# MULTIPROCESSING FOR THE IMPOVERISHED

## Part 5: Serial I/O

by Brad Rodriguez

This article first appeared in The Computer Journal #70 (Nov/Dec 1994).

At long last, the final installment of the "Scroungemaster II" multiprocessor. I may in the future write some software articles around this board -- a 6809 Forth is certain -- but this wraps up the hardware description. Go, and prototype!

## SERIAL INTERFACES

Figures 1 and 2 [pages **5 and 6** of the PDF file] contain the two UARTs which provide four serial ports for the Scroungemaster II. Since these circuits are nearly identical, I will discuss them together. Afterwards I will mention the differences.

U10 and U11 are Zilog Z8530 Serial Communications Controllers (SCCs). These are *very* versatile chips, each of which includes two serial ports and two programmable baud rate generators, plus some miscellaneous I/O. All of the common asynchronous formats are supported, plus many synchronous ones (see sidebar). The 4 MHz part will transfer data at up to 1 Mbit/second. (I'm told Apple uses this chip for the Appletalk network in the Macintosh.)

U10 is enabled by chip select IO0\, and U11 by IO1\. Each chip looks at only the low two address lines, so each occupies four consecutive locations in memory. As wired here, the SCC registers appear as follows:

```
U10   (U11)   address register
FFC00 (FFC80) Port B (D) command
FFC01 (FFC81) Port B (D) data
FFC02 (FFC82) Port A (C) command
FFC03 (FFC83) Port A (C) data
```

(Remember that these are addresses in the mapped 1 MB memory space.) The ports of each chip are called A and B in the Zilog documentation. On the Scroungmaster II board I refer to U10 as ports A and B, and U11 as ports C and D. If only two serial ports are needed, U11 can be removed.

The serial ports of most personal computers use signal levels adhering to the Electronics Industries Association (EIA) RS-232 standard. This standard specifies a voltage of -5 to -15 volts for logic "1", and +5 to +15 volts for logic "0" [PIP85]. U26 and U25 generate and receive RS-232 levels for the Scroungemaster II. U26 is an LM1488 quad RS-232 driver, and U25 is an LM1489 quad RS-232 receiver. Only the Receive Data (RXD) and Transmit Data (TXD) from each serial port are converted to RS-232 levels. (The RS-232 standard specifies a number of control signals, such as DCD, DTR, DSR, RTS, and CTS, all of which must use these same logic levels. This board does not support these signals.)

Note that if *and only if* RS-232 drivers are used, +12v and -12v power supply voltages must be provided. (Everything else runs on +5v only.)

RS-232 is limited to short cable runs, relatively low data rates, and point-to-point connections. To allow higher data rates or longer cables, EIA defined the RS-485 standard. RS-485, an improved version of the earlier RS-422, uses balanced transmission, which means that each signal is carried on a *pair* of wires: one wire carries the "true" signal, and the other wire carries its logical inverse. Signal levels are approximately 5 volts and 0 volts, but the detailed specifications of RS-485 require the use of special driver chips and not just TTL outputs. The Scroungemaster II uses a 75174 quad RS-485 driver (U13), and a 75175 quad RS-485 receiver (U12). Again, only the serial RXD and TXD are converted to these levels.

*Important:* On this board, the choice of RS-232 vs. RS-485 is made by which pair of driver chips you install. This means that the four ports may be all RS-485 or all RS-232, but you can't have a mix of RS-485 and RS-232. Do *not* install both sets of drivers! Install only U12/U13, or U25/U26.

Another advantage of RS-485 is that it allows *multidrop* operation, in which several serial ports are connected to one pair of wires. When you connect a network of CPUs in this way, any CPU can talk to any other (and you can also save a lot of wire!). For this to work, though, only one CPU can drive the wires at any one time. The other CPUs must have their RS-485 drivers *disabled*, i.e., in a high-impedance state, so they neither drive nor load the serial "bus." (The problem of deciding *when* each CPU can enable its driver is worth several more articles, and I won't talk about it here.)

