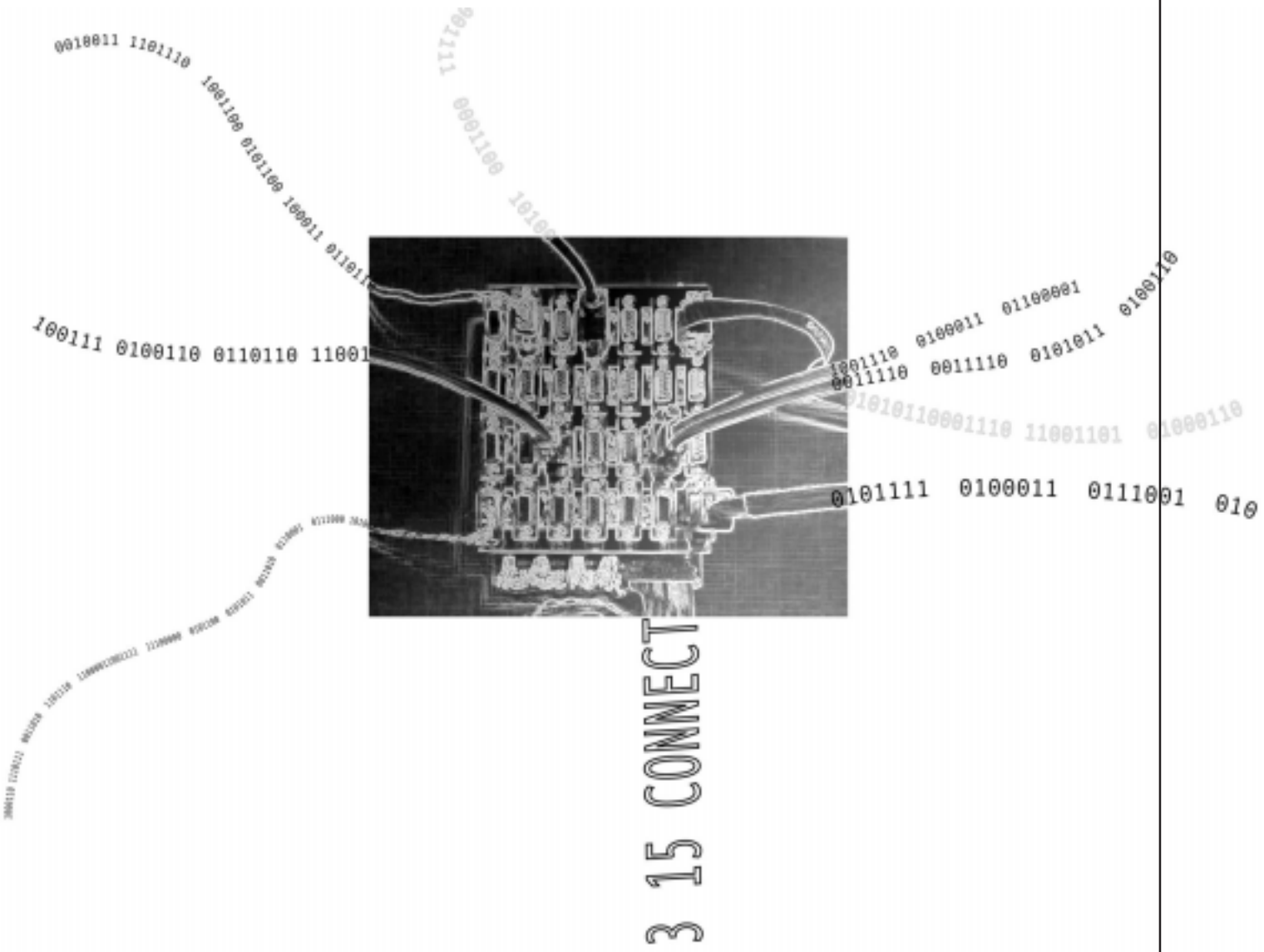
SEXBAR:

By Steven K Roberts

A SERIAL CROSSBAR SWITCH NETWORK



The complete design for a microprocessor-controlled, 32-channel serial crosspoint switch that can provide up to 4 simultaneous bidirectional connections, automatic cable polarity detection, and display of current network status.

Who is Nomadic Research Labs?

Throughout this publication, there are various references to *BEHEMOTH* and *Microship*, as well as occasional mention of sponsors and volunteers. If none of this rings a bell, we include this brief backgrounder to put it all in context...

In 1983, Steven K. Roberts was slaving away in the midwest as a freelance writer and industrial control systems consultant. He had been an electronics hobbyist since well before the beginning of the microcomputer revolution, and was sinking more and more into the torpor of routine business... in *Ohio* of all places.

One day he had the strange idea of combining all his passions into a lifestyle: travel, adventure, electronics, computers, publishing, bicycling, romance... and avoiding work. The obvious solution was to computerize a recumbent bicycle and travel full time while writing for a living, so that's exactly what he did (not expecting it to become a career).

The adventure immediately struck a chord. Sponsors started donating products, the media kept up a steady stream of stories, and volunteers haunted various "bikelabs" to lend a hand. By 1991, the once simple *Winnebiko* had morphed into *BEHEMOTH*, a 580-pound, 105-speed machine with handlebar chord keyboard, head mouse, 72-watt solar array, satellite Internet link, audio crossbar, console Macintosh, SPARCstation, three PC's, helmet cooling system, pneumatically retractable landing gear, robust ham radio station, and quite a bit more. But the acronymic moniker was prophetic: *Big Electronic Human-Energized Machine... Only Too Heavy.* After 17,000 miles on the road, Roberts started dreaming of a technomadic life with no hills.

The obvious solution was a move to water, and for more than 5 years he worked on his *Microship* in various labs... finally setting up a permanent shop on a wooded island in Puget Sound. There, along with his partner Lisa, he brought the dream of canoe-scale pedal/solar/sail micro-trimarans to fruition... in the process spinning off a variety of technical monographs (like this one) in an attempt to augment his unpredictable income via Technology Transfer. These publications reflect the contributions and inventions of industry experts, engineering students, and corporate sponsors... and represent a huge R&D investment that in any normal company would lead to products. But not at Nomadic Research Labs... Roberts is every bit as restless as ever, and manufacturing clones of *Microship* subsystems would drive him quite mad with wanderlust.

The result of all this is low-cost publication of complete, tested designs for interesting gizmology that is not available off-the-shelf. Some of these have distinct product potential, and NRL welcomes inquiries from interested entrepreneurs.

For ongoing information on the Microship techno-adventure, see <http://www.microship.com> or contact wordy@qualcomm.com to request addition to the Nomadness mailing list.

