CIRRUS LOGIC, Inc., 1463 Centre Pointe Drive, Milpitas, CA 95035 Telephone 408.945.8300 Telex 171918 FAX 408.263.5682

Technical Briefing:
CL–CD180 Intelligent Octal Channel Asynchronous Communications Controller Performance Benchmark

September, 1988

517001-001

CL–CD180 Intelligent Octal Channel Asynchronous Communications Controller Performance Benchmark

Data Communications Product Group
CIRRUS LOGIC, Inc.

The following is a sample program for a full duplex operation in a 10 Mhz 68010-based system. The CL–CD180 has a Global Interrupt Vector register that can be initiated with a device ID# by programming the most significant five bits of the vector; thus up to 32 CD180s can be directly accommodated. The lower three bits of the vector provides the exact cause of the interrupt.

In this example, we shall assume a single CD180 and look at the code required to handle normal data reception and transmission as well as to perform receive and transmit buffer management. The transmit and receive routines are identical, using different control blocks and buffer areas in memory.

The memory organization for receive data is as follows:

RBUFCRB (Buffer Control Block) points to the beginning of a 32 word control block organized into 8 sub-blocks, each with four 32-bit words for each channel. The same applies for memory control blocks for the transmit data buffer. The four words for each channel are:

- RBUFWAP - points to where received data is to be written into the channel's active buffer
- RBUFWEP - points to the last byte or end of channel's active buffer
- RBUFRAP - points to the beginning of next free receive buffer or the last free location in a used buffer
- RCV STATUS - holds current status of the receive buffers

  Bit 0 - if set indicates that the 1st receive write address

  is the address of a new receive buffer

  Bit 1 - if set indicates that the 2nd receive buffer

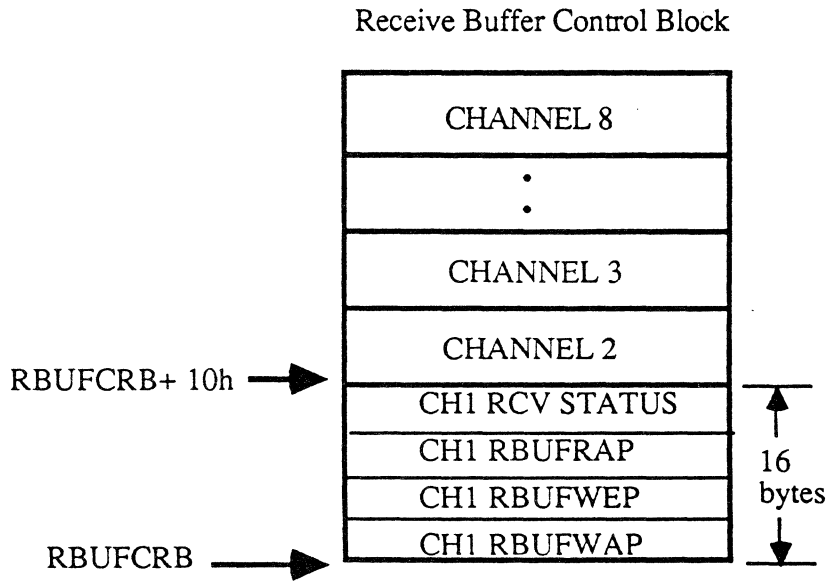
  address is the start of a new receive buffer

  Bit 2 - if set indicates that the 1st receive buffer is

  full, this means that both buffers are full and receive interrupts have been disabled

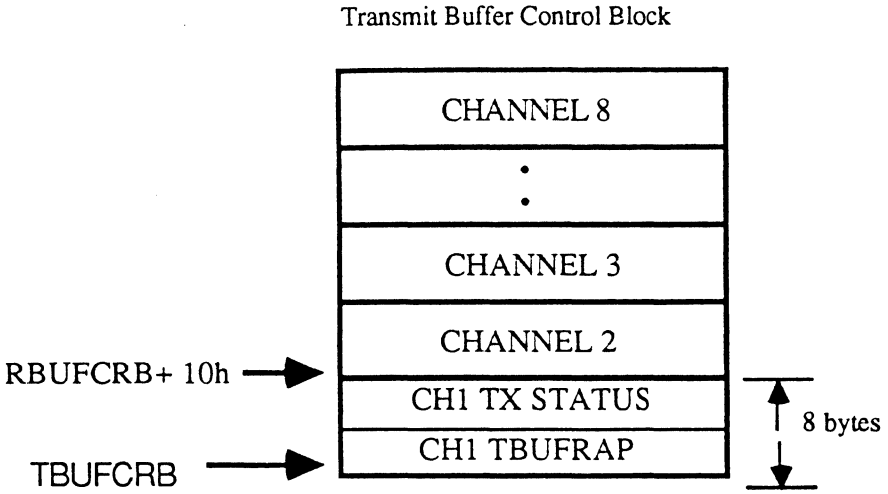
  Bit 3 - if set indicates that a the 2nd receive buffer is full

In this particular implementation, two buffers are chosen for each channel, an active and a standby buffer. The buffer sizes are chosen on the latency required, the length of the messages, the bit rate of the asynchronous communications, the speed at which the host can process the incoming data and the amount of memory available.