The 75174 chip (U13) is somewhat unusual in that its RS-485 drivers are enabled in pairs. (There are only two enable inputs for the four drivers.) This means that you can't use all four Scroungemaster II serial ports for multidrop, since each multidrop port needs an independently controlled transmitter. I've arranged the drivers so that ports A and C are enabled together, and likewise B and D. This means that if you have only one SCC installed (U10), you have two fully-functional multidrop RS-485 serial ports. If you install both SCCs, you can run all four ports RS-485 point-to-point (since for point-to-point operation you leave the driver always enabled). Or you can run two ports point-to-point and one port multidrop (e.g. B and D point-to-point, A multidrop, with C unusable because it's driver is enabled according to A's needs).

The A and C drivers are enabled when the RTSA\ output of U10 is high. The B and D drivers are enabled by RTSB\ of U10. (Thus U11 is optional in a two-port system.) The RTS\ outputs must be controlled by software. The RS-485 receivers are always enabled.

The serial signals are brought out on 10-pin headers such that two ports may be connected together by putting a half twist in the ribbon cable:

```
pin signal     signal pin

1     CTS\ <-> RXRDY\ 10
2      RXC <-> TXC    9
3      GND <-> GND    8
4      RX- <-> TX-    7
5      RX+ <-> TX+    6
6      TX+ <-> RX+    5
7      TX- <-> RX-    4
8      GND <-> GND    3
9      TXC <-> RXC    2
10  RXRDY\ <-> CTS\   1
```

In other words, a half twist serves as a null modem. (If RS-232 signals are used, the signals appear on the TX- and RX- pins.)

TXC and RXC are the serial clock output and input, respectively, of the serial port. (Strictly speaking, TXC can be programmed to be either an output *or* an input, but only the former is useful here.) If the TXC of one port is connected to the RXC of another, the two ports can use the x1 clock mode, and all synchronous modes. This is intended for 1 Mbit/sec operation between adjacent CPUs. TXC is not buffered, so the cable length can be only a few inches.

Inverters U9C through U9F generate "Rx ready" signals that I hope will allow automatic flow control between two connected serial ports, as follows: The SCC's W/REQ\ pin can be programmed to go low when the receive buffer contains a character ("DMA Request on Receive" mode). Thus, when the receive buffer is empty, W/REQ\ is high and RXRDY\ is low. This is connected to the CTS\ input of the "sending" SCC, which should be programmed in the "Auto Enables" mode. In this mode, a low on CTS\ enables the transmitter, and a high disables the transmitter (after it finishes sending any character in progress). This scheme is still to be tested, but if it works, it will prevent one serial port from "overrunning" the other...very handy at 1 Mbit/second! Note that the RXRDY signals are not buffered, and, like the clock signals, are only usable over short distances.

Only the RX and TX signals should be run over long distances (up to 50 feet for RS-232, or 4000 feet for RS-485). In the absence of the clock signals, use a x16 or x64 asynchronous mode, or use the DPLL for clock recovery in synchronous modes. (For further details refer to the Zilog technical manual.) Flow control must be done in software.

You can run the clock and flow control signals long distances if you provide suitable external drivers (RS-232 or RS-485). You may also want to add external logic functions or level translation (say, to produce a MIDI port). To facilitate this, pin 10 of the serial connector can be jumpered to +5 volts (JP10-JP13). Of course, you lose the RX Ready signal if you do this.

The DTR\ pins of the SCC can be used as general-purpose outputs. These two outputs from U10 provide as the the TASK and SWREQ\ control signals, used elsewhere in the Scroungemaster II. (Another reason why you should keep U10, and not U11, if you need only two serial ports.)

The DCD\ pins of the SCC can be used as general-purpose inputs. These can also be programmed to generate an interrupt whenever the level changes in either direction. So, these two inputs on U10 are used for SWGRANT\ and IRQ4\. But in the Auto Enables mode, the DCD\ inputs must be grounded to enable the SCC receivers. This can be done with JP14 and JP15 (disabling the SWGRANT\ and IRQ4\ functions.)

The DTR\ and RTS\ pins of U11 have no assigned function, and so are brought out to a 4-pin header. The DCD\ pins are permanently grounded on U11.

