-CD2400/2401

**Data Communications Products Group**
**Cirrus Logic, Inc.**

**Scope**
This report presents a performance evaluation of the Cirrus Logic, Inc. CL-CD2400/CD2401 four-channel, multi-protocol communications controller as compared to the 85C30. It is the product of an ongoing test program at Cirrus Logic.

**Related Documents**
— CL-CD2400/2401 Data Sheet

## 1. OVERVIEW OF THE CL-CD2400 AND CL-CD2401

The CL-CD2400 and CL-CD2401 are four-channel sync/async communications controllers designed specifically to reduce host-system processing overhead and increase efficiency in all types of communications applications. They have four fully independent serial channels that support all popular asynchronous, character-synchronous, and bit-synchronous protocols at rates up to 64 kbps. Both devices are based on a proprietary on-chip RISC processor that performs all time-critical, low-level tasks, which would otherwise be performed by the host system.

The CL-CD2400 and CL-CD2401 boost system efficiency with on-chip DMA, on-chip FIFOs, intelligent vectored interrupts, including Good Data™ Interrupts, and intelligent protocol processing. The on-chip DMA controller provides 'Fire and Forget' transmit support — the host only needs to inform the CL-CD2400 or CL-CD2401 of the location of the packet to be sent. Similarly, on receive, the CL-CD2400 or CL-CD2401 will automatically receive a complete packet with no host intervention or assistance required. The DMA controller also has an 'Append Mode' for use in asynchronous applications.

The DMA controller uses a dual-buffer scheme that makes it easy to implement simple or complex buffer schemes, such as linked lists and circular queues. Each channel and direction has two active buffers; the device transfers into or out of one while the host updates the other. The CL-CD2400/2401 then interrupts the host at the completion of a frame or a buffer. In applications where buffers are of a small fixed size, the dual buffering scheme allows large frames to be chained across multiple buffers.

For applications where DMA interface is not desired, the part may be operated as an interrupt-driven or polled device. This choice is available individually for each channel and each direction — for example, you can have DMA transmit and interrupt-driven receive.

In either case, 16-byte FIFOs on each channel and in each direction reduce latency time requirements and make both software and hardware design less time-critical. Threshold levels on the FIFOs are user-programmable.

Efficient vectored interrupts are another way the CL-CD2400/2401 helps system efficiency. Separate interrupts are generated for transmit, receive, and modem-signal-change, with unique user-defined vectors for each type and channel. This allows very flexible interfacing and fast, efficient interrupt code. For example, the Good Data Interrupt allows the host to vector directly to a routine that transfers the data — no status or error-checking is required.

The CL-CD2400 and CL-CD2401 are functionally equivalent and differ only in their pinout. The CL-CD2400 is packaged in an 84-pin PLCC, and offers five clock/modem pins per channel; the CL-CD2401 is packaged in a 100-pin QFP, and offers eight clock/modem pins per channel.

## 2. CL-CD2400/2401 AND 85C30 FEATURES COMPARED

The CL-CD2400/2401 combines the functions of two 85C30s and one DMA controller. Since different types of DMA controllers can be used with the 85C30, it is not possible to compare the relative merits of the CL-CD2400/2401 DMA with the 85C30; however, some assumptions can be made. For instance, no DMA controller can transfer data faster than the 85C30 allows.

The following table compares the features and capabilities of CL-CD2400/2401 and the 85C30.

| **GENERAL** | **CL-CD2400/2401** | **85C30** |
|---|---|---|
| Number of serial channels per chip | 4 | 2 |
| Number of DMA channels per chip | 8 | 0 |
| Number of timers per channel | 2 | 0 |
| Transmit FIFO size, per channel | 16 | 1 |
| Receive FIFO size, per channel | 16 | 3 |

| **PROTOCOLS** | **CL-CD2400/2401** | **85C30** |
|---|---|---|
| Asynchronous | YES | YES |
| HDLC | YES | YES |
| SDLC | YES | YES |
| Bisync | YES | YES |
| X.21 | YES | NO |

