TECHNICAL CONSIDERATIONS WHEN IMPLEMENTING LOCALTALK LINK ACCESS PROTOCOL

The LLAP Protocol is an important part of the Appletalk network system. It manages access to the node-to-node transmission of network data packets, governs access to the link, and provides a means for nodes to discover valid addresses...all error free.

INTRODUCTION

The LLAP (LocalTalk Link Access Protocol) is the ISO/OSI (International Standards Organization/Open Systems Interconnection) link layer protocol of the AppleTalk network system. This protocol manages the node-to-node transmission of data packets in the network. LLAP governs access to the link and provides a means for nodes to discover valid addresses. It does not guarantee packet delivery; it does guarantee that those packets that are delivered are error-free.

This Appnote (Application Note) does not address the architectural issues of writing a driver but it does focus on the details of using an SCC to send and receive LLAP frames. However, some of the problems of transmitting and receiving LLAP frames are discussed, using sample code written for Zilog's Z80181 Emulation Adapter Board. Also, the problems of sending sync pulses, timing transmissions and determining that a frame has been received properly will be discussed.

GENERAL DESCRIPTION

The LocalTalk Link Access Protocol (LLAP) is the ISO-OSI link layer protocol of the AppleTalk network system using LocalTalk. Along with ELAP (the corresponding Ethernet link layer protocol) and TLAP (the Token Ring link layer protocol), it provides the foundations upon which the other protocols rest. The LLAP protocol supports the node-to-node transmission of packets used by DDP and RTMP to route packets around the internetwork; DDP, in turn, supports the name binding functions of NBP, the reliable frame delivery of ATP, and the rest of the AppleTalk protocol stack.

A majority of the difficult timing and all of the hardware interface problems crop up in the LLAP driver. These problems are so difficult that it makes sense to start writing such a driver by writing experimental routines that transmit and receive frames. This App Note addresses the intricacies of the interframe and interdialog timings before trying to engineer code that will truly be a driver. Also, some of the experimental routines to run on the Z80181 Emulation Adapter Board will be explained.

The LLAP provides the basic transmission of packets from one node to another on the same network. LLAP accepts packets of data from clients residing on a particular node and encapsulates that data into its proper LLAP data packet. The encapsulation includes source and destination addresses for proper delivery. LLAP ensures that any damaged packet is discarded and rejected by the destination node. The LLAP makes no effort to deliver damaged packets.

Carrier Sense Multiple Access with Collision Avoidance

It is LLAP's responsibility to provide proper link access management to ensure fair access to the link by all nodes on that network. The access discipline that governs this is known as Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). A node wishing to gain access to the link must first sense that the link is not in use by any other node (carrier sense); if the link has activity, then the node wishing to transmit must defer transmission. The ability of LLAP to allow multiple access to the link also

GENERAL DESCRIPTION (Continued)

leaves room for possible collisions with other data packets. LLAP attempts to minimize this probability (collision avoidance).

Two techniques are used by LLAP in its implementations of CSMA/CA. LLAP outlines this procedure but falls short in endorsing which hardware to use. (The SCC is, of course, used by Apple.) The first technique takes advantage of the distinctive 01111110 flag bytes that encapsulate the data packet (note that this implies that SDLC is used). LLAP stipulates that a minimum of two flags precede each of these data packets. The leading flag characters provide byte synchronization and give a clue to any listener that some other node is using the link at a particular time (use the Hunt bit in RR0 if the SCC is used).

In SDLC mode, the receiver automatically synchronizes on the flag byte and resets the Hunt bit to zero. The SCC has some latency in detecting these flag bytes due to the shifter, etc. This is not ideal because the node needing to transmit may determine that the link is free, when in fact the flag bytes are still being shifted into its receiver (i.e., the link is not idle at all).

A closing flag is also needed to fully encapsulate the data packet. LLAP requires that 12 to 18 ones be sent after this closing flag. The LocalTalk hardware (i.e., the SCC) interprets this as an abort sequence and causes the node's hardware to lose byte sync; this then confirms that the current sender's transmission is over. In SDLC mode, seven or more contiguous 1's in the receive data stream forces the receiver into Hunt (Hunt bit set) and an External/Status interrupt can be generated. This is important because the node wishing to use the bus can simply wait for this interrupt before preparing to transmit it's packet.

LLAP uses a second technique in its carrier sensing. LLAP requires that a synchronization pulse for an idle period of at least two bit times be transmitted prior to sending the RTS handshaking frame (Figure 1). This synchronization is obtained by first enabling the hardware line so that an edge is detected by all the receivers on the network. This initial edge is perceived as the beginning of the clocking period. It is soon followed by an idle period (a period with no carrier) of at least two bit times. All the receivers on the network see this idle period and assume that the clock has been lost (missing clock bit set on RR10). This method is much more immediate than the byte flag synchronization method and provides a quicker way of determining whether the link is in use. Unfortunately, an interrupt is not generated by this missing clock and, therefore, polling must be implemented.

The Z80181 code used for polling the missing clock bit is approximately fifty clock cycles which at 10 MHz is about 5 µsec or about one bit time. This is still relatively quicker than the time required for the SCC to reset the Hunt bit (the flag character takes at least eight bit times for it to be shifted through the buffer before the Hunt bit is reset to zero). Synchronization pulses can be sent before every frame but because of the time constraints associated with the interframe gaps it makes sense to send such pulses only before the lapENQ and lapRTS frames.

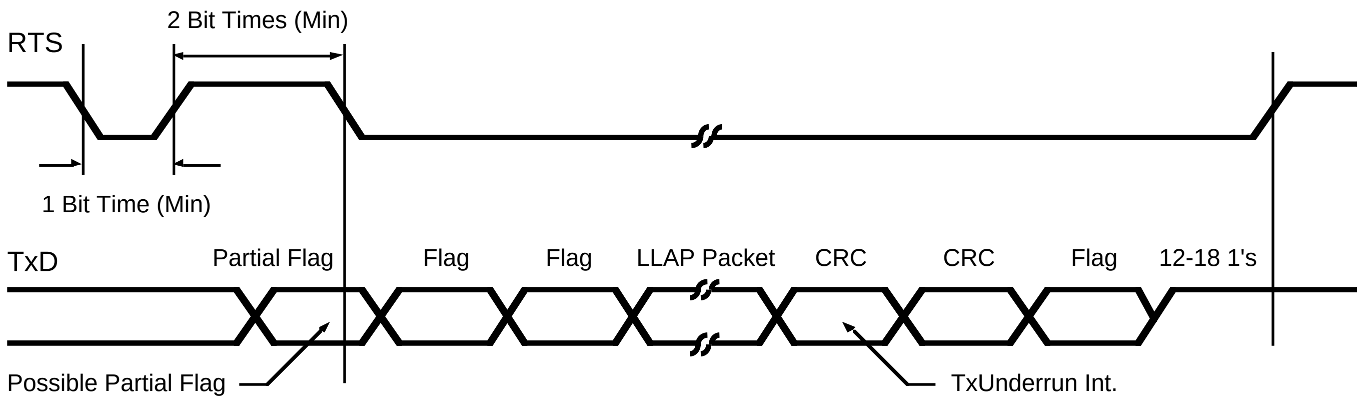


Figure 1. CSMA/CA Synchronization Pulse Timing Diagram

Dynamic Node ID

LLAP requires the use of an 8-bit node identifier number (node ID) for each node on the link. Apple had decided that all LLAP nodes must have a dynamically assigned node ID. A node would assign itself its unique address upon activation. It is then up to that particular node to ascertain that the address it had chosen is unique. A node randomly chooses an 8-bit address (for example, the refresh register value on the Z80181 is added to a randomly chosen value on the receive buffer to obtain a pseudo random 8-bit address).