JP7 and JP8 control interrupt assignments of the SCCs. Along with JP9 and JP4, which control the PIO and PC bus interrupts (respectively), they allow the following possible assignments:

```
SCC#1  SCC#2  PIO  bus interrupt

        IRQ   IRQ  IRQ
FIRQ   FIRQ
NMI          NMI
                   IRQ4 (via U10)
```

All three I/O chips may have distinct interrupts, which makes for the simplest interrupt routines. Either or both SCCs may use the 6809's fast interrupt (FIRQ) for high data rates; but of course, when two chips share an interrupt, your software must poll them to identify which is generating the interrupt. Also, if a chip is connected to the Non-Maskable Interrupt (NMI), its interrupt cannot be disabled by the CPU -- it only be disabled by writing to that chip's interrupt control register. (Both the SCC and the PIO have such a register.)

If you are using PC bus interrupts and all three I/O chips, then two of the four interrupt sources must share one of the three 6809 interrupts (and polling must be used). Or, you can route the PC bus interrupts to "IRQ4" (U10's DCDB\ input), in which case PC bus interrupts will be processed through U10's interrupt logic.

## LED DISPLAY

Whenever I design a CPU board, I always include at least one LED -- the most useful tool for bringing up a new (or dead) board. By popular demand, the Scroungemaster II has *eight* programmable LEDs, in a 10-segment bargraph, D1. (The middle two segments are used as a power indicator.) Octal latch U23 (74HCT377) serves as an 8-bit parallel output port, selected by IO3\ (mapped address FFD80). When a "0" is written to any bit of this output port, the corresponding LED will illuminate. For simplicity, the LEDs share a single current limiting resistor, so the more LEDs you turn on, the dimmer they get. A 74HCT377 is recommended instead of a 74LS377 because of the HCT part's greater current capacity.

## PROGRAMMABLE INTERRUPT GENERATOR

One of the experiments I want to perform with my multiprocessor system is having one processor interrupt another, as a method of signalling. U29 and U30, if installed on one CPU board, make a programmable interrupt generator. The comparator U29 (74LS688) identifies when a write occurs to any I/O address 000-007 on the IBM PC bus. (In IBM PCs, these addresses are reserved for the motherboard, so we know no plug-in peripheral card will ever use them.) When such a write occurs, the programmable latch U30 (74LS259) writes the data bit XD0 into one of eight latches. Which of the eight is selected by address bits XA0-XA2.

The eight latches output to interrupt lines IRQ0-IRQ7. Each of the eight possible CPUs can be jumpered to one of these lines. So, if CPU #5 is jumpered to receive IRQ5, you can interrupt this CPU by writing a "1" to PC bus I/O address 005. (Recall that this appears as mapped address DFC05 to the 6809. Also, PC bus interrupts are active *high*, so writing a "1" asserts the interrupt, and a "0" removes the interrupt.) Since U29 and U30 are electrically connected directly to the IBM PC bus, and not to the board's "internal" bus, *any* CPU can write to them. So, by installing U29 and U30 on one board, and jumpering the interrupts appropriately, any CPU can interrupt any other.

Normally, U29 and U30 will be installed on the bus master. *These should not be installed if IBM PC peripheral cards will be generating interrupts.* This is because U30's outputs are always active -- like many PC perhiperals -- and if one board is pulling an interrupt line high while another pulls it low, someone's drivers will be damaged.

## TIMING DIAGRAM

When designing microprocessor circuits, it's not enough to get the voltage levels and the logic right. You have to get the *timing* right, too. Timing specifications are published by the manufacturer (usually in the data books, e.g. [MOT83, TEX85, ZIL88]) for all microprocessor and logic chips. These specs describe:

a) when each input is *required*,
b) when each output is *valid*, and
c) how long the chip will take to perform a given function.

While automated tools exist which will analyze the timing of any digital circuit, many engineers still do it manually by constructing a *timing diagram*.