Contents

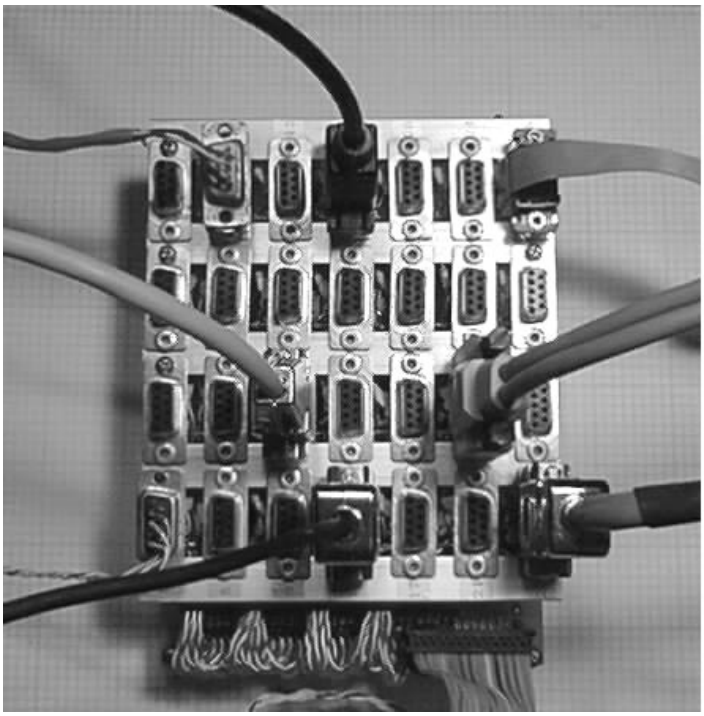
Who is Nomadic Research Labs? 2
1.0 A CROSSBAR BACKGROUNDER 8
1. SEXBAR SYSTEM DETAILS 9
1.1. IMPLEMENTATION NOTES 10
1.2. CIRCUIT DESCRIPTION 12
1.3. RS-232 POLARITY SENSOR 13
1.4. SEXBAR SOFTWARE DISCUSSION 14
1.5. SPECIAL NOTE ON LED BLINK TASK 18
1.6. SEXBAR PACKAGING 19
1.7. SHARED SEXBAR/AUXBAR INTERFACE 21
1.8. NOTES, HACKS, AND COMMENTS 21
2.1. Sexbar FORTH Listing 23
2.2. Multitasker FORTH Listing (optional) 29
SEXBAR SCHEMATICS 32
A. ACKNOWLEDGMENTS 38
B. REFERENCES 39

A Nomadic Research Labs Publication
First Published 2/99
Design, layout and photography
by Lisa Roberts

Sexbar: A Serial Crossbar Switch Network

by Steven K. Roberts
Nomadic Research Labs

The complete design for a microprocessor-controlled, 32-channel serial crosspoint switch that can provide up to 4 simultaneous bidirectional connections, automatic cable polarity detection, and display of current network status.



Front view of Sexbar with connectors

NOTE: This publication contains copyrighted software and circuit design information. The purchaser of this monograph is hereby granted permission to use this material for non-commercial purposes. For information on licensing any part of this design package for resale or other commercial use (easily arranged), please contact the author, Steven K. Roberts, at *wordy@qualcomm.com* or the physical address below.

Nomadic Research Labs
1313 S. Hagen Road,
Camano Island, WA 98292

Entire contents © 1999

To help us track the distribution of this design information, both print and PDF versions of NRL monographs are subject to a simple "honor-system" registration procedure. Please send us a brief email message with your name and address -- along with any notes about your intended use of the Sexbar design and comments about the publication. This will enable us to send appnotes and updates reflecting changes or errors. (NRL never sells or abuses its database, so you won't receive spam or end up on anybody else's mailing list!) If you would also like to receive occasional news of the Microship adventure about once a month, please mention that in your note.

••• If you build a Sexbar, please let us hear about it! •••

INTRODUCTION

It became clear in the early days of Microship system development that some deeply-integrated communication tools would be necessary for what we have come to call "low-level networking" — the roughly 2,000 potential one-to-one interconnections among dozens of audio, video, and serial data streams. Among the Macintoshes and wireless manpacks, Ethernet handles high-speed networking; among the FORTH controllers, linux boxes, PCs, communication happens over a 9600-baud multidrop line. But what about all the little guys? How do we allow software to establish any random link (including countless possibilities not anticipated at design time) between systems that lack TCP/IP protocol stacks and networking hardware?

The problem is serious, and one obvious (and very wrong) solution is to add switches, patch panels, and relays as needed. You see this sort of thing all the time in complex environments like ham radio stations and large yachts... a multiplicity of speakers, microphones, patch cables, mode switches, and kluge interface boxes. These are almost impossible to "edit" as needs change... and change they will!

Multimedia networking tools are becoming commonplace in the marketplace, allowing streams of digitized video and audio to be handled as data. It would be nice if this could work universally, but until every radio, cellphone, speech synthesizer, and cheap video camera adopts a universal digital interface standard (don't hold your breath), something else is necessary.

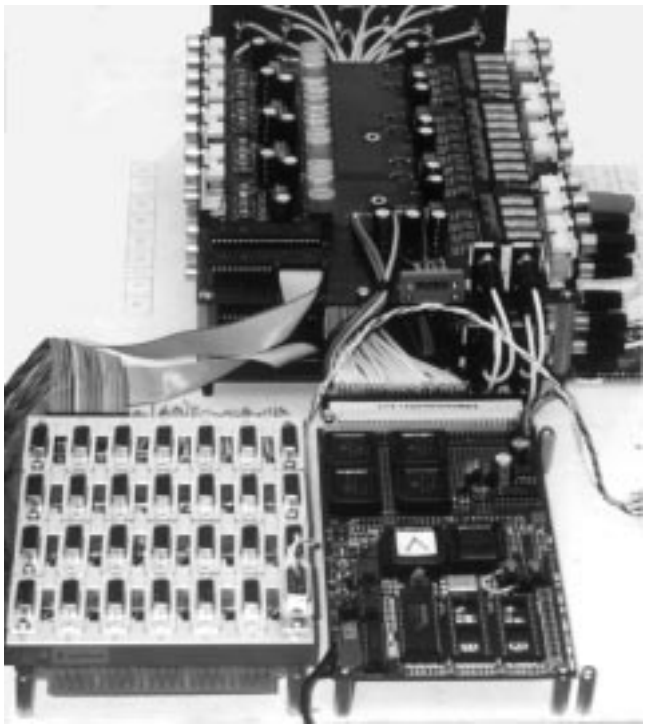
Enter the crosspoint switch. Typified by the Mitel 88xx series, these devices are conceptually simple and very elegant — an X-Y array of addressable analog FET switches with a corresponding RAM (in most cases, write-only) to hold the current pattern. Each switch corresponds to a bit in this memory, and by writing an address along with a 1 or 0 to that location, you can turn the switch on or off. The 8816, for example, is an 8x16 array of switches in a 40-pin DIP or 44-pin PLCC...the URL for a downloadable datasheet is in the References section (page 39). There are a lot of ways to build these devices into a system, and there are three different implementations in the Microship...

Serial Crossbar (Sexbar — Monograph NRL-601): The subject of this publication, the Sexbar has become an extremely useful tool on the Microship, providing up to four simultaneous bidirectional connections among any of 32 devices. This 8816-based design includes an interesting feature... when a connection request is made, the controller software successively tests all the involved pins for transmit or receive behavior and then creates a "straight cable" or a "null modem cable" as needed. The physical interface consists of a panel of 28 female DB-9 connectors, with three of the other four channels used for internal networking in the control system and the final one used for the *polarity testing* function.

Audio Crossbar (Auxbar — Monograph NRL-602): This one was actually the first to come online, and was developed during the BEHEMOTH bicycle project back in 1991. This expandable system is based on a printed circuit board with two 8816s, one providing 16 input channels through its "X" lines, and the other 16 output channels. The 8 "Y" lines are used to bus the two chips together, and thus impose an upper limit of 8 simultaneous connections among any of the 32 I/O channels. The trick, however, is that those 8 lines are also brought out to connectors, allowing an arbitrary number of boards to be stacked (in our case, two). The design allows rudimentary mixing, though without any level-control features; it's also possible to drive multiple outputs, but the noise goes up if you're not careful because op-amp inputs see each other. The physical interface uses 64 RCA jacks, and each channel has an amplifier to standardize impedance and line-level outputs.