The receive buffers are organized as follows. The system will first allocate two free buffers setting bits 0 and 1 to indicate availability. Receive interrupts can then be enabled and bit 0 cleared indicating the buffer is in use. When the first buffer is filled the pointers in the table are swapped, bit 1 is cleared and bit 3 set to indicate that the 2nd buffer is full and needs to be processed by another routine. When the 2nd buffer has been processed and a new buffer has been allocated bit 1 is set again. If no new buffer is allocated then buffer 1 will eventually be filled, bit 2 set and receive interrupts disabled, as no receive data space is available.

Transmit buffers are organized in a simpler manner than the receive buffer. The control block contains an address and a count. Initially the address is set to the start of a transmit buffer and the count is set to the number of bytes that need to be transmitted. When the transmit interrupt is enabled the transmit routine will send the number of bytes in the buffer incrementing the address and decrementing the count as it progresses. The maximum number of bytes transmitted at any interrupt is 8 corresponding to the transmit FIFO size. When transmission of the buffer is complete the transmit interrupt is disabled. The count of bytes in the buffer will be zero which indicates to the system that a new block can be transferred.


Transmit Buffer Control Block
CHANNEL 8
CHANNEL 3
CHANNEL 2
RBUFCRB+ 10h
CH1 TX STATUS
8 bytes
CH1 TBUFRAP
TBUFCRB

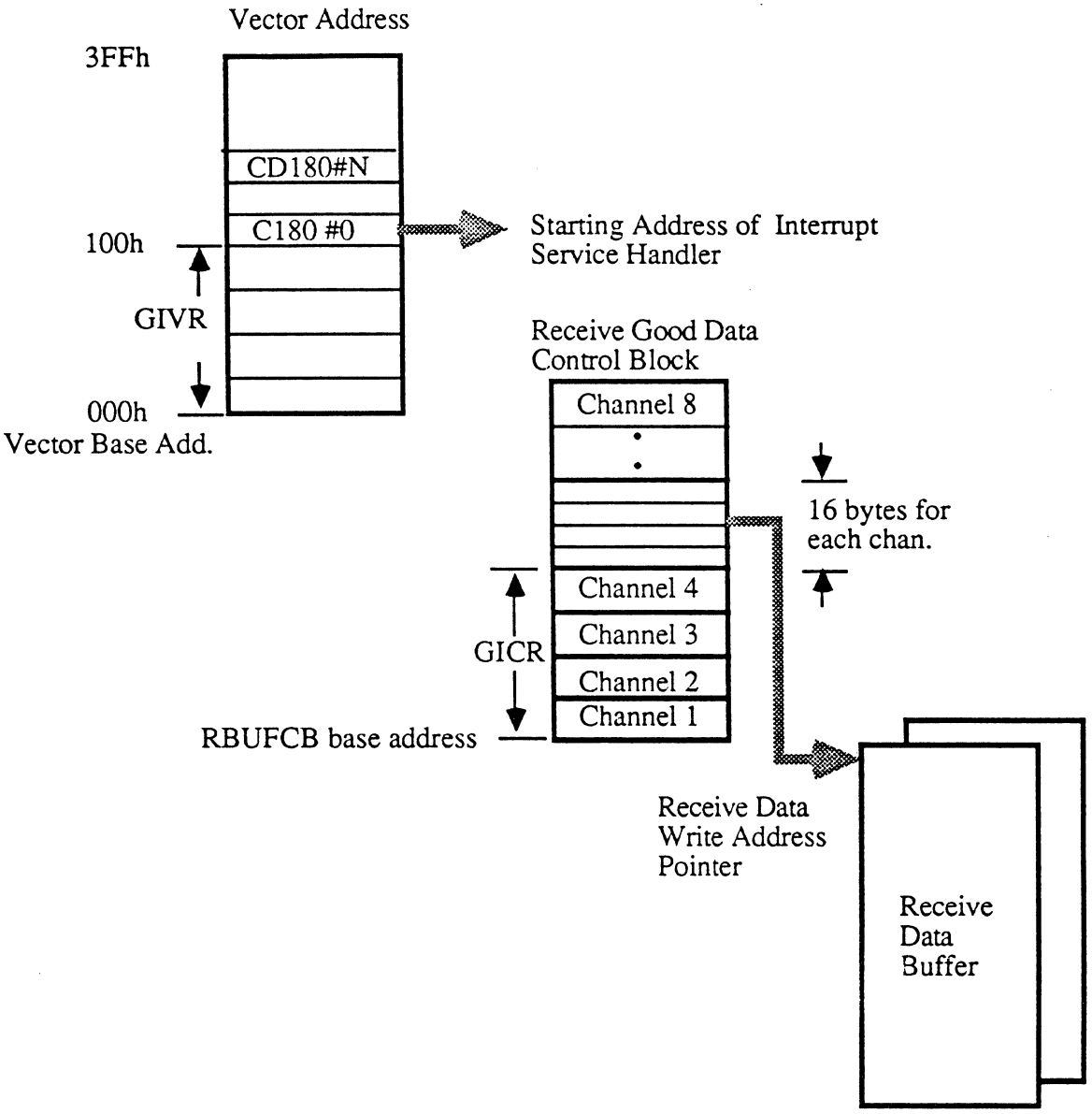
Register Definitions:

GICR - Global Interrupting channel Register - three bit encoded channel number denoting the interrupting channel's number. These three bits are located in bit positions 2-4. Bits 0-1, 5-7 are user programmable.

RDCR - Receive Data Count Register . This register stores the current number of consecutive good bytes of data available in the receive FIFO starting with the top of the FIFO. The host does not have to read the status register when fetching these bytes of data.

RDR - Receive Data Register. The host accesses this register to read the data of the interrupting channel's Receive FIFO. The CD180 will automatically switch the FIFO data into this register for the host to read.

TDR - Transmit Data Register. This has the identical function as that of RDR, except it is for use with the transmit operation.



Memory Organization for 680X0-based Interrupt System

Sample 680X0 Receive Good Data Interrupt Handling Program

```
# definition of CD180 in memory, located between 100000 and 1001ff
# on lower byte of memory
     CD180 =  0x100000     |address of CL-CD180 in memory

CCR  =  CD180 + 0x03
IER  =  CD180 + 0x05
COR1  =  CD180 + 0x07
COR2  =  CD180 + 0x09
COR3  =  CD180 + 0x0B
CCSR    =  CD180 + 0x0D
RDCR   =  CD180 + 0x0F
SCHR1  =  CD180 + 0x13
SCHR2  =  CD180 + 0x15
SCHR3  =  CD180 + 0x17
SCHR4  =  CD180 + 0x19
MCOR1  =  CD180 + 0x21
MCOR2  =  CD180 + 0x23
MCR  =  CD180 + 0x25
RTPR  =  CD180 + 0x31
MSVR  = CD180 + 0x51
RBPRH  =  CD180 + 0x63
RBPRL  =  CD180 + 0x65
TBPRH  =  CD180 + 0x73
TBPRL  =  CD180 + 0x77

# GLOBAL CD180 REGISTERS
GIVR  =  CD180 + 0x81
GICR  =  CD180 + 0x83
PILR1  =  CD180 + 0xC3
PILR2  =  CD180 + 0xC5
PILR3  =  CD180 + 0xC7
CAR  =  CD180 + 0xC9
GFRCR  =  CD180 + 0xD7
PPRH  =  CD180 + 0xE1
PPRL  =  CD180 + 0xE3
RDR   =  CD180 + 0xF1
RCSR  =  CD180 + 0xF5
TDR  =  CD180 + 0xF7
EOIR  =  CD180 + 0xFF

#receive control blocks 16 bytes per channel
.lcomm                  rbufcrb,128   |receive control block area

#transmit control blocks 8 bytes per channel

.lcomm                  tbufcrb,64    |transmit control block area

.text

# buffer size used for receive

buffsize = 256
# 680X0 based interrupt routines for cl-cd180
# save all registers used in the receive interrupt
# put the start address of the interrupting channel numbers control
# block in a0 = rbufcrb + (16 * channel number)
#             = rbufcrb + (4 * GICR)
```

```
                                                                  Bus R/W
        Instructions                    Comments                Cycles Clocks
# recv good data interrupt 68000                                    6     36
# recv good data interrupt 68010                                    7     38
rcvgd:  movem.l    a0-a2/d0,-(sp)  |save working registers         10     40
        lea        rbufcrb,a0      |                               3     12
        eor.l      d0,d0           |zero for addition              2      8
        move.b     GICR,d0         |                               4     16
        lsl.w      #2,d0           |16 bytes per block             1     10
        add.l      d0,a0           |point to correct               1      6
                                   |channel's control block

# a0 now points to the beginning of this channel's control block
# get current receive buffer pointer in a0
# number of good data bytes in d0, address of RDR in a2 (register
# relative addressing faster than direct addr)

        move.l     (a0),a1         |receive buffer write addr 3     12
        eor.w      d0,d0           |zero for addition              1      4
        move.b     RDCR,d0         |number bytes in rcv fifo       4     16
        lea        RDR,a2          |if 68020 do not do this        3     12

# loop to move the receive good data bytes into the buffer

rcvlp:  move.b     (a2),(a1)+      |save data
        dbne       d0,rcvlp        |continue till moved all data

#  above loop for 68000                                            5     22
#above loop for 68010  first 2 executions                         5     22
#                       subsequent executions                     2     14
#                        last execution                           4     18

# restore the pointer for the next interrupt and check if
# sufficient bytes remain in the buffer to support another interrupt

        move.l     a1,(a0)         |restore pointer               3     12