Figure 3 is a partial timing diagram for the Scroungemaster II. The timing is shown for the fastest CPU (68B09 with an 8 MHz oscillator) and the slowest recommended "glue" logic (74LS TTL). This is usually the "worst case" for timing analysis, since the

CPU imposes the "deadlines" while the glue logic introduces the "delays." If any part of the logic adds too much delay, I can replace chips with a faster equivalent (74ALS, 74AS, or 74F TTL). *Important:* although supposedly equivalent, many 74HCT parts are actually slower than 74LS. If you want to use 74HCT, verify the timing first. Better still, use the faster 74HCTLS, 74ACT, or 74AHCT families. Also, you may be able to substitute "plain" 74 or 74S family parts in some places, but beware of their much higher power consumption.

Each line of Figure 3 represents one signal (or a group of signals with identical timing). The horizontal axis is time; each division is 50 nsec. Only the most critical signals are shown. For many devices (e.g. RAM and EPROM) you can compare the timing requirements of the chip directly with this diagram -- you don't need to draw every signal.

The first line is the 6809 E signal. The "falling edge" of E (from +5V to 0V) is the start of a 6809 memory cycle, and is the "reference point" for most of the timing specs, so I have drawn the diagram with time T=0 at the falling edge. Note that, according to Motorola specs, E could be low for as little as 210 nsec. Thus I've drawn the rising edge of E to span a range from T=210 to 250 nsec. During this "window", E could rise at any time, so we *don't know* for sure whether E will be high or low.

The second line is the 6809's "quadrature" clock signal, Q. If the falling edge of E is T=0, then Q will rise between times T=80 and T=125 nsec. Although the Scroungemaster II doesn't use the Q signal, some of the 6809 signals are specified relative to Q, so it needs to be on the diagram.

The 68B09's address and R/W\ lines are guaranteed to be valid and stable 15 nsec before the rising edge of Q. They remain stable throughout the cycle, and 20 nsec into the next cycle, i.e., 20 nsec after the *next* falling edge of E. (Obviously, for the first 20 nsec of *this* cycle, address and R/W\ carry "old" data.) On the diagram, I have "X'ed out" the period when these lines are changing (and thus unknown). Also, when they are stable they could be high *or* low -- the address could be anything! -- so I've drawn *both* signal levels (0 and +5) during this period. These are common conventions used in timing diagrams.

The 68B09 requires that data read from memory or I/O be valid 40 nsec before the start of the falling edge of E. Because of E's rise and fall times, this edge could be as soon as T=470 nsec, so data must be stable at T=430 nsec. Also, this data must remain stable until 10 nsec after the end of the falling edge of E, i.e., until T=510 nsec. This is a *need* -- a requirement to be satisfied.

The 68B09 guarantees that data written to memory or I/O will be valid 110 nsec after the rising edge of Q, and will remain valid until 30 nsec after the next falling edge of E. On this diagram, this is from T=235 to T=530 nsec. This is a *given* -- a known timing characteristic of a signal.

Now we can analyze the Scroungemaster II circuit. First, look at the memory mapping and decoding schematic. All of the address and control inputs to the 74S189's are stable at T=110 nsec. The 74S189 takes at most 35 nsec to access its data, so its outputs (and thus the "mapped address") are valid at T=145 nsec.

The WRMAP\ and OEPROM\ signals are derived from A15, R/W\, and E. You can see that A15, R/W\, and the inverted R/W\ (U7D) will be stable long before E goes high. So E is the deciding factor, and the WRMAP\ or OEPROM\ signal will go low 10 to 15 nsec after E goes high (assuming that the CPU is writing the map or reading the EPROM, respectively). WRMAP\ or OEPROM\ will remain low until 10-15 nsec into the next cycle. This is the propagation delay of a 74LS10 NAND gate.

The second stage of decoding, U4 and U8A, produces the signals IOZONE\ and ONBOARD\. These signals are logically derived from the CPU address and the mapped address, and we know the CPU address must be stable before MA12-MA19 can be stable. So IOZONE\ and ONBOARD\ will be stable at T=160 nsec, 15 nsec (the maximum delay of the NAND gates) after the mapped address. In this case we're not worried about the minimum delay of the NAND gates (which would tell us how *soon* IOZONE\ and ONBOARD\ could *possibly* be asserted). Both of these signals will remain stable until the address changes, at least 20 nsec into the next cycle -- which, we will see, is sufficient.