Video Crossbar (Vixbar — Monograph NRL-603): Using four Mitel 88V32 chips, which are optimized for video in various ways (200 MHz bandwidth, -80dB crosstalk at 5MHz, grounding of open connections, separate analog and digital power supplies, etc.), this subsystem gives us a non-expandable array of 16 video inputs and 8 outputs, with up to 8 simultaneous unrelated connections. One of the features of the 88V32 is a second stage of latching to allow clean switching of multiple channels at the same instant (which can even be synchronized with the retrace interval if you're serious about minimizing video glitches). Like the Auxbar, the physical interface is via RCA jacks, and the overall package is *very* small... it all fits in the kluge area of a New Micros 68HC11 board.

All three of these systems happen to be mounted on a single substrate in the *Microship*, but if you only need one of them, you can simplify things considerably (they are treated separately in this publication series). Though the detailed design data will focus on our working implementation for maximum accuracy, we'll give you a number of suggestions for factoring out components or scaling to a different number of channels. This is much easier than it sounds... the three crossbars were developed individually and then merged, so don't be overwhelmed by this picture of *Grand Central Station*.



Grand Central Station

I can't stress enough the utility and convenience of this system. Low-level serial interfacing has become so simple that sometimes I make RS-232 connections between devices just for the fun of it... a statement I've certainly never been able to make in 2.5 decades of doing battle with pins 2, 3, and 7. One evening, just for kicks, we made a quick setup macro: With one click, the Icom 725 ham transceiver was commanded via the Sexbar to the NAVTEX frequency, its audio output routed via Auxbar to the KAM+ terminal node controller, the resulting serial data piped via Sexbar to the Audapter speech synthesizer, and *its* audio linked to a little Ramsey FM transmitter. We walked around the lab listening to live Coast Guard weather beacons on a Walkman. The whole process only required 5 lines of code, and it worked on the first try.

As a bonus for marine users out there, we include the design of a little circuit that provides NMEA-0183 <—> RS-232 conversion. This allows off-the-shelf GPS receivers, compass sensors, autopilots, chart plotters, and so on to share the convenience of the Sexbar. One of the *last* things you want to mess with at sea is connector-swapping, and it would be nice if the NMEA hardware standard had been written with a few more network-engineering principles in mind — if so, it would handle multidrop and offer better noise immunity. But you know standards: they're a good thing... that's why there are so many of them!

I hope you'll find some or all of this of value in your projects, and that you will contact us with reports on your work. We'll include user commentary in future versions and email updates. Also, *please* be sure to register and give us your email address (if we don't have this info already) and keep us posted on your contact info. If I find a bug in the code, I want to be sure the update makes it to all users, and we routinely send appnotes and comments to all registered Design Package owners via the Net.

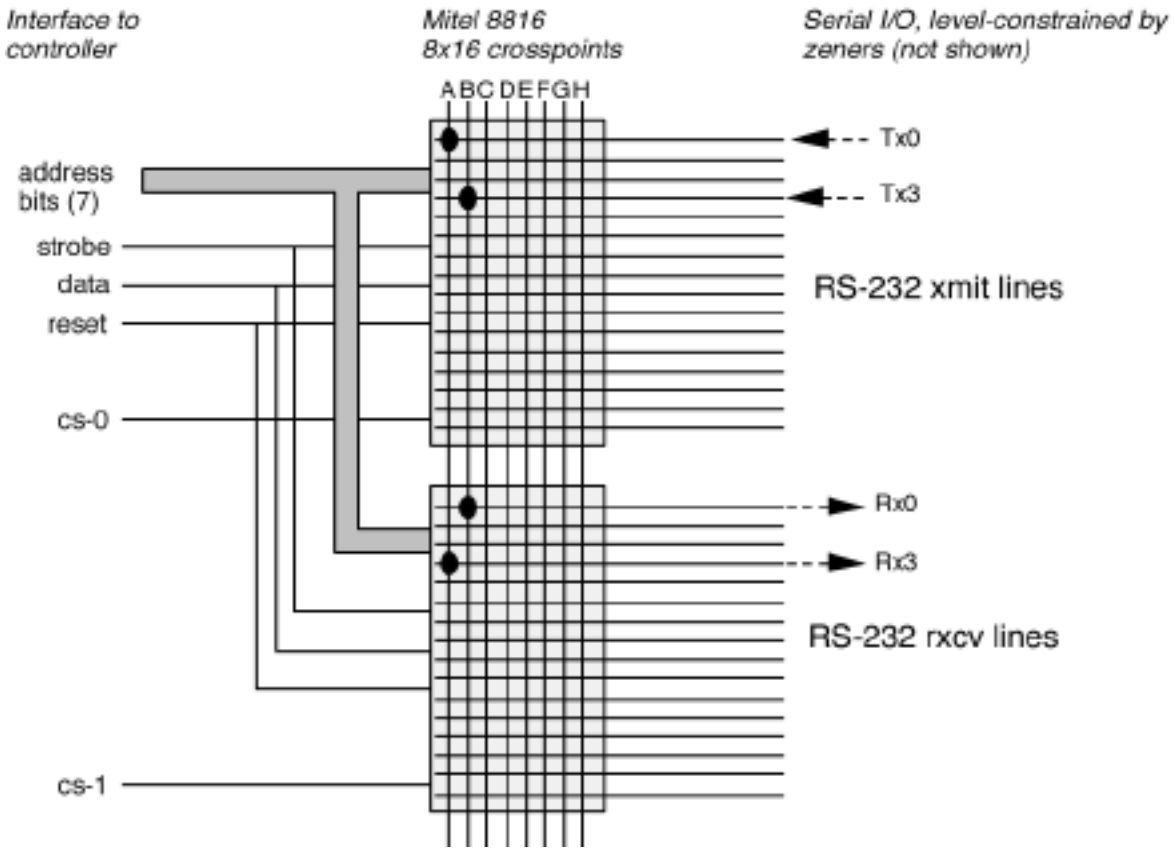
Let's do it!

1.0 A CROSSBAR BACKGROUNDER

The key components in all three of our crossbar subsystems are Mite crosspoint analog switch arrays (we use the terms crossbar and crosspoint more or less interchangeably). A full data sheet for the 8816 may be found at Mitel's website: <http://www.mitelsemi.com/products/pdf/dataserv.cgi?mt8816> -- smaller devices in the product family include 8x4 arrays (8804 and 8806), 8x8 arrays (8808 and 8809) and 8x12 arrays (8812, 8814, 8815, and MT093).

These devices are conceptually straightforward, and have a wide variety of interesting uses. Basically, you can think of them as a matrix of FET switches in which any combination can be turned on or off through a simple RAM-like addressing scheme. This allows any pair of X and Y lines to be connected together, multiple lines on one axis to be bussed to a single line on another, easy synthesis of analog multiplexers, and, of course, the crossbar applications discussed in this book and others of the series (see NRL-602 for the Auxbar, and NRL-603 for the Vixbar).

The conceptual block diagram below should give you a good idea of how the devices work. In this drawing, we have 16 inputs and 16 outputs — requiring only two 8816 chips.



Sexbar: A conceptual block diagram

The vertical Y lines are all tied together, providing a set of 8 buses. The chips can of course be interconnected in various ways, but this gives us the maximum number of inputs and outputs with a reasonable upper limit of 8 simultaneous connections. The horizontal lines, without any external circuitry, are fully symmetrical — although as we will see in a moment there are some interesting "polarity" issues with RS-232 connections... not to mention the need to constrain the network to a +/- 5V signal range with external zeners. For many applications the chips alone are quite sufficient, with each ON switch representing a DC resistance of approximately 65 ohms.

The control lines are all shown at the left. 7 bits of address are internally decoded into 1-of-16 X lines and 1-of-8 Y lines, and this is latched into the 8816's address decoder when Strobe goes high (assuming that the device's *Chip Select* line is also high). In the system described in this book, an external 74HC138 decoder will be used to generate the *CS* lines. The action taken by the addressed switch — on or off — is determined by the *Data* line. And finally, a *Reset* line is available to clear all the switches in the device. Obviously, large crosspoint arrays can be configured by simply busing together the control lines as shown in the conceptual diagram.