The node then sends out an LLAP Enquiry control packet to all the other nodes and waits for the prescribed interframe gap of 200 µsec. If another node is already using this node ID, then that node must respond within 200 µsec with a LLAP Acknowledgment control packet. The new node must then rebroadcast a new guess for its node ID. If a LLAP Acknowledgment packet is not received within 200 µsec then the new node assumes that the address is indeed unique. The new node must rebroadcast the LLAP enquiry packet several more times to account for cases when the packet could have been lost or when the guessed node ID is busy and could have missed the Enquiry packet.

LLAP Packet

LLAP packets are made up of three header bytes (destination ID, source ID and LLAP type) and 0 to 600 bytes of variable length data. The LLAP type indicates the type of packet that is being sent. 80H to FFH are reserved as LLAP control packets. The four LLAP control packets that are currently being used are: The lapENQ, which is used as enquiry packet for dynamic node assignments; the lapACK, which is the acknowledgment to the lapENQ; the lapRTS, which is the request to send packet that notifies the destination of a pending transmission; and the lapCTS, which is the clear-to-send packet in response to the RTS packet. Control packets do not contain data fields.

LLAP Packet Transmission

LLAP distinguishes between two types of transmissions: a directed packet is sent from the source node to a specific destination node through a directed transmission dialog; a broadcast packet is sent from the source node to all nodes on the link (destination ID is FFH) through a broadcast transmission dialog. All dialogs must be separated by a minimum Inter Dialog Gap (IDG) of 400 µsec. Frames within these dialogs must be separated from each other with a maximum Inter Frame Gap (IFG) of 200 µsec.

The source node uses the physical layer to detect the presence or the absence of data packets on the link. The node will wait until the line is no longer busy before attempting to send its packets. If the node senses that the line is indeed busy, then this node must defer. When the node senses that the line is idle, then the node waits the minimum IDG plus some randomly generated time before sending the packet (the line must remain idle throughout this period before attempting to send the packet). The initial packets to be sent are handshaking packets. The first packet sent by the source node to its destination node is the RTS packet. The receiver of this RTS packet must return a CTS packet within the allowable maximum IFG. The source node then starts transmitting the rest of its data packet upon receiving this CTS.

Collisions are more likely to occur during the handshaking phase of the dialog. The randomly generated time that is added to the IDG tends to help spread out the use of the link among all the transmitters. A successful RTS to CTS handshake signifies that a collision did not take place. An RTS packet that collides with another frame has corrupt data that shows up as a CRC error on the receiving or the destination node. Upon receiving this, the destination node infers that a collision must have taken place and abstains from sending its CTS packet. The source or the transmitting node sees that the CTS packet was not received during the IFG and also infers that a collision did take place. The sending node then backs off and retries.

The LLAP keeps two history bytes that log the number of deferrals and collisions during a dialog. These history bytes help determine the randomly generated time that is added to the IDG. The randomly generated time is readjusted according to the traffic conditions that are present on the link. If collisions or deferrals have just occurred on the most recently sent packets, then it can be assumed that the link has heavier than usual traffic. Here, the randomly generated number should be a larger number in order to help spread out the transmission attempts. Similarly, if the traffic is not so great, then the randomly generated number should be smaller, thus reducing the dispersion of the transmission attempts.

LocalTalk Physical Layer

LocalTalk uses the SDLC format and the FM0 bit encoding technique. The RS-422 signalling standard for transmission and reception was chosen over the RS-232 because a higher data rate over a longer physical distance is required. LocalTalk requires signals at 230.4 Kbits per second over a distance of 300 meters.

HARDWARE CONFIGURATION

As shown in Figure 2, the hardware used to implement this LLAP driver consists of the Z80181 (an integration of the Z80180 compatible MPU core with one channel of a Z85C30 SCC, Z80 CTC, two 8-bit general-purpose parallel ports and two chip select signals) operating at 10 MHz, a 3.6864 MHz clock source and an RS-422 line driver with tri-state.

The SCC's clocking scheme decouples the microprocessor's clock from the communication clock (Figure 3). The DPLL uses the /RTxC pin as its source. The /RTxC also drives the Baud Rate Generator which divides its input by sixteen. The resulting 230.4 kHz signal is then used as transmitter clock. This 230.4 kHz signal is also used by one of the Z80181's counter/timer trigger inputs (Z80 CTC's channel 1) which is used to count the number of elapsed bit times. In counter mode, each active edge to the CTC's CLK/TRG1 input causes the downcounter of the CTC to be decremented. The /TRxC pin is programmed as BRG output and is connected to the CLK/TRG1 input through an external wire.

The /RTS signal is used to tri-state RS-422 to allow other node transmitters to drive the line. This signal is asserted and deasserted through bit1 of the SCC's Write Register 5.