IOZONE\ is the last input of U24 to become stable, so U24's outputs -- the IOn\ select lines -- will follow IOZONE\ by 32 nsec, the maximum delay of the 74LS138. Similarly, the OFFBD\ signal (U6D) follows ONBOARD\ by 15 nsec.

The read and write strobes, however, are not output by U5 until E goes high. You can see from the diagram that all the other inputs of U5 will be stable before the rising edge of E. So, the various read and write strobes will trail E by 14 to 26 nsec (the propagation delay from the 74LS138's G1 input to its outputs). In this case we want to know upper *and* lower bounds on the strobes.

The next two lines show transitions propagating through the memory request logic. If the bus is not available, MRDY will be pulled low no later than time T=250 nsec. For MRDY to stretch a memory cycle, the 68B09 needs to see it pulled low no later than T=360 nsec (110 nsec before the falling edge of E). You can see that, even in the worst case, we have 110 nsec to spare!

If we're doing a three-cycle stretch, we can't have glitches on the REQ signal (NAND gate U6A). To ensure this, STRETCH\ must go low before OFFBD\ goes high. STRETCH\ will go low no later than T=520 nsec (20 nsec after the falling edge of E).

We know that OFFBD\ follows ONBOARD\, which follows the mapped address, which follows the 68B09 address, which remains valid 20 nsec into the next cycle...so there's no problem.

We can control (via JP6) how much extra delay is added to a bus cycle when the 6809 has to wait for ownership. What happens if this CPU already owns the bus? The next six lines of the timing diagram show that DRIVENBL\ is asserted (low) at T=205 nsec. The address will then be output to the bus no later than T=235 nsec, and the bus RD\ and WR\ strobes will be asserted no later than T=310 nsec, and will be at least 185 nsec wide. Read data must be valid *on the bus* no later than T=420 nsec, to allow for a 10 nsec delay through the bus buffer. Write data will be valid *on the bus* no later than T=245 nsec. Finally, if a bus peripheral (such as a CGA card) wants to stretch the memory cycle, it must assert IORDY no later than T=330 nsec. (Does this meet the specs of PC peripherals? Who knows? Only experimentation will tell.)

We now have enough information to determine if any given memory or I/O chip will work with the Scroungemaster II CPU. The first example shown is the Z8530 SCC chip. (This analysis is also valid for the Z8536 parallel I/O chip, since it has nearly identical timing requirements.) For convenience, I have duplicated the lines showing the read and write strobes and the IOn\ selects.

The Z8530 requires that its address lines be valid from T=125 to T=520 nsec. Since A0-A1 come from the CPU, we can see that this requirment is met.

If this chip is to be accessed, the Z8530's CE\ input (IO0\ or IO1\) must go low before RD\ or WR\ go low, and must stay low as long as RD\ or WR\ is low. (As Zilog specs it, CE\ must go low at least 0 nsec before RD\ or WR\ goes low, and must stay low at least 0 nsec after RD\ or WR\ goes high.) This requirement is met with time to spare. If this chip is *not* to be accessed, CE\ must be high at least 100 nsec before RD\ or WR\ goes low...which means that the IO0\ or IO1\ select must not stretch more than 125 nsec into the next cycle! Again, no problem.

The 8 MHz Z8530-8 will output read data no later than T=430 nsec (140 nsec after its RD\ goes low), which exactly matches the 68B09's need. This data will remain valid at least 14 nsec -- the delay of U5 -- after E falls; the 68B09 needs it to be held only 10 nsec after the fall of E. This is a case where using a faster logic family (for U5) could theoretically *introduce* a timing problem!

The Z8530 parts are peculiar, and perverse, in requiring write data to be valid before the *falling* edge of WR\. 10 nsec before, to be precise. If the decoding logic is running near its fastest spec, WR\ could be asserted as soon as T=220 nsec. But the CPU doesn't guarantee write data until T=235 nsec! This 15 nsec discrepancy can be resolved several ways:

> a) hope that the 68B09 and Z8530 timing specs are sufficiently conservative to cover this 15 nsec discrepancy (e.g., the 68B09 may output data sooner);
>
> b) hope that the decoding logic operates near its nominal timing and not its minimum timing specs (another case where faster logic spells trouble!);
>
> c) add some gates to delay the RD\ and WR\ signals, being careful not to violate the data hold time spec; or
>
> d) use the CMOS Z85C30, which intelligently latches its write data on the *rising* edge of WR\.