To turn all this into a serial crossbar, all we have to do is make two connections through the network as shown in the conceptual diagram. As you probably know, all RS-232 connectors (DB-9 and DB-25) use pins 2 and 3 for transmit and receive; ignoring, for the moment, the fact that many cables swap these and render the standard endlessly confusing, let's go ahead and connect all pin 2's to the top chip and all pin 3's to the bottom one.

Now we can do some networking. Our sample connection is between the devices plugged into channels 0 and 3. The black dots represent "on" switches in the crossbar chips... if you follow the lines, you can see that the transmit line of channel 0 (Tx0) is connected via two switches to the receive line of channel 3 (Rx3); likewise Tx3 is connected to Rx0. This was accomplished by four separate write commands to the pair of 8816 chips, in each case setting up the switch address, setting the data line high to set a bit, and doing the appropriate dance with chip select and strobe.

But what if one of the cables had been a null-modem type, swapping pins 2 and 3? Because this is so likely in any environment full of RS-232 devices, it would be nice to design the whole Sexbar system so we simply don't have to care about signal polarity. After all, the network supports it — if we move a couple of those black dots in the drawing, we can rearrange the connections at will. How can we do this automatically?

As you will see later in this book, the magic trick here is a hardware *window comparator* that is used by the controlling program to successively test all four pins of the channels involved in the requested connection. In each case, a line is connected through the network to a special reserved channel, and the comparator determines whether it is a receiver or a transmitter. Knowing this about all four lines allows the software to set up a "virtual straight cable" or a "virtual null-modem cable" as needed, transparently making the connection without the user having to care how it happened... as long as the connectors fit and the baud rates are the same, it will work.

In the Microship application, control information comes directly from output ports of a 68HC11 microprocessor running FORTH. These are low-cost (under $100) controllers from New Micros of Dallas, TX, and offer a very easy development environment for small embedded systems (call 214-339-2204 or see http://www.newmicros.com). All the code in the listing later in this book assumes that this is what is running the show, but there's nothing to stop you from reverse-engineering it for a different kind of processor (or language) if you wish.

With that, let's get to the implementation details. Please note that the design presented here was created for the Microship and thus makes reference to certain specific devices. Fortunately, however, this crossbar system is so general by its very nature that it can drop with virtually no changes into any other application. Feel free to change the matrix size or integrate the Sexbar with one or both of the others by hanging them all on a single controller (at the moment, Auxbar and Sexbar share similar software and the same interface circuitry on the *Microship* Hub processor; Vixbar was designed separately and has its own underworked processor... certainly not necessary!)

1.1. IMPLEMENTATION NOTES

CHANNEL UTILIZATION

The serial crossbar allows four simultaneous bidirectional connections (8 lines) to exist among 32 possible channels.

The idea here is that "dumb" RS-232 devices such as GPS, compass, packet, speech synthesizer, serial LCD, and the like should not have to be TCP/IP or multidrop network savvy. They are all designed to plug onto serial ports, and this circuit takes care of that under control of the FORTH Hub. Four of the 32 channels are associated with the Hub itself; the other 28 are presented as a 4x7 matrix of female DB-9 connectors piggybacked on the Sexbar board.

NOTE: In Sexbar applications that are not based on the Microship's *multidrop network of controller boards, think of the Hub as any standalone controller. 31 of the 32 channels are available in general applications, but as noted below, the ship system uses the Sexbar to link the Hub with the multidrop network of nodes (as well as other devices), and it's also capable of redirecting its own console using a latching relay to allow alternative hosts. Feel free to make channels 29, 30, and 31 general purpose. It was also convenient to bring out 28 channels to the connector array, since it was easy to package as a 4x7 matrix.*

In the *Microship*, the four Hub-specific channels are:

00 — the transmit-receive sensor and steady mark, about which more later
29 — the ACIA through which the Hub talks to nodes and all serial devices
30 — the Beeline multidrop network to all FORTH nodes
31 — The Hub's console if redirected via latching relay

Channel 31 is one level removed from the connected device... the Hub console port is defaulted on power-up or reset to connect to the server that sits on top of it. Only by switching the latching relay does it appear instead on crossbar channel 31, whereupon any other device can have direct control over the Hub. This is to prevent deadly embrace conditions in which software crash prevents regaining control of the system. You can safely ignore all this and use it just like any other general purpose channel — I include this implementation-specific commentary here just to give you an idea of some of the tricks that are possible once anything can connect to anything.

Also note that the normal connection between the Hub and the multidrop network below it ties up one of the four available bidirectional comm channels, with the 29-30 link active. This originally was handled by another latching relay, but the added complexity of switching between the network and random devices led to simplification at the cost of slightly increased volatility. Again, other applications are probably not affected by this.

The bottom line is that the only sacred channel that you absolutely may *not* use for connected devices is 00 — it's hardwired to the window comparator! The other 31 are all yours, though the schematic here will only show 28 of them going to DB-9 connectors since that's how I happened to do it.

USER INTERFACE COMMENTS

As will be clear from the software discussion in Section 1.4, use of the Sexbar reduces (in most cases) to the commands DSCONNECT, DSREMOVE, and SRESET. Connecting channels 14 and 20, for example, is just a matter of issuing the command:

```
14 20 DSCONNECT
```

to the controller, whereupon it makes the connection if it successfully finds two healthy channels, then returns a status byte (see Section 1.8). Obviously, this is at the low end of the user-friendliness spectrum, and it is assumed that most people will immediately add an interface layer.

The simplest way to do this, of course, is to simply assign channel names to the numbers using constants in FORTH:

```
COMPASS NAVPC DSCONNECT
```

The downside of this, if you bake the code in ROM, is that the resulting inflexibility might be at cross purposes to the delightful generality of the Sexbar itself... today's assignment of COMPASS to channel 14 might later become WINDSENSOR, whereupon you have to live with a confusing name or reprogram the EPROM.

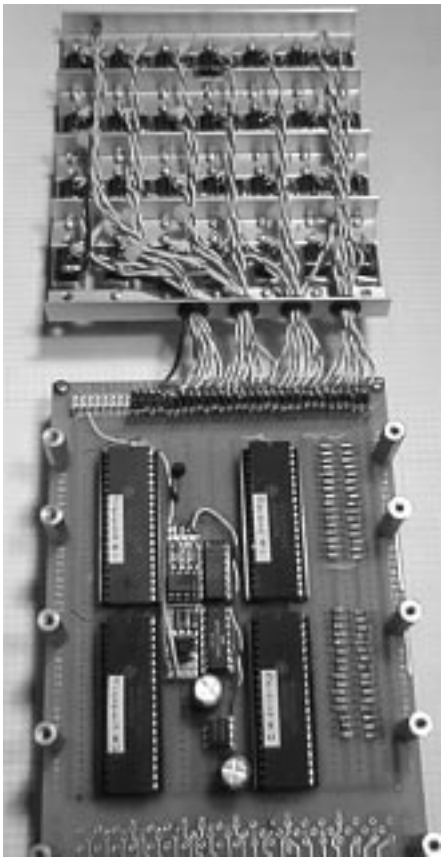
Moving up a level, you can embed the naming layer in a host system, as we do in the Microship. That way, the Sexbar remains a general-purpose switchbox, and your front-end machine takes care of making the experience more pleasant. We have used HyperCard, NewtonScript, and most recently, CGI Scripts to issue one-line crossbar commands in response to requested events. Having sampled all three and become a fan of platform independence, I recommend using an embedded PC as a general purpose server (we're using linux with Apache), and building HTML and Perl tools to suit the application. In most situations that I can imagine, the Sexbar will not be used as a raw connection device, but as a component in a larger suite of activities... although you can certainly build a raw matrix tool that controls the crossbar system via clicking on device images followed by a CONNECT button, with either a textual list or graphic representation of active connections.