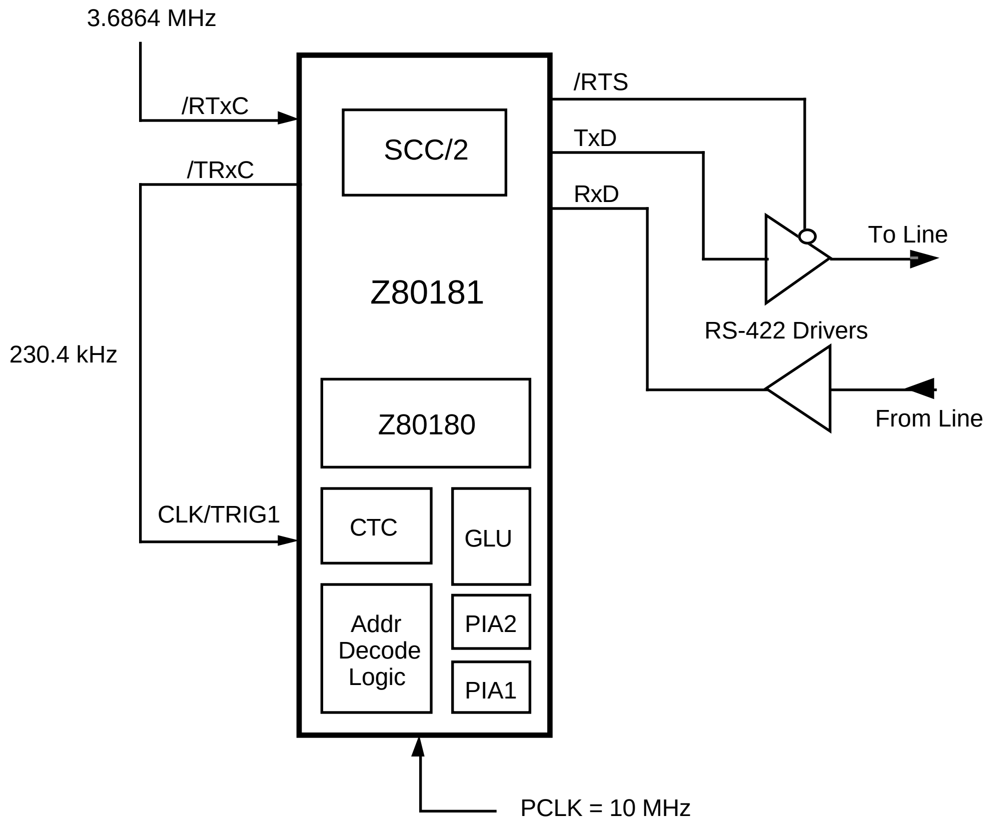


Figure 2. Driver Hardware Configuration

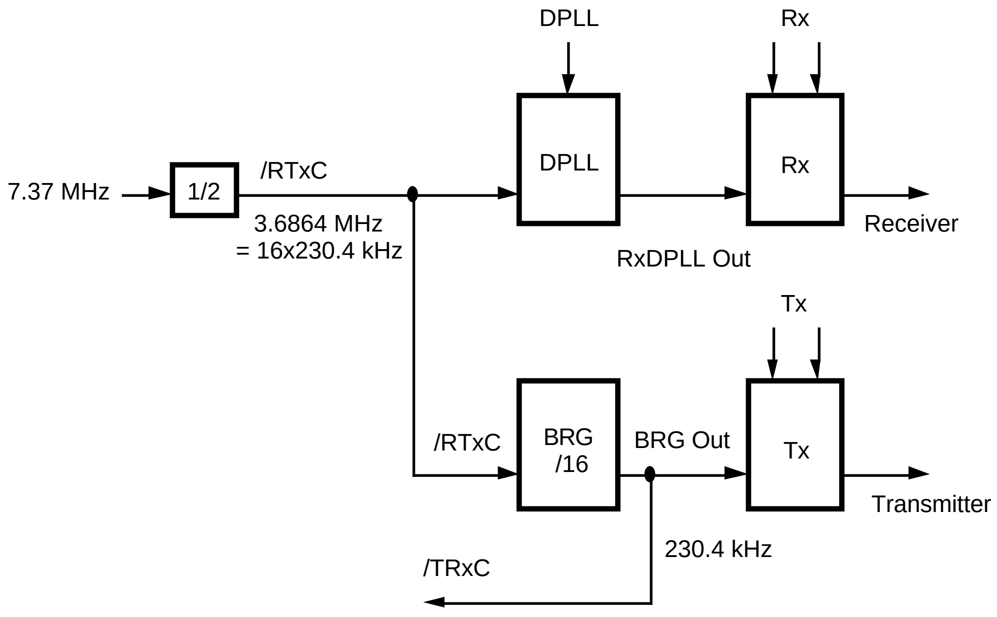


Figure 3. SCC Clocking Scheme

Listing 1 (Reference Appendix A for Listings 1 through 4) shows the assembler code for this SCC initialization. Note that the SCC is treated as a peripheral by the Z80181's MPU. For example, an I/O write to the scc_cont (address e8H) or to the scc_data (address e9H) is a write to the SCC's control and data registers, respectively. As shown in Listing 1, the SCC is initialized by issuing I/O writes to the pointer and then to the control registers in an alternating fashion. It is therefore very important that all interrupts are disabled during this initialization routine.

The SCC is initially reset through software before proceeding to program the other write registers. A NOP is sufficient to provide the four PCLKs required by the SCC recovery time after a soft reset. The SCC is programmed for SDLC mode. The receive, transmit and external interrupts are all initially disabled during this initialization. Each of these interrupt sources are enabled at their proper times in the main program. The SCC is programmed to include status information in the vector that it places on the bus in response to an interrupt acknowledge cycle (see Listing 4 of the SCC interrupt vector table for all the possible sources).

Since SDLC is bit-oriented, the transmitter and receiver are both programmed for 8 bits per character as required by LLAP. Address filtering is implemented by setting the Address Search Mode bit 2 on WR3. Setting this bit causes messages with addresses not matching the address programmed in WR6 and not matching the broadcast address to be rejected. Values in WR10 presets the CRCs to ones, sets the encoding to FM0 mode and makes certain that transmission of flags occur during idle and underrun conditions. WR11 is set so that the receive clock is sourced by the DPLL output; the transmit clock is sourced by the Baud Rate Generator output; /TRxC's output is from the BRG. The input to the BRG is from the /RTxC.

The BRG's time constant is loaded in WR13 and WR12 so that the /RTxC's 3.6864 MHz signal is divided by 16 in order to obtain a 230.4 kHz signal for the transmitter clock. WR14 makes certain that the DPLL is disabled before choosing the clock source and operating mode. The DPLL is enabled by issuing the Enter Search Mode in WR14.

TRANSMITTING A LLAP FRAME

Listing 2 shows the assembler code for subroutine txenq, which sends an lapENQ frame on the line once the system has determined that the line is quiet. Note that this routine can easily be generalized to send any frame.

The first responsibility of txenq is to send the sync pulse required by the CSMA/CA protocol. To do this, txenq sets the /RTS pin active low, enabling the transmitter drivers, and then sets it high again to disable them. In order to satisfy the requirements of the CSMA/CA protocol, the transmitter drivers must remain off for at least one bit time (4.3 µsec) to guarantee that all the receivers see at least one transition. Our routine satisfies this requirement because the two ld instructions (7 T states each), the two nop instructions (4 T states each) and the two “out” instructions (11 T states each) required to set the /RTS line high, take more than 4.3 µsec to execute on the 10 MHz Z80181. The transmitter drivers must then remain off for at least two bit times in order to ensure that all receivers lose clock; again, the routine depends upon the time required to execute the instructions before we turn the transmitter drivers on again.

After sending the sync pulse and waiting the required period of silence, txenq turns on the transmitter drivers to send the frame. Now, the routine must wait while the SCC sends out the leading flags. This takes 16 bit times, and since the SCC does not tell when this has happened, the transmit routine has no choice but to time this. Our routine does this by calling bit time, which is discussed below.

When the two flags have been transmitted, the first data byte is written to the data register of the SCC. Thereafter, the routine polls the SCC status register, and when that register shows the transmit buffer register is empty, the routine sends the next data character. This polling method can obviously be replaced by an interrupt routine that is entered when the transmit buffer is empty or by setting up the Z80181’s DMA to send characters on demand to the SCC.

After the first data byte is transmitted, the txenq routine sets the SCC to mark on idle so that the abort is sent when the frame is over. This operation can be done any time after the first data character has been placed in the transmit buffer and before the trailing flag is shifted out. Txenq asserts this mark on idle command after the first character is placed in the transmit buffer so that LLAP has control and that no issues of latency may arise (particularly in designs using interrupt or DMA).