| **SYSTEM BUS** | **CL-CD2400/2401** | **85C30** |
|---|---|---|
| Data bus width | 8 or 16 bits | 8 bits only |
| Big-endian/little-endian addressing | pin-selectable | not available |
| Maximum clock frequency | 20 MHz | 16 MHz |
| Number of Interrupt Types | 3 | 1 |
| Ready/Wait signal | YES | NO (note 1) |
| Interrupt daisy chain | YES | YES |
| Read/Write access time | 140 ns | 6 clock cycles (note 2) |
| DMA transfer bandwidth | 10 Mbytes/sec. (200 ns) | 2.66 Mbytes/sec. (375 ns) (note 3) |
| Package options | 84-pin PLCC and 100-pin QFP | 40-pin DIP and 44-pin PLCC |

| **SERIAL CHANNELS** | **CL-CD2400/2401** | **85C30** |
|---|---|---|
| Number of modem lines/channel | 5 | 3 (note 4) |
| Number of clock lines/channel | 3 (note 5) | 2 |
| Number of DPLLs per channel | 1 | 1 |
| Types of clock encoding supported | NRZ, NRZI, Manchester | NRZ, NRZI, FM0/1 |
| Number of BRGs per channel | 2 | 1 |

**NOTES:**

1) As it is muxed with its DMA request signal, the 85C30 wait signal is not available in DMA systems.
2) Two clock cycles are required for access, and four more are required for 'recovery'. Recovery time is required after any type of access to the 85C30.
3) The DMA bandwidth depends on the slower of the two, either the 85C30 or the external DMA controller. The limits of 85C30 have been shown; it is generally lower than the external DMA device. Bus arbitration time has not been included for two reasons: first, it is system-dependent, not peripheral-dependent; and second, it is likely to be the same regardless of the communications chip chosen.
4) Reflects pin count when full duplex DMA request is implemented. Some applications have four modem signals available, depending on how DMA and wait signals are implemented.
5) The CL-CD2400 multiplexes five modem and three clock signals onto five pins. The CL-CD2401 provides eight independent pins to support these functions. This is the only difference between the CL-CD2400 and CL-CD2401.

## 3. COMPARISON OF PERFORMANCE

### 3.1 The 85C30

The 85C30 imposes significant limitations on system design and throughput. It requires immediate, high-priority response by the CPU, both for interrupt service and for DMA access to the system bus. Its 3-byte-deep receive FIFO (without programmable threshold), and 1-byte-deep transmit FIFO are too small to provide adequate latency between the host and the serial lines. As a result, a dedicated high-performance CPU is needed to service the 85C30 in order to prevent overrun and underrun due to overflowing and underflowing of the FIFOs.

Adding a DMA controller to the system (or integrating a DMA cell to the 85C30), will not provide significant gains of performance to the system because of the shallow FIFOs. Multiple short bursts create repetitive overhead in bus arbitration, and slow CPU performance by tying up the bus with data transfers.

Finally, the intrinsic 'recovery time' required for every access cycle to the 85C30 increases time for data movement, with or without DMA. (Recovery time is the time needed by the 85C30 after any access, for internal circuits to stabilize. The 85C30 requires four idle clocks between each access for internal synchronization of system interface and instruction execution). In typical real-world applications, the host CPU frequently has to access the 85C30 for control and status exchange in addition to data transfers.

In asynchronous applications, the CPU has to poll the receive character available bit after each character read, even though all three bytes of FIFO are full. Parity and status checking are also needed because the character may not be Good Data.

In HDLC transmit applications using DMA, the CPU still has to write the first and the last character separately, in order to properly initiate and terminate the DMA transfers. (In some cases, the CPU may have to transfer the first two bytes — refer to Figure 1 for details.) In addition, the CPU must set the Abort/Flag Bit to 'abort' during every frame, then as the last byte is being transferred, set the bit to 'flag'. Because there is only a 1-byte-deep transmit FIFO, DMA latency must be kept very short, and the CPU has to monitor HDLC transmit very closely to prevent underrun.

In Bisync applications, the CPU has to monitor character availability closely in order to enable/disable CRC checking/generation in real time. Sending and receiving 'small' frames in HDLC, and Bisync can consume a high percentage of CPU and system bus time, due to the amount of time-critical software overhead per packet.

### 3.2 The CL-CD2400 and CL-CD2401

The CL-CD2400 and CL-CD2401 offer less burden on the host CPU. There are no time-critical interrupt or DMA latency requirements, and as the DMA is integrated with the serial interface, no CPU activity is required during the course of transmission of a packet.