As an example of integration into other applications, in the Microship there is a page (viewed with Netscape) that sets up packet radio operation. A single button powers up the hardware, links serial devices through the Sexbar, makes audio connections through the Auxbar, and opens a terminal window so I can type messages and view incoming text. I don't go to a power control view to turn things on, then a Sexbar control view to make the serial connection, then an Auxbar control view to link the radio to the TNC, and then finally to a terminal so I can use the damn thing. The simple one-liner interface of the Sexbar code means that all serial interconnect commands can be scripted easily and dropped into other applications, reducing this to the level of an invisible background tool (which is, after all, where cabling belongs!)

Every installation is different, of course, and I'd love to hear how you build your front end. The next version of this book will hopefully include a few other application examples besides my own...

1.2. CIRCUIT DESCRIPTION

The main body of the Sexbar is on page 1 of the schematic (Section 3). Four Mitel 8816 chips (8 x 16 crosspoint switches) are wired into an array in which the 8 **Y** lines are all connected together. The 32 **X** lines for receive into the matrix (right side of the page) and transmit from the matrix (left side) are brought out to the outside world. (See the following section for a discussion of RS-232 polarity issues).



Sexbar unfolded

A Maxim DC-DC converter (MAX660) is on board to provide the +/-5V required for bipolar operation, and this supply voltage created one of the more annoying design constraints — the need to restrict the RS-232 operating range from the common +/-12 V or so to +/-5V in order to avoid destroying the 8816 chips. This explains the profusion of zener diodes on the board (*receive* side) and the connector array (*transmit* side). Each line carries a series resistor and a pair of back-to-back 4.3V zeners (1N5229) to ground. The sum of 4.3V and the nominal .7V forward drop prevents voltage excursions outside the +/-5V range, and the resistor keeps this from loading the transmitter more than about 1 mA.

(Originally, it was assumed that careful wiring would guarantee that all transmit lines would come into the receive side of the system, and our student team at UCSD (Jeff Simon and Dan Sebald) assembled the board with only 32 protection networks. Upon later reflection, it became clear that the inevitable RS-232 connection errors would cause problems, and in final packaging and integration I added another set of networks on the DB-9 connectors, the only convenient spot. The system is electrically symmetrical, however strange it may look... if you design a Sexbar board, I strongly encourage you to put all the protection networks on it instead of going through the tedious point-to-point wiring required for the other method!)

The network is controlled via a ribbon-cable connection from the Hub interface (shown as "Sexbar cable" on the last page of the schematics in Section 3) which arrives at the unit via the 26-pin header (1-B6 on the Sexbar schematic). Note that pin 1 (or 28a, to match the physical notation on the board) has been removed since the connector on the other end of the ribbon is a DB-25; note also that pin 28b is deliberately vacant so that the next three hub serial data pairs will occupy twisted pairs in the ribbon cable for noise reduction. The pins from 33a to 40b (or 10-25 at the DB-25 end) are for control, and carry the address lines to the 8816s, reset, the data bit used to set/reset any addressed location, and strobe. Chip selects are generated by U5, the 74HC138 at 1-D7, and inverted by four sections of an HC04.

NOTE: Again, what you see in the schematics reflects the Microship implementation. If you are building a standalone Sexbar, you should ignore the hardwired channels 29-31 and drive the cable at the lower left corner of the schematic directly from your processor's output ports. The interface circuitry supports Sexbar and Auxbar on our Hub nexus board, but this can be done any way you like as long as the address and control bits get there somehow! See Section 1.7 for more on this.

Other than the circuit discussed in the next section, the only other item of note is the Count LED mounted on the connector frame, driven by U6e from pin 20 of the Hub-Sexbar cable. This can be driven by a task in the Hub that visually indicates the number of active connections by blinking in short bursts of up to four flashes.

NOTE: We left this in the schematic because it may be of value to other users, but in the Microship the LED task has been eliminated in favor of a status display on a small LCD. This involves less software overhead, and is a more useful display. The LED was too tempting to ignore, however, so the current code just turns it on whenever any new connection is in the process of being set up. If you want to use the status task, it is presented in Section 1.5.

1.3. RS-232 POLARITY SENSOR

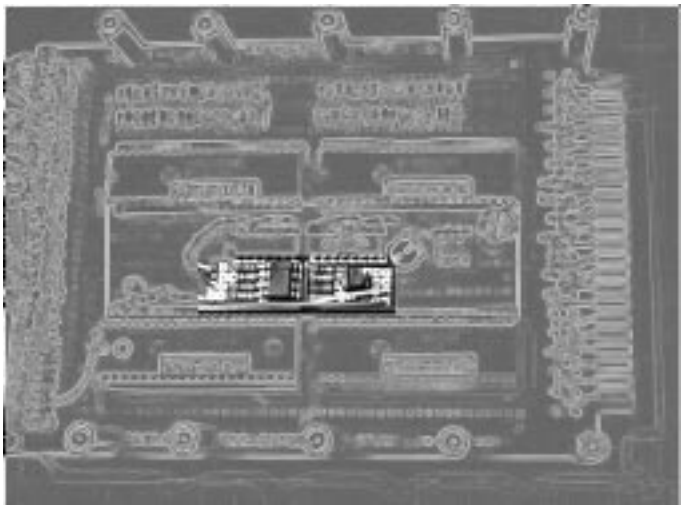
It is impossible to standardize a DCE-DTE protocol in this system, since random external devices of both flavors can be connected. Instead of heavily using "null modem" connectors or splitting the DB-9 matrix into two polarities and requiring the user to pay attention (always dangerous), we decided to fix it in software.

Specifically, the physical wiring is such that all DB-9 pin 2 lines are into the "receive" side of the crossbar matrix, but the system is completely symmetrical. The challenge is to figure out which pin (2 or 3) is transmit from each of the connected devices, and then swap or not as needed. For example, if channels SX14 and SX20 are being connected, and each contains a Tx and an Rx line, we would have the following two choices:

```
DIRECT:     Tx14 — Rx20  and  Rx14 — Tx20
 — OR —
SWAP:       Tx14 — Tx20  and  Rx14 — Rx20
```

The latter seems counterintuitive until you realize that there's nothing sacred about our internal Tx and Rx notations once we step outside the boundaries of this system, no matter how well we adhere to convention internally.

Our first thought was to burden the human with the setup, providing an LED test fixture which would be used prior to plugging a device into the Sexbar. This is, of course, silly. The real need was for the Hub software to easily detect the difference between transmit and receive pins of a connected channel, and this was accomplished with the circuit on page 2 of the schematic.



Daughterboard highlighted

This *window comparator*, built as a small daughter board, uses a voltage divider (the two 56K and the 100K resistors) to establish setpoints at approximately approximately +/-2.5V" and via the commoned open-collector outputs yields a high if the input line is between these values and a low

otherwise. This low, alas, is about -3V, so the transistor is there as a kluge to both logically invert the signal and constrain it to 0-5V operating range (yes, a FET would have been better). Since the input to the circuit is Tx0 and the output goes to hub input bit 0 of port C031 (known locally as XIB-0) we how have an easy way to switch either side of any channel to this line under software control, see if it's asserting either mark or space (as opposed to receiving, in which case the voltage is roughly zero), and determine accordingly whether the device is wired as DCE or DTE, to use that arcane RS-232 terminology. Connection can then proceed either direct or swapped, as needed.