        move.l     (a0)4,d0        |receive buffer end            4     16
        sub.l      a1,d0           |# empty bytes in buffer       1      6
        cmp.w      #8,d0           |                              2      8
        bmi        swap            |swap if less 8 free bytes     2      8

# terminate interrupt and return

rxend:  move.b     d0,EOIR         |terminate cl-cd180s intr       4     16
        movem.l    (sp)+,a0-a2/d0  |                              10     40
        rte                        |intr complete 68000           5     20
                                   |               68010          6     24
# current buffer unable to support another interrupt
# test if 2nd buffer is available

swap:   bclr       #1,(a0)12       |is new buff available         4     16
        beq        nobuf           |                              2     10

# new buffer is available - flag 2nd buffer full
# swap buffers and calculate buffer end point
# and terminate interrupt
```

```
        bset      #3,(a0)12       |2nd buffer full          4      16
        move.l    (a0)8,(a0)      |start addr of next buff  6      24
        move.l    a1,(a0)8        |old buffer end ptr        6      24
        move.l    (a0),d0         |                         3      12
        add.l     buffsize,d0     |                         3      16
        move.l    d0,(a0)4        |write buff end point     3      12
        move.b    d0,EOIR         |terminate cl-cd180s intr 4      16
        movem.l   (sp)+,a0-a2/d0  |                        10      40
        rte                       |intr complete 68000      5      20
                                  |              68010      6      24

# 2nd buffer not available flag 1st buffer as full,
# disable further receive interrupts and
# terminate this interrupt

nobuf:  bset      #2,(a0)12       |1st buffer full          4      16
        bclr      #0x10,IER       |clear receive data intr  6      24
        move.b    d0,EOIR         |terminate cl-cd180s intr 4      16
        movem.l   (sp)+,a0-a2/d0  |                        10      40
        rte                       |intr complete 68000      5      20
                                  |              68010      6      24


# transmit interrupt - one buffer used, plus count of number of bytes in
# buffer, when count reaches zero it indicates buffer has been sent

#       transmit interrupt 68010                            7      38
txint:  movem.l   a0-a2/d0,-(sp)  |save registers          10      40
        lea       tbufcrb,a0      |tx control block addr    3      12
        eor.l     d0,d0           |zero for addition        2       8

        move.b    GICR,d0         |interrupting channel no  4      16
        lsl.w     #1,d0           |8 bytes per block        1       8
        add.l     d0,a0           |point to correct block   1       6
                                  |for this channel

# a0 now points to the correct channels control block
# get the transmit pointer and check the number of bytes left

        move.l    (a0),a1         |tx buffer read address   3      12
        move.w    (a0)4,d0        |remaining byte count     3      12
        cmp.w     #9,d0           |check if > 8 bytes left  2       8
        bcs       lstblk          |if 8 or less - last block 2      8

# more than 8 bytes left in buffer so transmit 8 this time

        move.w    #8,d0           |transmit 8 bytes         2       8
        bra       trans           |                        2      10

# 8 or fewer bytes to transmit so this is the last block

txlst:  bclr      #2,IER          |disable transmit interrupt 6    24

# restore the count for next interrupt

trans:  sub.w     d0,(a0)4        |update transmit count    4      16
        lea       TDR,a2          |for loop speed           3      12
```

```
# loop till sent d0 characters

txlp:   move.b     (a2),(a1)+      |transmit byte            3      12
        dbne       d0,txlp         |until done all           2      10

# restore pointer and terminate interrupt

        move.l     a1,(a0)         |save ptr for next intr   3      12
        move.b     d0,EOIR         |terminate intr in CD180  4      16
        movem.l    (sp)+,a0-a2/d0  |restore registers       10      40
        rte                        |and return 68000         5      20
                                   |           68010         6      24
```