In asynchronous applications, the 16-byte FIFOs eliminate time-critical interrupts — the threshold for interrupts can be set to any level — so the time allowed for the host to respond to the interrupt can be as long as needed. DMA can also be used to transfer asynchronous data. Again, the 16-byte FIFOs virtually eliminate the need for time-critical access to the bus by the DMA controller. Likewise, the large FIFO means fewer DMA bursts, and less bus arbitration overhead. The Figures 1 and 2 show the steps required to receive a string of 12 async characters.

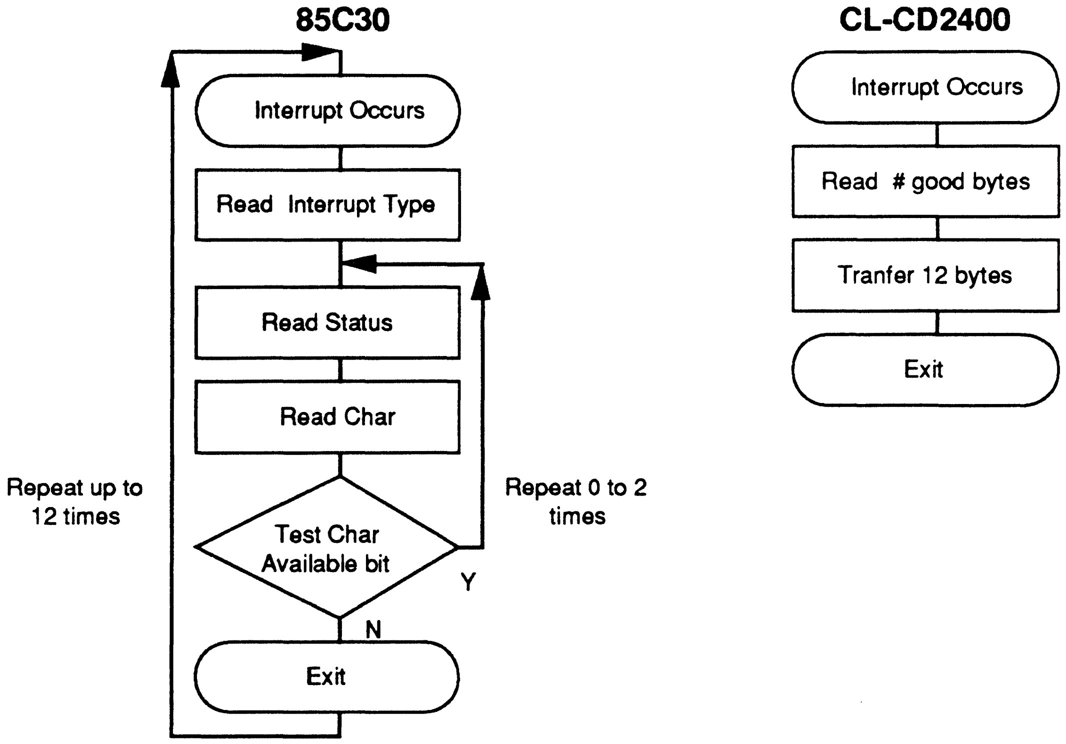

**Figure 1. 85C30 and CL-CD2400 Flow Charts — required to receive a string of 12 async characters**

Note that 85C30 has only a three-byte FIFO on receive (with no threshold feature), so the host system will have to handle at least four or as many as 12 interrupts.

In synchronous applications, the CL-CD2400 offers 'Fire-and-Forget' operation. The host can pass an address pointer and length to the CL-CD2400, and an entire frame will be sent with no further effort. Similarly, the CL-CD2400 will receive complete frames without any host intervention. The 85C30, on the other hand, requires substantially more host CPU support. There are two time-critical portions of code involved in starting and ending the frame transmission, as well as several additional steps overall. The two flowcharts on the following page show the steps required to transmit a 50-byte frame in HDLC.

Refer to the CL-CD2400/2401 Data Sheet for further information.