The dedication of half of the SX0 channel leaves the other half free, so we had an opportunity to solve another potential problem — switching glitches. I've often noted that plugging and unplugging RS232 cables throws a garbage character or two at the devices, and if that proves to be a problem here we can just connect sensitive channels to Rx0... which is tied to -5V to provide a steady MARK condition. This is done at 1-H6. So far in actual use, this has not been necessary.

1.4. SEXBAR SOFTWARE DISCUSSION

The first pass at serial crossbar software was little more than a clone of the audio crossbar code by Jason Corley, based on earlier work for *BEHEMOTH* by Mike Perry. This did not accommodate paired channels, autosense, or the special requirements of serial data, and has been substantially enhanced for use in the Microship environment.

The FORTH code in Section 2 should be consulted for the following discussion.

Note that two forms of output port handling exist. Internal 68HC11 memory-mapped port bits have the advantage of being readable, making it trivial to modify a bit without affecting its neighbors (such as PORTA C@ 04 OR PORTA C! to set only b3). Unfortunately, the 64 output bits provided by the NMIS-3004 board addressed at $C020 are NOT readable, requiring that they be accessed through array XOUTS[] so a task can change something within a port without trashing the state set up by another piece of code. The low-level port fetch and store utilities, XC@ and XC!, are located in the hub tools file.

```
C020 CONSTANT XOBASE
CREATE XOUTS[] 8 ALLOT

: XC@ ( port — 8b )
      XOUTS[] + C@ ;

: XC! ( 8b port — )
      2DUP XOUTS[] + C! XOBASE + C! ;

XOUTS[] 8 0 FILL
XOBASE 8 0 FILL
```

NOTE: Most users of this would have no need for the extra output board and the associated XC@ and XC! code, since the HC11 plus the HC24 PRU generates enough output bits to do the job (just define the internal ports at the beginning of the listing.) But just in case you DO need to use the same method with an external output port board, here's the support code referenced in the listing. Specify the desired port as 0-7; the "8b" stack notation refers to any 8-bit value.

Operation of the Sexbar software can be viewed at two levels. Individual channels can be interconnected or dropped via the `SCONNECT`, `SADD`, and `SREMOVE` commands... but this is not how we usually think of RS-232 connections. The higher level (`NULLMODEM`, `STRAIGHT`, and most notably, `DSCONNECT` and `DSREMOVE`) deals with channel pairs.

The only words you are likely to use as commands affecting the Sexbar matrix are DSCONNECT, DSREMOVE, and SRESET, underlined below.

• SCONNECT (channelA channelB —)

This accepts two numeric raw channel addresses, in the range of 00-3F, and connects them via the first available free bus (there are 8). Note that this does NOT perform the T-R and R-T connection required for a pair of serial data lines — this is the low-level, single channel connection and would not normally be used as a user commend. Channels $0-1F are considered Transmit and $30-4F considered Receive, though the array is symmetrical and RS-232 cables rarely follow predictable rules. Each physical DB-9 connector has pin 2 connected to one of the Transmit lines and pin 3 connected to that address plus $20. SCONNECT confirms that the channels are not in use, finds a free bus, computes the 8816 addresses, and makes the link — returning the text error message "All buses busy!" if appropriate. It uses SFINDBUS, BUSSET, JACKSET, and SETSWITCH.

• SADD (channelA channelB —)

Connect channel A to the bus currently occupied by channel B. Almost never relevant in the serial crossbar environment, this is provided only to allow serial receivers to *subscribe* to active transmitters, as in the case of multiple systems needing the GPS or compass data stream — or a diagnostic terminal needing to monitor all network traffic. SADD only works if channel A is unconnected and B is already on a bus.

• SREMOVE (channel —)

This tool allows undoing the operation above, as well as breaking any connection established by SCONNECT without clearing the whole network.

• SRESET

Clears the Sexbar's 8816 internal arrays via the RESET line, as well as doing a zero-fill to clear the SWITCHES array in FORTH and a CLEANUP of the controlling ports.

```
SSHOW
 0 --------
 1 -------*
 2 --------
 3 --------
 4 --------
 5 --------
 6 --------
 7 --------
 8 --------
 9 --------
 A --------
 B --------
 C --------
 D --------
 E --------
 F --------
10 --------
11 --------
12 --------
13 --------
14 ------*-
15 --------
16 --------
```

• SSHOW

Displays an ASCII map of the 32 channels and 8 buses, used primarily for diagnostics but potentially valuable for resynchronizing the console system with the state of the network without having to rebuild the connections.The SSHOW command yields a raw display of the matrix, as in the example left. This is the result of a connection request between connectors 1 and 14, and illustrates how the vertical buses are assigned from the lowest to highest. Physical channel 1 has been connected to 34 on bus 0, and channel 14 has been connected to 21 on bus 1. Successive connections will consume free pairs from right to left until none remain.

• NULLMODEM (SXA SXB —)

Without questioning the wisdom of the command, this word makes a pair of connections between any of the 32 serial channels SXA and SXB (which are numbered from $0-1F and each consist of a PAIR of physical channels as noted above in the SCONNECT discussion). The connections performed by NULLMODEM amount to a swap of pins 2 and 3 just like the cable by the same name; thus the command 4 11 NULLMODEM would connect physical channels 4 to 31 and 24 to 11.

• STRAIGHT (SXA SXB —)

Similar to the foregoing, STRAIGHT connects two pairs of physical channels. The example above, were it given as 4 11 STRAIGHT, would connect 4 to 11 and 24 to 31.

```
17 --------
18 --------
19 --------
1A --------
1B --------
1C --------
1D --------
1E --------
1F --------
20 --------
21 ------*-
22 --------
23 --------
24 --------
25 --------
26 --------
27 --------
28 --------
29 --------
2A --------
2B --------
2C --------
2D --------
2E --------
2F --------
30 --------
31 --------
32 --------
33 --------
34 -------*
35 --------
36 --------
37 --------
38 --------
39 --------
3A --------
3B --------
3C --------
3D --------
3E --------
3F --------
OK
```

• DSCONNECT (SXA SXB — flag)

is the primary high-level word (Double SCONNECT) for making a connection via the Sexbar. This invokes either NULLMODEM or STRAIGHT by examining all four relevant pins of the two named serial channels, configuring the connection to connect transmit to receive and receive to transmit. This is performed with four calls to POLTEST, the polarity tester that uses the window comparator on Tx0 described earlier. The four returned bytes (each 1 or 0) are combined into a single byte between 0 and $F, and then tested for the four distinct conditions that represent valid pairs of serial cables. Any cable with both pins inactive (or active!) is an error condition, the connection is not attempted, and the condition byte is returned for analysis. The four VALID conditions and their results are:

```
3    TxB & TxA active NULLMODEM
C    RxB & RxA active NULLMODEM
6    RxA & TxB active STRAIGHT
9    RxB & TxA active STRAIGHT
```

(For the other 12 possible returned values, see the table in section 1.8.)

• DSREMOVE (SXA SXB —)

is the converse of DSCONNECT, and removes all four physical connections associated with a link between two serial channels. It calls DREMOVE twice, which in turn calls SREMOVE twice.

For debugging and hacking purposes, a short description of the FORTH words NOT intended for external use follows...

• ENCODE (n1 — n2)

This uses the TWISTY array to conveniently match the named 8816 pins to binary numbering.

• BIN. (n —)

Used by SSHOW to pretty-print the SSWITCHES[] array to the console. This word outputs the bit-by-bit contents of the byte passed on the stack, using the values ASCII0 and ASCII1 (originally 0 and 1, but now the more visible - and *).

• S8816_RESET ( )

Toggles the 8816 reset line, clearing the contents of all four chips.