So far I've had good luck relying on (a) and (b). If I run into problems, (d) is an easy out. If I were being paid for this I might expend the extra effort to do (c). But note that the write data must be held at least as long as WR\ is low, and that WR\ can go back high as late as T=525 nsec. The 68B09 holds write data until T=530 nsec. Any delaying circuit must therefore delay the falling edge of WR\, but *not* the rising edge!

The second example is the 2 MHz 6522A Versatile Interface Adapter. This was briefly considered for the SM II's parallel I/O. You can see the problem with this chip immediately: it requires that its CS\ be valid at T=130 nsec (80 nsec before the rising edge of E). But the IOn\ select lines are not valid until T=192 nsec! This 62 nsec discrepancy is too big to ignore, and there is no way to fix it short of putting the 6522A into the "unmapped" memory space...a solution I did not want to use.

Observe also that the 6522A provides read data 10 nsec too late for the 68B09. There's no easy way to fix this, since this is a 190 nsec delay from the rising edge of E. If it had been a delay from some other signal, we could try to provide that other signal sooner (e.g., with faster logic). As it is, we can only use a faster part...and a faster 6522 doesn't exist.

I conclude that 74LS logic can be safely used throughout the SM II, even with the faster CPU. I make the (perhaps unjustified) assumption that if 74LS logic is fast enough for the 68B09, the 1.5 MHz 68A09 and the 1 MHz 6809 will pose no problem. In view of the timing discrepancies of the Z8530, I may be too hasty assuming this, and I should do a fresh analysis at 1 MHz. I suspect that with the slower CPUs, the slower 74LS189 can be used instead of the 74S189 in the memory mapping logic. (The last time I checked, though, the 74S189 was still less expensive.) I'll leave these as exercises for the student, and I'll be interested to hear from you.

## TEST PROGRAM

Listing 1 is a simple test program for the Scroungmaster II, written for the PseudoSam Level 1 freeware assembler (A6809). It begins by initializaing the memory mapping registers (which are in a random state on power-up). It then counts in binary from 0 to 255 on the LEDs, initializes one SCC (U10), and enters a loop of receiving and sending characters on serial port A. The serial port will run at 4800 baud with the basic 4 MHz oscillator, or 9600 baud with an 8 MHz oscillator (requiring 68B09 CPU and Z8530-08 SCC). Either an 8Kx8, 32Kx8, or 128Kx8 static RAM must be installed for the serial initialization subroutine to work.

## SIDEBAR: SERIAL DATA FORMATS

In parallel data transmission, such as on the PC bus, several data bits are put on several wires at the same time, and there are usually additional wires to control the timing of the transfer. When you want to send data over a *single* wire, you have to send the data bits one at a time, or *serially*. Serial transfer opens a whole new can of timing and control problems.

Usually, the bits will be placed on the wire one at a time, for a fixed period, starting with the least-significant bit (D0). The *baud rate* is the reciprocal of the bit period, so that at 9600 baud, each bit is placed on the line for 1/9600th of a second. (For all our purposes, anyway; you can find a more detailed discussion in [ARR93].) Both ends of the serial link use a crystal-controlled clock to measure the bit period accurately.

But when does this bit period start...and in a continuous stream of data, how do we identify the first bit of a new byte?

In *asynchronous* transmission, each byte can be sent independently ("asynchronously"). Between bytes, the wire is held at a steady logic "1". To signal the start of a byte, a logic "0" is placed on the wire for exactly one bit period. This is called the "start bit." The 1-to-0 transition marks the "edge" of the bit period, and all succeeding bit periods can be timed from that reference point. Even if the two ports' clocks are a few percent different, the accumulated error over eight bits is small enough that the bits can be recovered. But over several bytes the error will build up, so after each byte a logic "1" is output for one or two bit periods -- one or two "stop bits." Having a stop bit means that the next byte will always start with a 1-to-0 transition, resynchronizing the bit timing. (Two stop bits were required in the days of mechanical teletypes. Electronic serial ports get by fine with just one.)