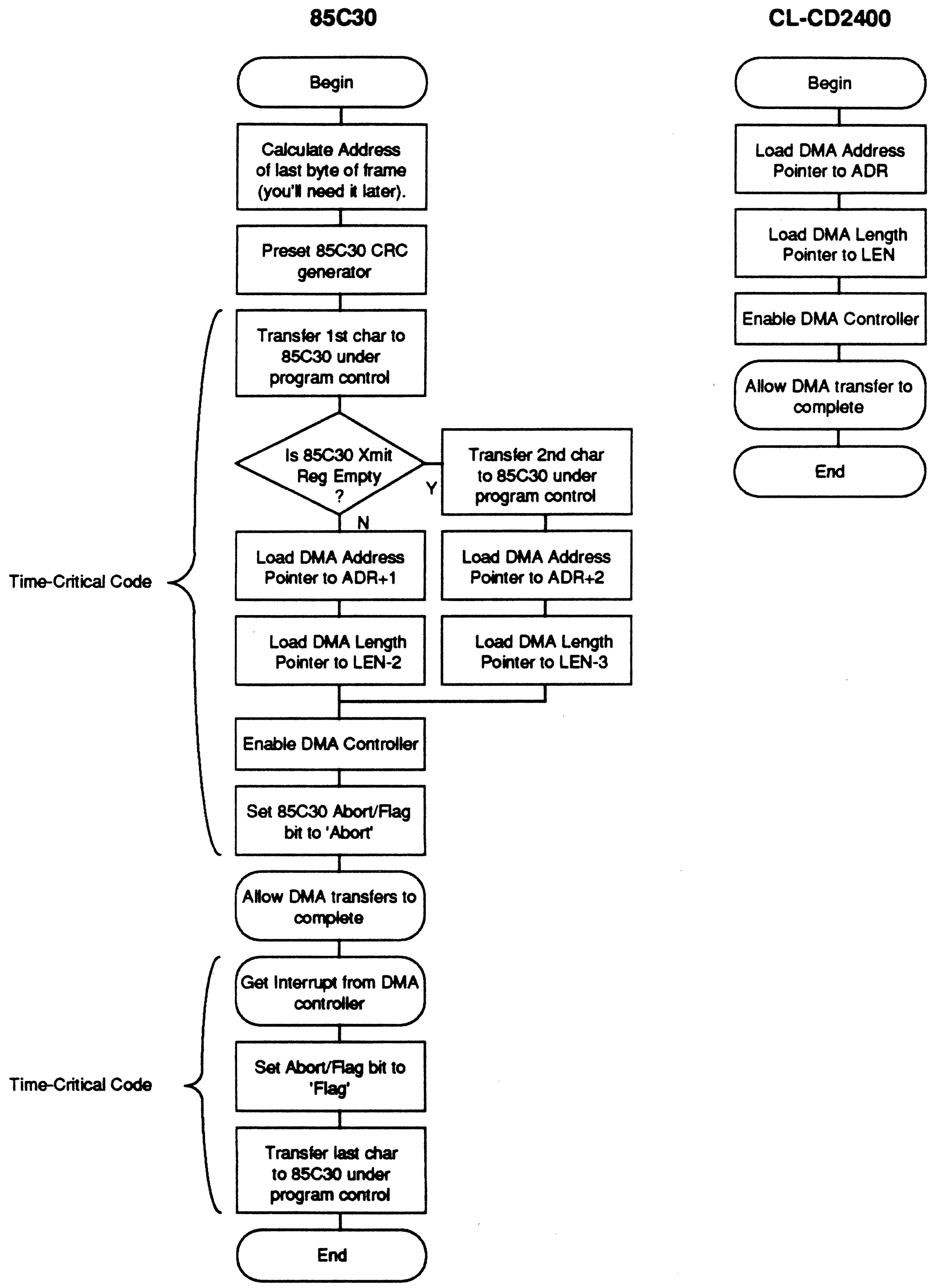
85C30
CL-CD2400
Begin
Calculate Address of last byte of frame (you'll need it later).
Preset 85C30 CRC generator
Transfer 1st char to 85C30 under program control
Is 85C30 Xmit Reg Empty ?
Y
N
Transfer 2nd char to 85C30 under program control
Load DMA Address Pointer to ADR+1
Load DMA Address Pointer to ADR+2
Load DMA Length Pointer to LEN-2
Load DMA Length Pointer to LEN-3
Enable DMA Controller
Set 85C30 Abort/Flag bit to 'Abort'
Allow DMA transfers to complete
Get Interrupt from DMA controller
Set Abort/Flag bit to 'Flag'
Transfer last char to 85C30 under program control
End
Time-Critical Code
Time-Critical Code
Begin
Load DMA Address Pointer to ADR
Load DMA Length Pointer to LEN
Enable DMA Controller
Allow DMA transfer to complete
End