• DROPMASK (mask — n)

Converts a byte-long single-bit mask to a number between 0 and 7

• STROBE ( )

Toggles the strobe line to the 8816s, latching the address, data, and chip select values present on the ports.

• DATA1 ( ) and DATA0 ( )

Set or clear the data line to make or break a connection.

• SCS (chip# —)

Pass the appropriate Sexbar 8816 chip select to port XOB.

• BUSSET (mask —)

Selects a bus on port XOA via a bit mask, using DROPMASK to convert to an 8816 Y address.

- `JACKSET ( jack# — )`

Sets the 8816 X address in port XOA. Jack# must be in the range 0-15, and should be the table position of the connection, and ENCODE is called to map this to the 8816 array addressing.

- `SETSWITCH and CLRSWITCH`

These are used to actually set or reset a selected switch in the 8816 array.

- `CLEANUP`

Clears the control and address bits to prevent stray nonsense. Added during debugging, and probably not truly necessary!

- `POLTEST ( ch# — flag )`

This is used by DSCONNECT, and connects the specified physical channel to Tx0, which is the window comparator. A value of 1 on bit 0 of the port at $C031 means that the connected device is currently asserting either MARK or SPACE, and is not a receive line. This flag is returned.

- `DREMOVE ( SXn — )`

Remove both connections associated with a serial channel, the base number (transmit) SXn and the receive side SXn+$20.

1.5. SPECIAL NOTE ON LED BLINK TASK

As mentioned earlier, the original status display on the serial crossbar consisted of a single LED driven by a 68HC11 port bit. This is simple and effective, but in the Microship environment it made more sense to collect data from variables with the Hub and display the connection counts on a small LCD — the approach reflected by the software listing (in this case, reading the SEXCOUNT variable). For your convenience in applying a less hardware-intensive method to crossbar status indication, the following code segment along with Bill Muench's multitasker (included in Listing 2.2, but otherwise not necessary for Sexbar operation) will do the trick, yielding a blink pattern similar to common answering machines: two active connections, for example, will cause the LED to blink twice quickly, then pause for 5 seconds or so. The timing is trivially tweakable.

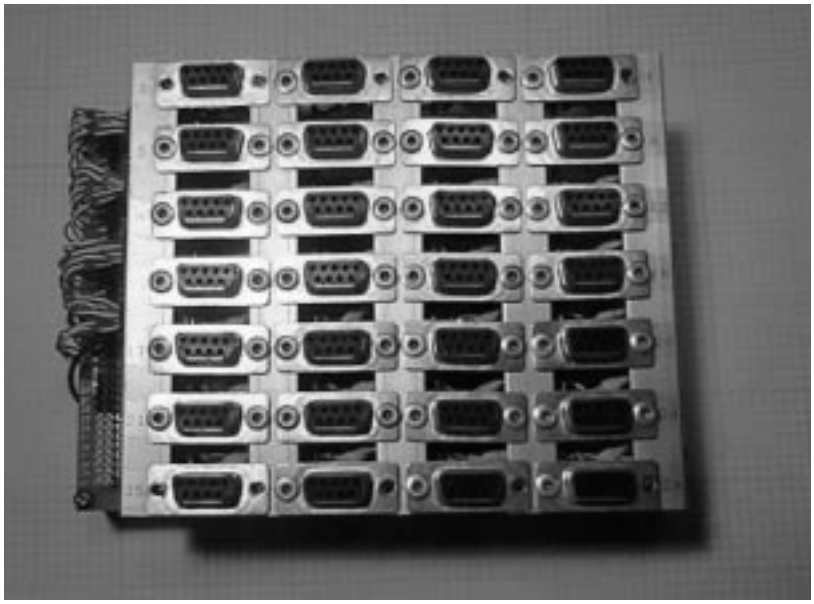
```
CREATE B.LEDTASK ( marker )

0 0 0 HAT LED ( allocate task table )

HEX

: REDON B000 C@ 20 OR B000 C! ;
: REDOFF B000 C@ DF AND B000 C! ;
: LAG 300 0 DO PAUSE LOOP ;
: FLASHER REDON LAG REDOFF LAG ;
: GAPOSIS 30 0 DO LAG LOOP ;

\ OR the whole array together to find busy buses...
: BIGMASK ( — mask )
     0 ARRLEN 0 DO SSWITCHES[] I + C@ OR LOOP ;

\ Blink red LED once for each busy bus
: BLINKY ( — )
     BIGMASK
     8 0 DO
          DUP 1 AND IF FLASHER THEN 2/
     LOOP
    DROP ;

: LEDGO ( — ) ( define new task )
  LED ACTIVATE ( init task, it will run on next PAUSE )
  BEGIN
    BLINKY GAPOSIS
  AGAIN ;

: COLDTASK ( — ) ( init task on startup )
  6 4 ! ( default user base address )
  STATUS FOLLOWER ! ( init FOLLOWER )
  MAIN AWAKE ( should always be awake )
  [ ‘ PAUSE CFA ] LITERAL  ‘PAUSE ! ( set PAUSE vector )
  LED BUILD ( link this task )
;

COLDTASK
LEDGO
```

1.6. SEXBAR PACKAGING

The physical layout of this system is not at all critical, but I should make a few comments about the choices that were made for the Microship.

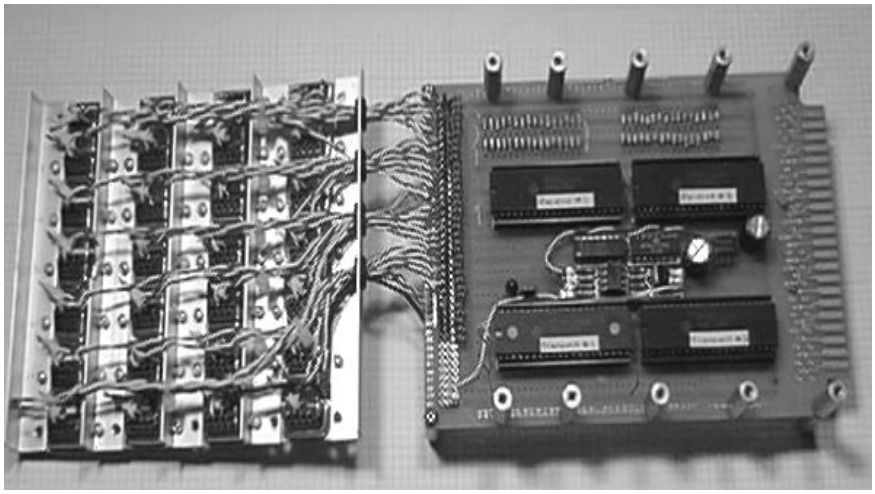
One of the first issues was the choice of connector standards, with everything from screw terminals to raw headers to Mini-DIN considered. Since this implementation is essentially inside an enclosure, almost all interconnects are custom wire runs... so adhering to an industry connector standard was certainly not necessary. The DB-9 is so ubiquitous and cheap, however, that I decided to use them anyway — not only did it simplify the fabrication of the connector array, but it also made the system easy to use during development and hacking without constantly having to fiddle with individual wires. The cost is overhead — 6 out of every 9 connector pins are entirely unused, amounting to 168 pins of dead weight on the Sexbar itself plus whatever is plugged into it. Oh well...

As you can see from the photos, the connector array is a simple garage-shop packaging job with no fancy machining. Five 4.7" rails of .5" x .5" aluminum angle (available from any hardware store) serve as a mounting structure for a 4x7 array of female DB-9 aluminum-shell connectors, each pop-riveted in place. This entire affair is mounted to the circuit board with aluminum standoffs and stainless 4-40 screws. The cable harness is routed out one end and into the board, allowing the assembly to unfold easily for service.