	Receive	Transmit
Total bus cycles for one good character	77	77
Total bus cycles for eight good characters	96	96
Average bus cycles per character using full FIFO	~12	~12
Total number of clocks for one good character:		
- zero wait state	326	328
- one wait state	403	405
Total number of clocks for eight good characters:		
- zero wait state	436	438
- one wait state	532	534
Average number of clocks per character when using full FIFO:		
- zero wait state	~55	~55
- one wait state	~64	~64

Average 68010 time per character (10 Mhz, 1 wait state) = 6.4 microseconds

Thus full duplex operation on all channels operating at 9600 bps (960 characters per second) would take 16 half duplex channels x 960 characters/channel x 6.4 µsec/character = ~98,304 µsec or about 10 % of real time. The number of full duplex asynchronous channels capable of being supported by the 68010 at 10 Mhz, if dedicated to interrupt handling only, would be:

- 9, 600 bps 80 channels (10 CD180s)
- 19, 200 bps 40 channels (5 CD180s)
- 38, 400 bps 20 channels (< 3 CD180s)

Sample 68010 Interrupt Service Routine for Signetics SCC2698

The code that follows is taken from the July 15, 1987 edition of Electronic Products magazine: "CMOS Octal UART design enhances multi-channel datacomm design" by A. Goldberger, J. Goodhart, R. Carreras.

The code appears to handle the reception and transmission of characters to and from memory. The actual buffer management code is apparently not part of the service routine. Please contact Signetics Corp. (Sunnyvale, CA) for any questions. Due to the interrupt vectoring mechanism defined the code will only run on a 68010 and not 68000, so only 68010 timings are shown

	Instructions		Comments	Bus R/W Cycles	Clocks
	; **interrupt itself**				38
INT98	MOVEM.L	A0-A2/D0-D1, -(SP)		12	48
	MOVE.W	26(SP), D0		3	12
	SUB.W	VBASE98, D0		3	12
	LSL.W	#3, D0		1	12
	LEA	BASE98, A0		3	12
	MOVE.B	SRA (A0, DO.W), D1		3	14
	BNE.S	HAVUART		1/2	6/10
	ADD	$16, D0		2	8
	MOVE.B	SRA (A0, Do.W), D1		3	14
	BEQ	INTERR		2	10
HAVUART	LEA	BUFAD98, A1		3	12
	CMP.B	#$F, D1		2	8
	BGT.S	RCVERR		1	6
	BTST	#0, D1		2	10
	BEQ.S	DOXMIT		1/2	6/10
RCVE	MOVE.L	0 (A1, D0.W), A2		4	18
	MOVE.B	RHR (A0, D0.W), (A2)+		3	18
	MOVE.L	A2, 0 (A1, D0.W)		4	18
	SUBQ.W	#1, 4(A1, D0.W)		4	18
	BLE.S	RCVEND		1	6
	MOVEM.L	(SP)+, A0-A2/D0-D1		12	52
	RTE			6	24

DOXMIT	MOVE.L	8 (A1, D0.W), A2	4	18
	MOVE.B	(A2)+, THR (A0, D0.W)	3	18
	MOVE.L	A2, 8(A1, D0.W)	4	18
	SUBQ.W	#1, 12(A1, D0.W)	4	18
	BLE.S	XMITEND	1	6
	MOVEM.L	(SP)+, A0-A2/D0-D1	12	52
	RTE		6	24

1st UART/2nd UART		Receive	Transmit
Total bus cycle		76/83	76/83
Total clock	- 0 wait state	356/384	360/388
	- 1 wait state	436/471	440/475

Average 68010 time per byte @ 10 Mhz and 1 wait state = 45.5 µsec

Supporting full duplex operation on all channels at 9600 bps (960 characters per second) would take 16 x 960 x 45.55 = 679,648 µsec or about 68% of real time.

Additional time should be required to manage the receive and transmit buffers used by the device. Assuming ZERO time for these tasks, the number of full duplex asynchronous channels capable of being supported by the 68010 @10 Mhz:

- @ 9,600 bps 11 channels (> 1 SCC2698)
- @ 19,200 bps 5 channels (< 1 SCC2698)
- @ 38,400 bps 2 channels