**Figure 2. 85C30 and CL-CD2400 Flow Charts — required to transmit a 50-byte frame in HDLC**

## 3.3 Interrupt and DMA Latency Comparison

There are two kinds of latencies to consider when evaluating system performance, DMA Bus-access latency and host CPU interrupt repsonse latency. The CD2400/2401 is substantially faster in both of these categories than the 85C30.

| DMA Latency | CL-CD2400/2401 | 85C30 |
|---|---|---|
| Transmit | 2000 μsec | 125 μsec |
| Receive | 2000 μsec | 125 μsec |

| Interrupt Latency - Sync. Protocols | CL-CD2400/2401 | 85C30 |
|---|---|---|
| Transmit | infinite | 125 μsec |
| Receive | 2000 μsec | 375 μsec |

| Interrupt Latecny - Async. Protocol | CL-CD2400/2401 | 85C30 |
|---|---|---|
| Transmit | 2000 μsec (Note 1) | 125 μsec (Note 1) |
| Receive | 2000 μsec | 375 μsec |

**Note 1:** Maximum response time to avoid underrun. An underrun is not a fatal error, so violating this specification will not cause a problem; however, it will reduce system efficiency.

## 3.4 Performance Comparison

The HDLC family of protocols are the most widely used of the synchronous protocols, so we've begun with several examples showing the advantages of the CD2400/2401 over the 85C30 in HDLC applications. Comparison of performance in an interrupt or polled system is presented in the first two examples, since the 85C30 has no DMA controller on-chip.

Without knowing the DMA controller being used with the 85C30, we cannot quantify precisiely the performance improvement the CD2400 offers over the 85C30 solution, but we can make some reasonable estimates. These are shown in the third example.

Example 4 shows the loading effect on a host CPU of interrupts, and quantifies the benefits of using a datacom controller which reduces the number of interrupts the host must handle.

Example 5 shows the performance improvement in typical asynchronous applications.

In this example, and the ones that follow, we have not included the initialization code execution time. Only times based on per-packet processing are used because power-on initialization execution time is not usually a critical parameter.

***Example 1: Receiving HDLC Frame in Interrupt Mode***

The times shown here are the actual amount of time spent accessing the peripheral chip. This analysis does not include the time required by the host to set pointers, or do other work associated with handling the data once it has been transferred. Instruction execution time has not been included, because there are so many different CPUs of such a wide range of performance that it is not possible to make any general assumptions. In gneral, the 85C30 requires more instructions to receive a frame than required by the CD2400/2401. This means in the 'real world' the performance gain will be even greater.

| **85C30 Rx Data Mechanism** | |
|---|---|
| Interrupt Acknowledge Cycle | 160 ns* |
| Recovery Time | 250 ns |
| Read RR8 (for 1 char.) | 125 ns |
| Recovery Time | 250 ns |
| Reset Highest IUS | 125 ns |
| Recovery Time | 250 ns |
| **Total** | **1160 ns** |

| **CL-CD2400 Rx Data Mechanism** | |
|---|---|
| Interrupt Acknowledge Cycle | 200 ns* |
| Read RFOC (# of bytes in FIFO ) | 200 ns |
| Read RDR (Ready chars.) | 200 ns times N |
| Write to REOIR | 200 ns |
| **Total** | **(600 + 200N) ns** |
| where | N = 1 to 16 |

* CPU context switch time is not included in the calculation. Note that relative advantage of the CL-CD2400 performance will increase if context switch time is considered, because of the significantly fewer number of interrupts it generates. Refer to Example 3 on interrupt overhead for comparisons of this effect.

**Calculation of Performance**

| **Number of Bytes** | **85C30 Access Time** | **CL-CD2400 Access Time** | **Improvement Factor** |
|---|---|---|---|
| 8 | 1160 x **8** = 9.28 µs | 600 + 200 x **8** = 2.2 µs | **4.2** |
| 100 | 1160 x **100** = 116 µs | **6** (600 + 200 x **16**) + (600 + 200 x **4**) = 24.2 µs | **4.8** |
| 1000 | 1160 x **1000** = 1160 µs | **62** (600 + 200 x **16**) + (600 + 200 x **8**) = 238 µs | **4.9** |