From five to eight data bits can be sent between the start and stop bits (least-significant bit first). After the data bits, and before the stop bit, can be an additional bit called the *parity* bit. The parity bit is designed to make the total number of "1" bits either Even or Odd. If one bit is received incorrectly, the parity will be wrong, causing a "parity error." (The parity scheme thus detects all "one-bit errors." If two bits get flipped, the parity is unchanged.) Sometimes the parity bit is forced to "0" ("Space") or "1" ("Mark"), or is omitted entirely ("None"). So, when you see an IBM PC set up for "9600,N,8,1", you can now read that as "9600 baud, No parity, 8 data bits, 1 stop bit."

In *synchronous* transmission, the transmitter clock is connected to the receiver, and the bytes are sent in a continuous uninterrupted stream, without start and stop bits. Once the starting point is found, we simply read every eight bit periods as a new byte -- there had better not be delays between the bytes! To identify the starting point we send a *synchronization (sync) character* -- a unique bit pattern that we can detect as bits are shifted through the receiver. (It's also possible to use a separate sync signal, but this is uncommon.) One advantage of synchronous transmission is that it's 20% faster -- a byte is sent in eight bit periods, and not ten (eight data + one start + one stop).

Sometimes you don't need to physically connect the transmitter clock to the receiver. A digital phase-locked-loop (DPLL) such as contained in the Zilog SCC can synchronize the receiver almost perfectly to the transmitter, provided that there are frequent 1-to-0 or 0-to-1 transitions in the data. There are several ways to ensure this. One software scheme is group-coded recording (GCR), used in Apple floppy disks. One hardware scheme supported by the SCC is the synchronous data-link control (SDLC) protocol.

The hardware can also add extra transitions to the bit stream, to make "clock recovery" easier. Among the methods are FM (frequency modulation, also called "bi-phase" encoding), MFM (Modified FM), and Manchester encoding. (MFM is used for double-density floppy disks and many hard disks.) The details are too long to relate here; just be aware that the Zilog SCC can send and receive FM, and can receive Manchester encoded data. [ZIL86]

The bit timer (in asynchronous modes) and the DPLL (in synchronous modes) run from a crystal-controlled timebase (a programmable counter called the "baud rate generator"). If the timebase is 16 times faster than the bit period, then the bit period can be counted off in 16 steps -- we can say that the timing resolution is 1/16th of a bit. This is called a "16x" (16-times) clock.

In its various modes, the Zilog SCC can use a 64x, 32x, 16x, or 1x clock. If a 1x clock is used, the receiver has to be synchronized perfectly to the transmitter (by a physical connection), since the timer can't be "tweaked" in fractions of a bit. This is why the Scroungemaster II serial ports allow the receiver to be connected to the transmitter clock.

## REFERENCES

[ARR93] American Radio Relay League, The ARRL Handbook for Radio Amateurs, ARRL, Newington, CT (1993). A new edition is published every year. One of the best references for electronics and radio, but rather weak on computers and digital stuff -- except for communications, naturally. For some reason the ARRL thinks that "baud" has a plural, "bauds."

[MOT83] Motorola Inc., Motorola 8-Bit Microprocessor and Peripheral Data (1983).

[PIP85] Pippenger, Tobaben, et al., Linear and Interface Circuits Applications, Volume 2: Line Circuits, Display Drivers, Texas Instruments (1985), ISBN 0-89512-185-9. An excellent sourcebook which, alas, may no longer be available from Texas Instruments. Well worth finding.

[TEX85] Texas Instruments Inc., The TTL Data Book, Volume 2 (1985).

[ZIL86] Zilog Inc., Z8030 Z-BUS SCC / Z8530 SCC Serial Communications Controller Technical Manual (September 1986). Essential if you want to do anything fancy with the SCC.

[ZIL88] Zilog Inc., Z8000 Family Data Book (November 1988).