After the last data byte is written to the SCC, the transmit routine must wait while the last data byte (the one that the SCC had just sent to shifter), the two CRC bytes, one flag byte and 12 to 18 bit times of marking are transmitted. This total of 44 to 50 bit times is again timed by bittime. When bittime indicates that enough time has elapsed, the transmitter drivers are turned off.

Since our hardware includes connecting the output of the baud rate generator to the input of counter/timer 1 on the Z80181, that counter timer counts the bit times. The bit time routine feeds an appropriate count value into the counter and enables an interrupt routine to receive control when the count expires. The interrupt routine ctc1int, shown in Listing 4, sets the timeflag which the transmit routine polls.

Note that the call to bittime, the interrupt routine, the polling code and the length of time it takes to write to the SCC registers after a polling loop is exited, all take up a time that can be a significant number of bits. In order to do these timings accurately, calculate the number of PCLK cycles required to execute these pieces of code and to adjust the counter value that bittime requires. This adjustment is dependent on the hardware configuration and on the exact implementation details of the code.

Note, incidentally, that software must put the entire frame into the transmit register, including the addresses. The SCC does not generate addresses even when set in WR6.

RECEIVING LLAP FRAMES

In the experiments, the interrupt routines were used to receive characters and to handle special conditions when the frame is complete. Listing 3 shows the interrupt handlers that are entered when the SCC receives a character and when the SCC interrupts for a special condition (typically, end of frame). As with transmission, it is obvious that we could receive characters by polling the SCC (using up all available CPU cycles) or by using DMA (using up very few). It is estimated that the recint routine uses up about 1/3 of the available 34 µsec (4.3 µsec x 8-bit times) cycles on a 10 MHz processor.

The recint routine moves each character as it is received from the SCC to a memory buffer and increments the buffer pointer. The frame’s data length is checked to make certain that the maximum allowable frame size is not exceeded.

The spcond interrupt handler checks the status of the SCC to find out what has happened. The presence on an overrun condition or a CRC error is flagged as a receive frame error.

The second routine decrements the receiver buffer address by two to account for the two CRC bytes that are read from the SCC before the special condition interrupt occurs. Note that the SCC does not filter these CRC bytes, nor does it filter the address byte. Everything received after the leading flags and before the trailing flags appears in the receive buffer.

One difficulty that arises in LLAP that was not addressed here is that the receipt of a frame very often creates an obligation to send a frame back to the sender within the interframe gap, which is 200 µsecs. This difficulty is even greater than it might appear. The 200 µsec gap starts when the frame is received; it ends when the leading flags and destination address of the response are sent. Start sending the response soon enough to allow sending two leading flags (plus a possible leading flag fragment) and the first data character, and to allow for the 3-bit delay in the SCC shifter. Therefore, start sending early enough to transmit 35 bits before the interframe gap expires, or about 70 µsecs after you receive the frame.

CONCLUSIONS

The problems of sending the sync pulses, the timing of transmission packets, and the problems associated with the reception of packets as defined by LLAP are handled by the Z80181 and its peripherals. It was demonstrated that LLAP frames can be transmitted and received by using the straight forward polling method and by using interrupt routines. In a much busier environment where the processor cannot strictly be an LLAP engine, other methods such as using DMA in a fully interrupt driven environment must be used. It was also demonstrated that severe CPU overhead is used in setting up the sync pulses, timing out delays, etc., before each LLAP frame. A modified SCC that transmits and receives special LLAP frames helps in off loading some of this overhead, hence freeing the CPU to do other tasks.

APPENDIX A

Listing 1- Asembler Code for SCC Initialization

LISTING 1

```
                                                                     ;**************************************
                                                                     ;subroutine to initialize scc registers
                                                                     ;**************************************
000001e2                  initscc:
000001e2 f3                         di                               ;disable int while programming scc
000001e3 f5                         push        af
000001e4 c5                         push        bc
000001e5 e5                         push        hl

000001e6 3e09                       ld          a,09h                ;WR9
000001e8 d3e8                       out         (scc_cont),a         ;point to scc register
000001ea 3e80                       ld          a,80h                ;channel  reset
000001ec d3e8                       out         (scc_cont),a         ;scc register value
000001ee 00                         nop                              ;delay needed after scc reset


000001ef 21Wwww                     ld          hl,scctable          ;fetch start of scc init table
000001f2                  scc1:
000001f2 7e                         ld          a,(hl)               ;fetch register pointer value
000001f3 feff                       cp          0ffh
000001f5 caWwww                     jp          z,finscc             ;if reg a =0ffh then initscc finished
000001f8 d3e8                       out         (scc_cont),a
000001fa 23                         inc         hl
000001fb 7e                         ld          a,(hl)
000001fc d3e8                       out         (scc_cont),a
000001fe 23                         inc         hl
000001ff c3R000+01f2,               jp          scc1                 ;loop back

00000202                  scctable:
00000202 04                         db          04h                  ;WR4
00000203 20                         db          00100000b            ;sdlc uses 1x,sdlc mode,no parity

00000204 01                         db          01h                  ;WR1
00000205 00                         db          00h                  ;nothing,rx,tx and ext int disabled

00000206 02                         db          02h                  ;WR2
00000207 00                         db          00h                  ;vector base is 00h

00000208 03                         db          03h                  ;WR3
00000209 cc                         db          0cch                 ;rx 8b/char,rx crc enabled,address
                                                                     ;search mode for adlc address filtering
                                                                     ;rx disabled.

0000020a 05                         db          05h                  ;WR5
0000020b 60                         db          60h                  ;tx 8b/char, set rts to disable drivers

0000020c 06                         db          06h                  ;WR6
0000020d 00                         db          00h                  ;address field='myaddress' in main pgm

0000020e 07                         db          07h                  ;WR7
0000020f 7e                         db          7eh                  ;flag pattern

00000210 09                         db          09h                  ;WR9
00000211 01                         db          01h                  ;stat low, vis therefore vector returned
                                                                     ;is a variable depending on the source
                                                                     ;of the interrupt.