### *Example 2: Transmitting HDLC Frame in Interrupt Mode*

The times shown here are the actual amount of time spent accessing the peripheral chip. This analysis does not include the time required by the host to set pointers or do other work associated with handling the data once it has been transferred. Instruction execution time has not been included, because there are so many different CPUs of such a wide range of performance that it is not possible to make any general assumptions. In general, the CD2400/2401 requires fewer instructions that the 85C30, so the performance gain will be even greater than indicated here.

| **85C30 Tx Data Mechanism** | |
|---|---|
| Interrupt Acknowledge Cycle | 160 ns* |
| Recovery Time | 250 ns |
| Write WR8 (for 1 char.) | 125 ns |
| Recovery Time | 250 ns |
| Reset Highest IUS | 125 ns |
| Recovery Time | 250 ns |
| **Total** | **1160 ns** |

| **CL-CD2400 Tx Data Mechanism** | |
|---|---|
| Interrupt Acknowledge Cycle | 200 ns* |
| Write TDR (Write N chars.) | 200 ns times N |
| Write to TEIOR | 200 ns |
| **Total** | **(400 + 200N) ns** |
| where | N = 1 to 16 |

* CPU context switch time is not included in the specification. The time required by the processor to enter its interrupt context, and to save necessary registers is not included. The times shown are only the time required to fetch necessary status from the device. Note that the CL-CD2400 performance advantage will increase if context switch time is considered.

**Calculation of Performance (not including interrupt context switch time):**

| **Number of Bytes** | **85C30 Access Time** | **CL-CD2400 Access Time** | **Improvement Factor** |
|---|---|---|---|
| 8 | 1160 x **8** = 9.28 µs | 400 + (200 x **8**) = 2.0 µs | **4.64** |
| 100 | 1160 x **100** = 116 µs | **6** (400 + (200 x **16**)) + (400 + 200 x **4**) = 22.8 µs | **5.08** |
| 1000 | 1160 x **1000** = 1160 µs | **62** (400 + (200 x **16**)) + (400 + 200 x **8**) = 225 µs | **5.15** |

### *Example 3: Transmitting HDLC in DMA Mode*

It is not possible to provide an exact comparison between the CL-CD2400 and the 85C30 when using an external DMA controller because of the wide variety of DMA controllers currently available. However, some general comparisons and observations can be made.

Both the CL-CD2400 and the 85C30/DMA require the following:

1. Write a start-of-frame pointer to the DMA registers.
2. Write a length-of-frame pointer to the DMA registers.
3. Enable the DMA to transfer data.

In addition, the 85C30/DMA combination requires a start of the transfer manually by moving one byte[1]:

1. Modify the start-of-frame pointer by adding 1 to it before writing to the DMA.
2. Modify the length-of frame by subtracting 2 from it.
3. Calculate the pointer to the last byte in the frame and have it ready.
4. Manually reset the CRC generator.
5. Transfer the first byte under program control from RAM to the 85C30. This 'primes the pump', causing the 85C30 to then request more data from the DMA controller.
6. Enable the DMA controller.
7. *Immediately* set the 85C30 Flag/Abort Bit to 'abort', so that if the DMA controller underruns the frame, it will be aborted (as it should be in that case).
8. Wait for an interrupt from the DMA controller, indicating it is done with the transfer.
9. *Immediately* set the 85C30 Flag/Abort Bit back to 'flag'.
10. *Immediately* move the last character from RAM to the 85C30.

There are six time-critical steps involved in starting the transmission of a frame. Three of these are accesses to the 85C30, and three are accesses to the DMA controller. In addition to these six accesses, a number of host CPU cycles will be required to fetch pointers, etc. There are three time-critical steps involved in ending the transmission of a frame, including receiving an interrupt and entering an interrupt context.

Together, there are nine I/O accesses, one interrupt context-switch time, and several instructions that must be executed. At both the beginning and the end of the frame one byte time is available to perform these time-critical operations (to avoid short frames at the beginning and aborted frames at the end). A typical 56 kbps synchronous line allows 143 μs per byte; at 64 kbps this drops to 125 μs. The problem is then whether or not the code can execute in the available time.