```

```
00000212 0a          db      0ah     ;WR10
00000213 e0          db      0e0h    ;crc preset to one,fm0, flag idle/undr

00000214 0b          db      0bh     ;WR11
00000215 f6          db      0f6h    ;rtxc=xtal,rxc=dpll,txc=brg,trxc=brg out

00000216 0c          db      0ch     ;WR12
00000217 06          db      06h     ;brg tc low,for 230.4kbps using rtxc=3.68MHz

00000218 0d          db      0dh     ;WR13
00000219 00          db      00h     ;brg tc high

0000021a 0e          db      0eh     ;WR14
0000021b 60          db      60h     ;disable dpll
                                     ;no local loop back,brg source=rtxc

0000021c 0e          db      0eh     ;WR14
0000021d c0          db      0c0h    ;select fm mode
                                     ;no local loop back,brg source=rtxc

0000021e 0e          db      0eh     ;WR14
0000021f a0          db      0a0h    ;dpll source=rtxc,
                                     ;no local loop back,brg source=rtxc

00000220 0e          db      0eh     ;WR14
00000221 20          db      20h     ;enter search mode
                                     ;no local loopback

00000222 0e          db      0eh     ;WR14
00000223 01          db      01h     ;null,no local loopback,enable the brg

00000224 03          db      03h     ;WR3
00000225 cc          db      0cch    ;rx 8b/c,enable rx crc,addrs src,rx disable

00000226 0f          db      0fh     ;WR15
00000227 00          db      00h     ;ext/stat not used

                                     ;WR0
00000228 10          db      10h     ;reset ext/stat once
00000229 10          db      10h     ;reset ext/stat twice

0000022a 01          db      01h     ;WR1
0000022b 00          db      00h     ;disable all int sources

0000022c 09          db      09h     ;WR9
0000022d 09          db      09h     ;enable int
0000022e ff          db      0ffh    ;finished

0000022f         finscc:
0000022f e1          pop     hl      ;
00000230 c1          pop     bc      ;
00000231 f1          pop     af      ;
00000232 c9          ret

```

LISTING 2

```
                                                  ;*************************************************
                                                  ;Subroutine to transmit the llapenq packet
                                                  ;*************************************************
00000244         txenq:

00000244 f5              push    af               ;save status and a reg
00000245 c5              push    bc               ;save
00000246 e5              push    hl               ;save
                                                  ;
00000247 f3              di                       ;make sure that
                                                  ;no interrupt routine
                                                  ;nor should interrupt
                                                  ;occur during
                                                  ;this subroutine.
00000248 3e03            ld      a,03h
0000024a d3e8            out     (scc_cont),a     ;WR3
0000024c 3ecc            ld      a,0cch
0000024e d3e8            out     (scc_cont),a     ;8b/char,rx crc
                                                  ;enable,addrs src
                                                  ;and rx disabled

00000250 3e0a            ld      a,0ah            ;select WR10
00000252 d3e8            out     (scc_cont),a
00000254 3ee0            ld      a,11100000b      ;idle with flags
00000256 d3e8            out     (scc_cont),a

                                                  ;****enable transmitter *****
00000258 3e05            ld      a,05h            ;select WR5
0000025a d3e8            out     (scc_cont),a
0000025c 3e68            ld      a,01101000b      ;enable tx
0000025e d3e8            out     (scc_cont),a
                                                  ;
                                                  ;
                                                  ;****enable rs-422 driver *****
00000260 3e05            ld      a,05h            ;select WR5
00000262 d3e8            out     (scc_cont),a
00000264 3e6a            ld      a,01101010b      ;enable tx,
00000266 d3e8            out     (scc_cont),a     ;reset rts
00000268 00                      nop
00000269 00                      nop
                                                  ;nop's needed to complete 4.3 usec
                                                  ;for 1 bit time enable of transmitter.
                                                  ;total delay=2*(7+11+4) T states at 10 MHZ
                                                  ;
                                                  ;****disable rs-422 driver for 2 bit times*****
0000026a 3e05            ld      a,05h            ;select WR5
0000026c d3e8            out     (scc_cont),a
0000026e 3e68            ld      a,01101000b      ;enable tx, set rts
00000270 d3e8            out     (scc_cont),a
                                                  ;
00000272 3e80            ld      a,10000000b      ;reset txcrc
00000274 d3e8            out     (scc_cont),a
00000276 0601            ld      b,01h            ;delay count
00000278         csloop:
00000278 10fe            djnz    csloop           ;loop needed
                                                  ;to complete
                                                  ;8.6 usec  min.
                                                  ;or 2 bit times.
                                                  ;****enable rs-422 driver for llap transmission*****
0000027a 3e05            ld      a,05h            ;select WR5
```

```
0000027c d3e8        660           out     (scc_cont),a
0000027e 3e6b        661           ld      a,01101011b    ;sdlc crc,
                     662                                  ;txcrc enable,
                     663                                  ;reset rts
00000280 d3e8        664           out     (scc_cont),a
                     665
                     666                                   ;
                     667                                  ;**start counting out 2 flag character times **
                     668                                   ;
                     669                                  ;count 16 bit times
                     670                                  ;from the rs-422 enable
                     671                                  ;for 2 flags.
                     672                                  ;btdelay=subr delay+ctc1int+polling=8bits
                     673                                  ;16 bit times-btdelay=16-8=08h
                     674                                  ;
00000282 0e08        675           ld      c,08h
00000284 cdWwww      676           call    bittime        ;bittime delay
                     677                                  ;is stored in reg.c
                     678                                  ;and bit1 of timflg
                     679                                  ;will indicate
                     680                                  ;count termination.
                     681
00000287682 l6:                                           ;timer flag
00000287 3aWwww      683           ld      a,(timflg)     ;
0000028a cb4f        684           bit     1,a            ;if bit1=1 then
                     685                                  ;count terminated
0000028c 28f9        686           jr      z,l6           ;
0000028e cb8f        687           res     1,a            ;reset timflg bit1
00000290 32Wwww      688           ld      (timflg),a     ;update timflg
                     689                                   ;
00000293 3e03        690           ld      a,03h          ;
00000295 d3e5        691           out     (ctc1_cont),a  ;disable int,
                     692                                  ;software reset
                     693                                  ;to kill the counter1
00000297 0602        694           ld      b,02h          ;2+1 bytes to transmitted
00000299 21Wwww      695           ld      hl,txlapenq    ;point to txlapenq buffer
                     696                                  ;send 1st byte
0000029c 7e          697           ld      a,(hl)         ;
0000029d d3e9        698           out     (scc_data),a   ;and send it
0000029f 23          699           inc     hl             ;point to the next byte
                     700                                   ;
000002a0 3ec0        701           ld      a,0c0h         ;reset eom latch command
000002a2 d3e8        702           out     (scc_cont),a
                     703                                  ;
000002a4 f3          704           di                     ;disable all int
000002a5 3e0a        705           ld      a,0ah          ;select WR10
000002a7 d3e8        706           out     (scc_cont),a
                     707                                  ;idle with 1's
                     708                                  ;at the end of the frame
000002a9 3ee8        709           ld      a,11101000b
000002ab d3e8        710           out     (scc_cont),a
                     711                                  ;
000002ad 3e00        712 txq2:     ld      a,00h
000002af d3e8        713           out     (scc_cont),a   ;rr0
000002b1 dbe8        714           in      a,(scc_cont)   ;read rr0
000002b3 cb57        715           bit     2,a            ;read tx buffer empty
000002b5 28f6        716           jr      z,txq2         ;loop if zero
000002b7717 txq1:
000002b7 7e          718           ld      a,(hl)         ;
000002b8 d3e9        719           out     (scc_data),a   ;and send it
000002ba 23          720           inc     hl             ;point to the next byte
                     721
```

```
000002bb 10f0          djnz    txq2            ;loop until all
                                               ;bytes have been
                                               ;transmitted.


000002bd 3e28          ld      a,028h          ;reset tx int pending
000002bf d3e8          out     (scc_cont),a
                                               ;note:tx buffer
                                               ;empty happens as tx
                                               ;shifter is loaded.
                                                ;
                                               ;count= last byte+
                                               ;crc+flag+12bit times-btdelay
                                               ;btdelay=subr delay+ctc1int+polling=8bits
                                               ;8+16+8+12-8=36=24h
000002c1 0e24          ld      c,24h
000002c3 cdWwww        call    bittime         ;bittime delay
                                               ;is stored in reg.c
                                                ;
000002c6 l7:                                   ;timer flag
000002c6 3aWwww        ld      a,(timflg)      ;
000002c9 cb4f          bit     1,a             ;if bit1=1 then count finish
000002cb 28f9          jr      z,l7            ;
000002cd cb8f          res     1,a             ;reset timflg bit1
000002cf 32Wwww        ld      (timflg),a      ;update timflg
                                               ;
000002d2 3e03          ld      a,03h           ;
000002d4 d3e5          out     (ctc1_cont),a   ;disable int,software reset
                                               ;to kill counter
                                               ;****disable rs-422 driver after 12 to 18 1's*****
000002d6 3e05          ld      a,05h           ;select WR5
000002d8 d3e8          out     (scc_cont),a
000002da 3e60          ld      a,01100000b     ;disable tx, set rts
000002dc d3e8          out     (scc_cont),a

000002de 3e03          ld      a,03h
000002e0 d3e8          out     (scc_cont),a    ;WR3
000002e2 3ecd          ld      a,0cdh
000002e4 d3e8          out     (scc_cont),a    ;8b/char,rx crc enabled,
                                               ;address search and rx enabled

                                               ;************************************

                                               ;count for the interframe gap
                                               ;of 200 usec or 46 bit times.
                                               ;btdelay=subr delay+ctc1int+polling=8bits
                                               ;46 - btdelay=46-8=26h
                                               ;note that timflg will be polled in
                                               ;the main routine.
                                               ;
000002e6 0e26          ld      c,26h
000002e8 cdWwww        call    bittime
                                               ;
                                               ;bittime delay is stored in reg.c
                                               ;************************************
000002eb e1            pop     hl              ;restore
000002ec c1            pop bc                  ;restore
000002ed f1            pop     af              ;restore status and a reg
000002ee c9            ret
```

```
                                                  ;subroutine to time out bit time 4.3 usec per bit
                                                  ;register c contains the number of bits to be
                                                  counted down
                                                  ;**********************************************
000002ef          bittime:
000002ef f5                push      af           ;save status and a reg
000002f0 c5                push      bc           ;save
000002f1 e5                push      hl           ;save
                                                  ;
000002f2 3ed2              ld        a,0d2h       ;ctc1 int vector
000002f4 d3e5              out       (ctc1_cont),a

000002f6 3edf              ld        a,11011111b  ;
000002f8 d3e5              out       (ctc1_cont),a ;enable int
                                                  ;select counter mode
                                                  ;clk/trg edge starts with rising edg
                                                  ;time constant follows
                                                  ;software reset
000002fa 79                ld        a,c          ;reg c contains the number of bits
000002fb d3e5              out       (ctc1_cont),a ;load the number of bits to be counted
                                                  ;**
000002fd e1                pop       hl           ;restore
000002fe c1                pop       bc           ;restore
000002ff f1                pop       af           ;restate status and a reg
00000300 fb                ei
00000301 c9                ret
```