The answer is 'yes' for a single channel system. However, as the number of channels increases, it becomes increasingly difficult for one host CPU (even a fast one) to keep pace. A design based on a 16 MHz 68000 processor, for instance, needs about 10 μs for the interrupt context switch alone. A four-channel system

[1] Two bytes may need to be moved, depending on the relative speed of the DMA controller being used and the 85C30. When the first byte is moved to the 85C30 transmit holding register, the device moves it into the shift register, begins transmission, and requests another byte. In systems with sufficiently fast DMA controllers, the controller will supply the byte in time. Some systems may underrun, however, which causes an illegal short frame or aborted frame. The flowchart provided in Figure 2 shows the case where it is necessary to test the 85C30 transmitter to see if a second byte is needed; this case is more complex than the one outlined above in Example 5. Most real-world applications pre-calculate all of the pointers, because once the frame is started there is not much time for address calculation.

therefore has only 85 µs to handle pointer manipulation and all other work for four channels - very difficult, and this still does not include the receive-side code! Systems with more channels require much faster and more expensive host processors.

The CL-CD2400, on the other hand, has more relaxed time-critical code segments, as well as a significantly reduced host CPU effort required per packet. Only three steps are required to send a packet, so only a few simple host instructions are necessary. None of these are time-critical, so the host response can be as slow as necessary. On the receive side, the host need only respond within 2000 µs - easy even in multi-channel designs. Typical low-cost CISC processors can support many channels. Furthermore, in high-channel count systems, there is a fail-safe, self-limiting characteristic: an overly-busy system will simply have more flags between packets, instead of aborting frames and wasting bandwidth.

Given the number of bus cycles used in accessing the peripheral chip(s) in each of the two cases, it is possible to discern the amount of bus time used in each case. Assumed bus speed is 10 MHz, and both DMA controllers can do a cycle in two clocks, and an assumed packet is 50 bytes.

For the 85C30, the 10 steps shown in this example will require 28.3 µs of bus cycles to access the 85C30 and allow the DMA controller to do its transfers. This does NOT include the actual CPU instruction execution time or interrupt overhead.

For the CL-CD2400, the three steps in this example and DMA bus time will total 5.4 µs. Again, this does not include the actual CPU instructions or interrupt context overhead, but as noted above, it is substantially less with the CL-CD2400 than with the 85C30. According to this data, the CL-CD2400 is at least six times more efficient than the 85C30, and depending on the host system CPU, may be more efficient than that.

***Example 4: Effect of Interrupt Context Switch Time on Relative Performance***

Interrupt response time — the time required by the host CPU to switch contexts — depends on the processor and the number of registers that must be saved. Processors differ, so it is not possible to determine an absolute figure for the amount of overhead. Some estimates, however, can be made. The 68000, for instance, requires 38 clock cycles to save internal state, and 48 more clock cycles to save three address and two data registers. Restoring the registers and context at 16.67 MHz requires 52 and 24 clocks, respectively, and this requires 9.72 µs. This value is typical for the kinds of processors most often used in intelligent I/O systems. (Although it should be noted that RISC processors will generally be shorter; alternately, more complex software environments will result in the time being much longer.) The following two tables show the number of interrupts, and their effect on the host processor.

**Number of Interrupts:**

| Number of Bytes | # of 85C30 Interrupts | # of CL-CD2400 Interrupts | Improvement Factor |
|---|---|---|---|
| 8 | 8 | 1 | 8 |
| 100 | 100 | 7 | 14.3 |
| 1000 | 1000 | 63 | 15.8 |

Net Performance, 68000 @ 16.67 MHz.

| Number of Bytes | 85C30 Execution Time | CL-CD2400 Execution Time | Improvement Factor |
|---|---|---|---|
| 8 | 9.28 + 8 x 9.72 = 87 µs | 2 + 1 x 9.72 = 11.7 µs | 7.43 |
| 100 | 116 + 100 x 9.72 = 1204 µs | 22.8 + 7 x 9.72 = 90.8 µs | 13.3 |
| 1000 | 1160 + 1000 x 9.72 = 10880 µs | 225 + 63 x 9.72 = 837 µs | 13.0 |

***Example 5: Receiving Asynchronous Data in Interrupt Mode***

The times shown here are the actual amount of time spent accessing the peripheral chip. This analysis does not include the time required by the host to set pointers, or do other work associated with handling the data once it has been transferred. Instruction execution time has not been included because there are so many different CPUs of such a wide range of performance that it is not possible to make any general assumptions.