LISTING 3

```
                                                      ;******************************
                                                      ;receive int service routine.
                                                      ;******************************
                                                      ;save received character in receiver buffer
                                                      ;to by rxpointer

0000044d                recint:
0000044d f5                       push     af              ;save af
0000044e d5                       push     de
0000044f e5                       push     hl
00000450 dbe9                     in       a,(scc_data)    ;read scc data
00000452 2aWwww                   ld       hl,(rxpointer)  ;
00000455 77                       ld       (hl),a          ;save it
00000456 23                       inc      hl              ;update pointer
00000457 22Www                    ld       (rxpointer),hl  ;
0000045a ed5bWwww                 ld       de,(rxbufend)   ;end of rx buffer
0000045e af                       xor      a               ;reset cy
0000045f ed52                     sbc      hl,de           ;
00000461 c2Wwww                   jp       nz,recexit      ;if not zero,then receive
                                                           byte length is ok
00000464 21Wwww                   ld       hl,recerrflg    ;
00000467 cbc6                     set      0,(hl)          ;set bit0=1 maxfrmflg to indicate error
                                                           ;because of max frame
                                                           size exceeded.
00000469                recexit:
00000469 3e38                     ld       a,038h
0000046b d3e8                     out      (scc_cont),a    ;reset highest ius
0000046d e1                       pop      hl
0000046e d1                       pop      de
0000046f f1                       pop      af              ;restore af
00000470 fb                       ei                       ;enable int
00000471 c9                       ret                      ;return from int
                                                           ;note ret and not reti is used for scc
                                                           ;interrupts on the z80181.

                                                           ;*********************************************
                                                           ;special receive interrupt service routine
                                                           ;*********************************************
                        ;                                  "parity is special condition" bit is off.
                        ;                                  special conditions are eof or rx overrun error.
                        ;                                  crc error flag is valid only if eof is valid.
                        ;                                  if frame is ok then recerrflg bit1=0, otherwise

00000472                 spcond:
00000472 f5                       push     af              ;save af reg
00000473 c5                       push     bc              ;
00000474 e5                       push     hl;

00000475 3e01                     ld       a,01h
00000477 d3e8                     out      (scc_cont),a    ;read rr1
00000479 dbe8                     in       a,(scc_cont)
0000047b e660                     and      01100000b        ;check bit6 (crc) or bit5 (overrun)
0000047d caWwww                   jp       z,ok            ;
                                  ;
00000480 21Wwww                   ld       hl,recerrflg    ;fetch receive error flag
00000483 cbce                     set      1,(hl)          ;set bit1=1 for frame not ok
00000485 c3Wwww                   jp       crc_exit
```