| **85C30 Rx Data Mechanism** | |
|---|---|
| Interrupt Acknowledge Cycle | 160 ns* |
| Recovery Time | 250 ns |
| Read Status Byte | 125 ns |
| Recovery Time | 250 ns |
| Test Status Byte | 125 ns |
| Read Rcv Data Reg (for 1 char.) | 125 ns |
| Recovery Time | 250 ns |
| Reset Highest IUS | 125 ns |
| Recovery Time | 250 ns |
| **Total** | **1660 ns** |

| **CL-CD2400 Rx Data Mechanism** | |
|---|---|
| Interrupt Acknowledge Cycle | 200 ns* |
| Read RFOC (# of bytes in FIFO ) | 200 ns |
| Read RDR (Ready chars.) | 200 ns times N |
| Write to REOIR | 200 ns |
| **Total** | **(600 + 200N) ns** |
| where | N = 1 to 16 |

**Calculation of Performance**

| **Number of Bytes** | **85C30 Access Time** | **CL-CD2400 Access Time** | **Improvement Factor** |
|---|---|---|---|
| 8 | 1660 x **8** = 13.28 μs | 600 + 200 x **8** = 2.2 μs | **6.0** |
| 100 | 1660 x **100** = 166 μs | **6** (600 + 200 x **16**) + (600 + 200 x **4** ) = 24.2 μs | **6.8** |
| 1000 | 1660 x **1000** = 1660 μs | **62** (600 + 200 x **16**) + (600 + 200 x **8**) = 238 μs | **7.0** |

## SUMMARY

In all applications, the CL-CD2400 offers a number of advantages over the 85C30. It is easier to program, requires less host CPU time, and less host bus bandwidth to support a given level of traffic. It has no time-critical interrupt requirements, and has very low bus latency and interrupt latency requirements. The CL-CD2400/2401 offer the most functionality and high level of integration available today — four full channels, FIFOs, and DMA in one 84- or (smaller) 100-pin package.

CIRRUS LOGIC

## Direct Sales Offices

### *Domestic*

**N. CALIFORNIA**
San Jose
TEL: 408/436-7110
FAX: 408/437-8960

**S. CALIFORNIA**
Tustin
TEL: 714/258-8303
FAX: 714/258-8307

Thousand Oaks
TEL: 805/371-5381
FAX: 805/371-5382

**ROCKY MOUNTAIN AREA**
Boulder, CO
TEL: 303/939-9739
FAX: 303/440-5712

**SOUTH CENTRAL AREA**
Austin, TX
TEL: 512/794-8490
FAX: 512/794-8069

**NORTHEASTERN AREA**
Andover, MA
TEL: 508/474-9300
FAX: 508/474-9149

Philadelphia, PA
TEL: 215/251-6881
FAX: 215/651-0147

**SOUTH EASTERN AREA**
Boca Raton, FL
TEL: 407/994-9883
FAX: 407/994-9887

Atlanta, GA
TEL: 404/263-7601
FAX: 404/729-6942

### *International*

**GERMANY**
Herrsching
TEL: 49/08152-2030
FAX: 49/08152-6211

**JAPAN**
Tokyo
TEL: 81/3-5389-5300
FAX: 81/3-5389-5540

**SINGAPORE**
TEL: 65/3532122
FAX: 65/3532166

**TAIWAN**
Taipei
TEL: 886/2-718-4533
FAX: 886/2-718-4526

**UNITED KINGDOM**
Berkshire, England
TEL: 44/344-780-782
FAX: 44/344-761-429

## The Company

Cirrus Logic, Inc., produces high-integration peripheral controller circuits for mass storage, graphics, and data communications. Our products are used in leading-edge personal computers, engineering workstations, and office automation equipment.

The Cirrus Logic formula combines proprietary S/LA™ † IC design automation with system design expertise. The S/LA design system is a proven tool for developing high-performance logic circuits in half the time of most semiconductor companies. The results are better VLSI products, on-time, that help you win in the marketplace.

Cirrus Logic's fabless manufacturing strategy, unique in the semiconductor industry, employs a full manufacturing infrastructure to ensure maximum product quality, availability and value for our customers.

Talk to our systems and applications specialists; see how you can benefit from a new kind of semiconductor company.

† U.S. Patent No. 4,293,783

**CIRRUS LOGIC, Inc.,** 3100 West Warren Ave. Fremont, CA 94538
TEL: 415/623-8300 FAX: 415/226-2160