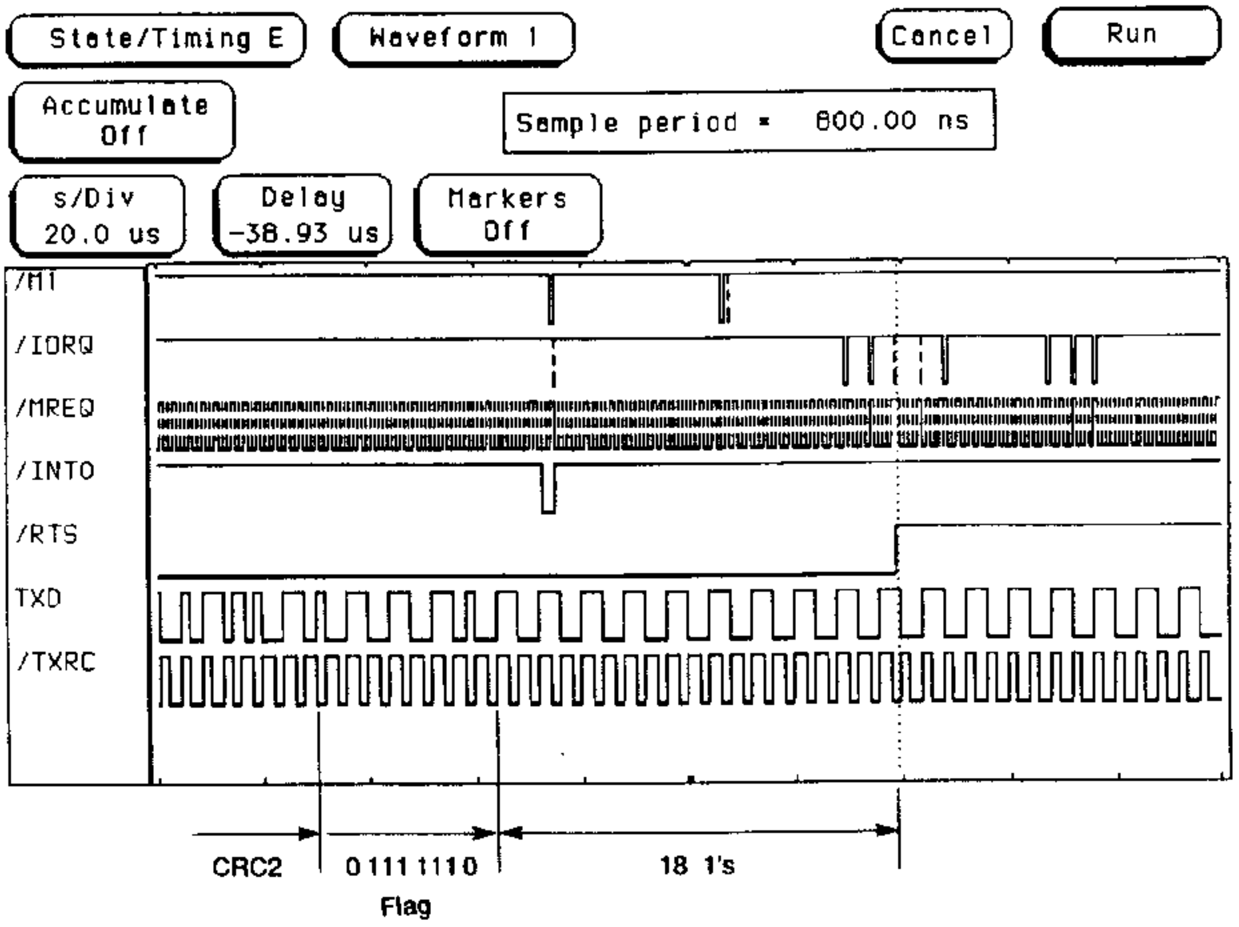
```
00000488                        ok:
00000488 21Wwww                 ld      hl,recerrflg        ;fetch receive error flag
0000048b cb8                    res     1,(hl)              ;set bit1=0 for frame  ok


0000048d                crc_exit:
0000048d dbe9                   in      a,(scc_data)        ;read 2nd crc (debug only) and
0000048f 2aWwww                 ld      hl,(rxpointer)      ;load pointer
00000492 2b                     dec     hl                  ;adjust rx buff ptr for crc1
00000493 2b                     dec     hl                  ;adjust rx buff ptr for crc2
00000494 22Wwww                 ld      (rxpointer),hl      ;

00000497                spexit:
00000497 3e38                   ld      a,038h
00000499 d3e8                   out     (scc_cont),a        ;reset highest ius

0000049b e1                     pop     hl                  ;restore hl
0000049c c1                     pop     bc                  ;restore be
0000049d f1                     pop     af                  ;restore af
0000049e fb                     ei                          ;enable int
0000049f c9                     ret                         ;return from int

```

LISTING 4

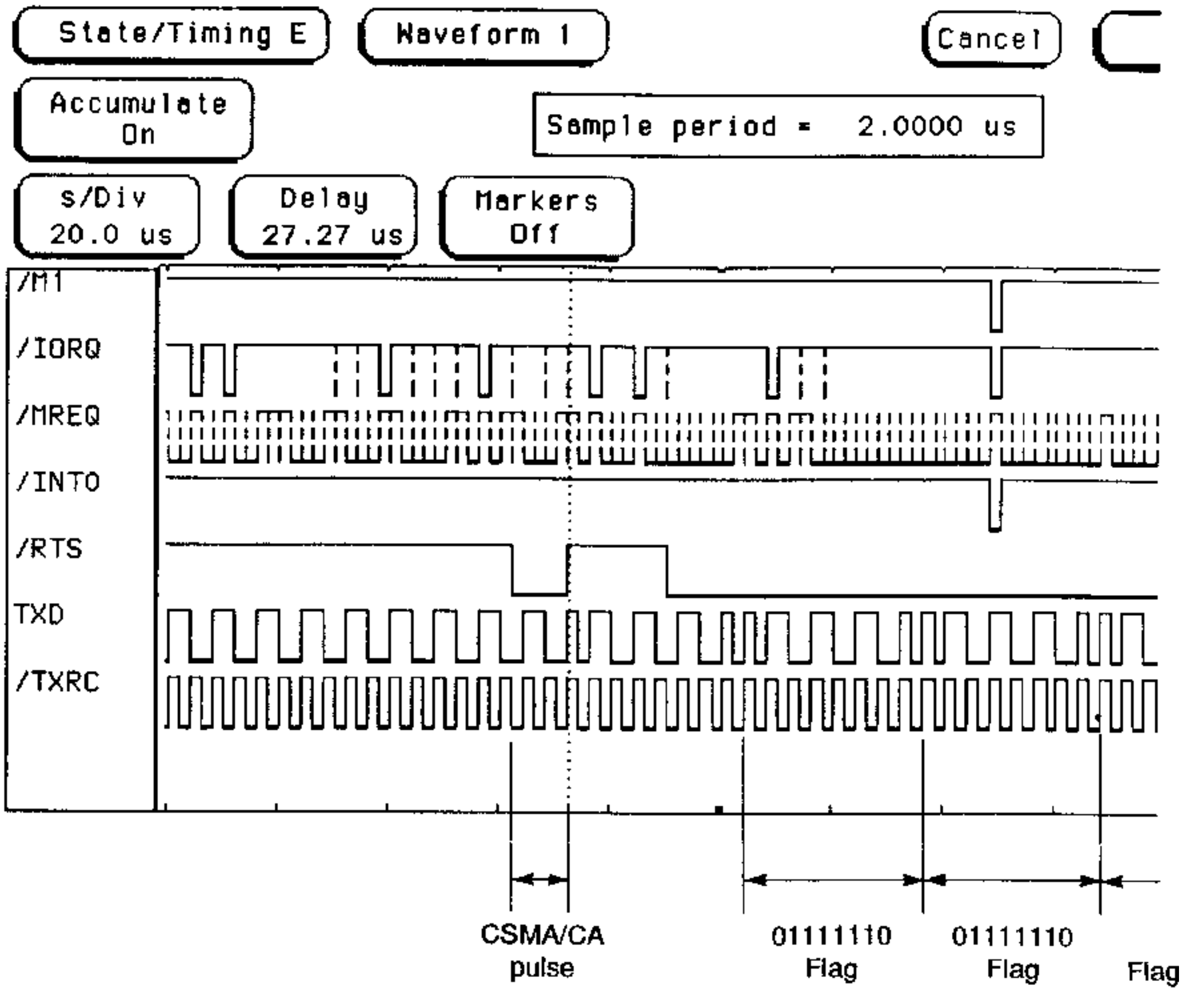
```
                                                              ;****************************************
                                                              ;ctc1 timer int handler
                                                              ;****************************************
00000509                   ctc1int:
                                                              ;ctc1 is programmed in counter mode.
                                                              ;external trigger edges is provided by
                                                              ;/trxc pin at intervals of 4.3 usec.
                                                              ;bit1 of timflg is set when count is terminated.
00000509 f5                          push        af
0000050a c5                          push        bc
0000050b e5                          push        hl
                                                              ;** update the timing flag **
0000050c 21Wwww                      ld          hl,timflg
0000050f 7e                          ld          a,(hl)       ;get recent timflg
00000510 cbcf                        set         1,a          ;bit1=1 after count is over
00000512 77                          ld          (hl),a       ;update the timflg
00000513 e1                          pop         hl
00000514 c1                          pop         bc
00000515 f1                          pop         af
00000516 fb                          ei
00000517 ed4d                        reti



                                                              ;**********************************
                                                              ;interrupt vector table for the scc
                                                              ;**********************************
                                                              ;the status of the interrupt source will affect
                                                              ;the interrupt vector. The interrupt handler's
                                                              ;address are set in a block, as below.
00000a00                             org         sdlc + 0a00h
00000a00                   sccvect:
                                     if          scc_a
00000a00                             .block      8            ;reserve vector for other ch
                                     endif
00000a08 R000+03e9,                  dw          txint        ;tx int
00000a0a R000+04c8,                  dw          ext_stat     ;ext/stat int
00000a0c R000+0433,                  dw          recint       ;rx char int
00000a0e R000+0454,                  dw          spcond       ;sp rec cond int

                                     if          not scc_a
00000a10                             .block      8            ;reserve vector for other ch
                                     endif

00000a18                   temp:     .block      1


                                                              ;**********************************
                                                              ;interrupt vector table for the ctc
                                                              ;**********************************
00000ad0                             org         0ad0h
00000ad0 R000+04d8,                  dw          ctc0int      ;reserved for ctc0 int routine
00000ad2                             org         0ad2h
00000ad2 R000+0509,                  dw          ctc1int      ;reserved for ctc1 int routine

                                                              ;***********************
                                                              ;receive buffer area
                                                              ;***********************
```

```
00001000                      org         1000h
00001000        rx_buff:      .block      length


                                                      ;************************
                                                      ;transmitter buffer area
                                                      ;************************
0000b000                      org         0b000h
                                                      ;
                                                      ;********************************
                                                      ;transmit llap enq packet (3bytes)
                                                      ;********************************
0000b258 ff     txlapenq:     db          0ffh        ;broadcast id
0000b259                      .block      1           ;guess at myaddress
                myaddress
0000b25a 81                   db          81h         ;llap enq type
                                                      ;
```

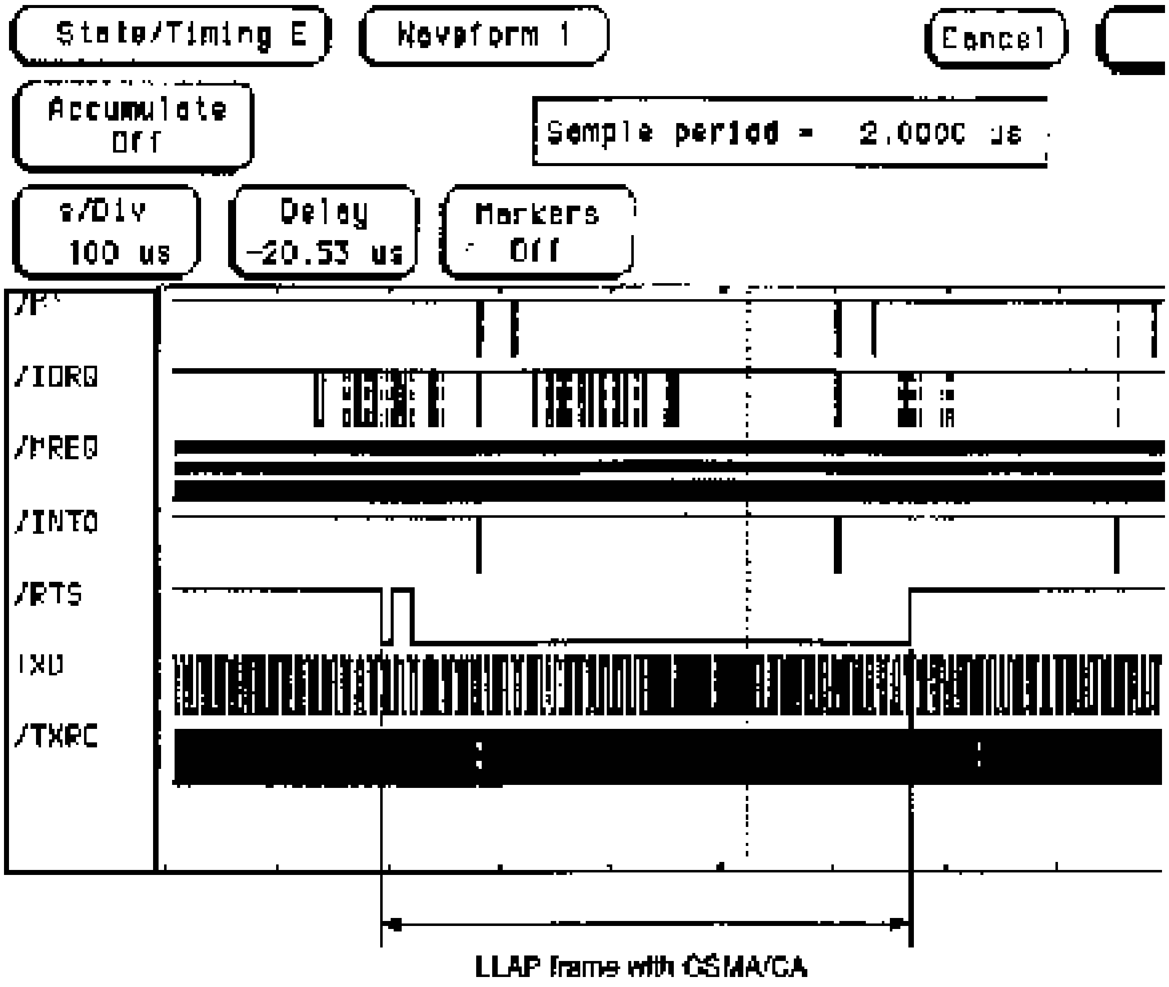
APPENDIX B



12 to 18 1’s at the end of an LLAP frame



CSMA/CA before an LLAP frame



An LLAP Frame