Sexbar connector array

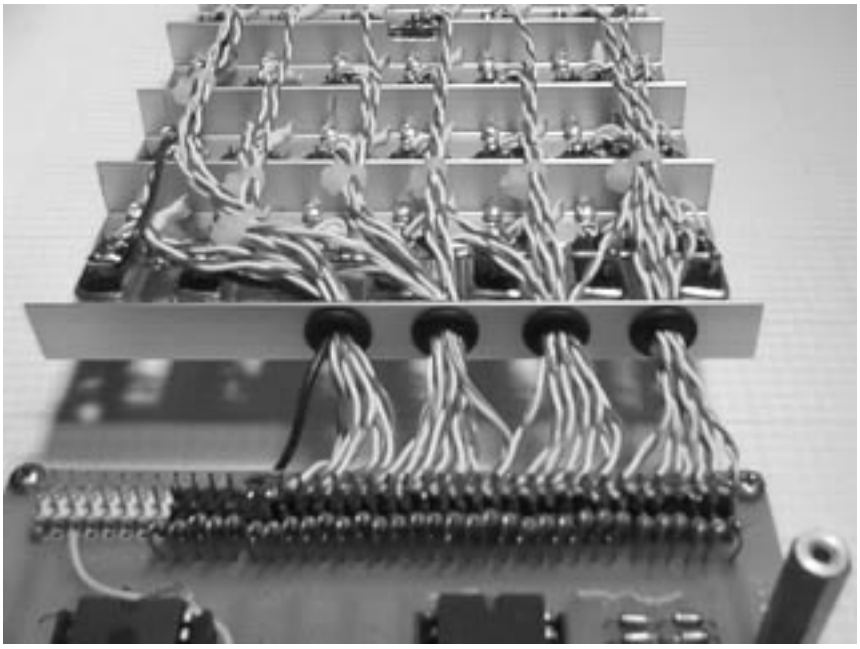
The board itself is a basic point-to-point wiring kluge (with #30 kynar) on pad-per-hole perfboard, in this case a single-sided 4.5" x 6" version. The 8816 chips and other ICs are socketed, and the window comparator (a later addition) is on a daughter board tied down between the 40-pin DIPs. Obviously, if I were making more than one of these, a printed circuit board would be much nicer... but layout is not critical, there are no blazing-fast circuits to consider, and as long as you scatter a few decoupling caps around for good measure any prototyping method should work fine.



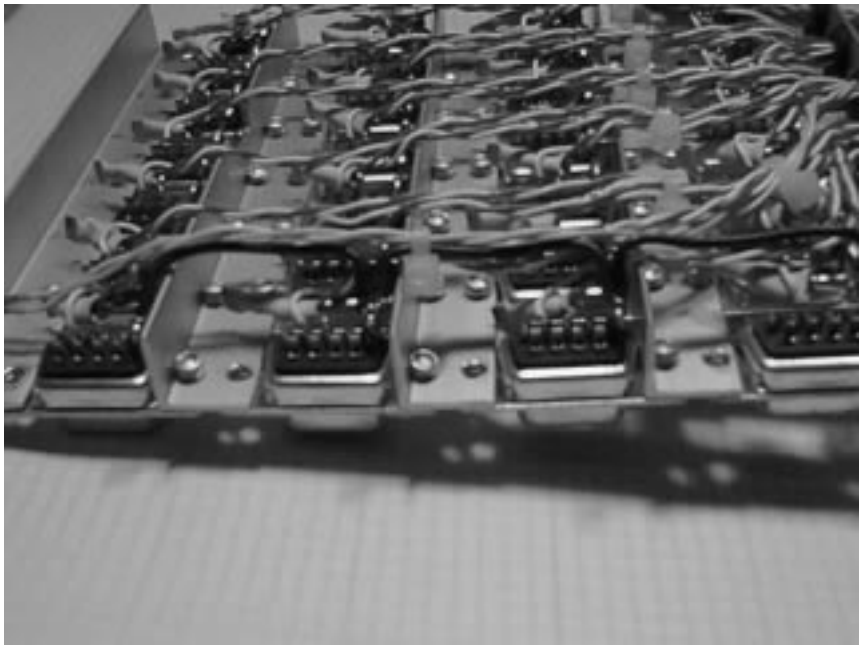
Sexbar unfolded

The board includes a dual inline ribbon cable header on the end, which provides convenient termination for the 28 connectors as well as the interface cable to the control board (the Hub). See the schematic for details.

Sexbar interface header



DB-9 Connector wiring



1.7. SHARED SEXBAR/AUXBAR INTERFACE

We commented back in Section 1.2 about the shared interface circuit shown at the end of the schematic series (5 of 5). There's nothing magic about this, but it does deserve a bit of commentary.

Essentially, this was a way to combine the Sexbar and Auxbar interfaces onto two 8-bit output ports. Doing so was slightly complicated in our case by the fact that the Sexbar board already contains an address decoder (U5) to generate the 8816 chip selects, and the Auxbar does not. The Sexbar/Auxbar interface circuit, therefore, is little more than the addition of a 74HC138 and a couple of cable headers to the Auxbar boards.

The Steering-Nexus connector at the upper left can be ignored — on the Microship, it carries those three dedicated "internal-use" serial ports we mentioned earlier and also delivers a few bits to a completely unrelated subsystem that displays 48 status conditions on an LED matrix. Other users of this design should ignore that connector completely.

The Hub-Sexbar cable is the one of interest here, and the signals have already been discussed: unless you are cloning our full system completely, all you need to do to interface the Sexbar is provide the right control signals to this cable. Nothing else on the interface schematic should matter to you — it is provided here only for a bit of additional context.

If you are building an Audio Crossbar with the aid of NRL-602, then use the right half of this schematic... it takes output port bits from the buses that enter at the bottom of the page, hands most of them unchanged to the two Auxbar interface headers (we use two on the ship, for a total of 32 inputs and 32 outputs), and decodes xob bits 3, 4, and 5 into the required chip selects. Note that three bits are necessary to eliminate ambiguity between this decoder and the one on the Sexbar board (which sees the same low two bits!).

1.8. NOTES, HACKS, AND COMMENTS

End-to-end resistance through the serial crossbar network is observed to be approximately 2-2.5K ohms. This includes about 110 ohms for the 8816s, plus the protection resistors.

In our implementation, receive side connections are from row A (outer row) of the interface header on the Sexbar board.

Chip select decodes:

CS1	CS0	Selected
0	0	Transmit 1
0	1	Transmit 2
1	0	Receive 1
1	1	Receive 2

Stack content after four `POLTEST` executions in `DSCONNECT`: `RxB, RxA, TxB, TxA`. The following table lists all possible one-byte codes that can be derived from this, only four of which represent valid bidirectional serial connections (3, C, 6, and 9). A 0 in the table indicates that the line is either receiving or not connected; a 1 means that mark or space is asserted (transmitting). Channel A is the first one named when `DSCONNECT` is called. A channel called 'crazy' means BOTH sides are transmitting, which should never occur. `STRAIGHT` means a straight-through virtual cable is called for (pin 2-2 and 3-3); `NULLMODEM` invokes the reverse (2-3 and 3-

2).

Code	RxB (pin 2)	RxA (pin 2)	TxB (pin 3)	TxA (pin 3)	Meaning
0	0	0	0	0	All dead
1	0	0	0	1	B dead
2	0	0	1	0	A dead
3	0	0	1	1	NULLMODEM
4	0	1	0	0	B dead
5	0	1	0	1	B dead A crazy
6	0	1	1	0	STRAIGHT
7	0	1	1	1	A crazy
8	1	0	0	0	A dead
9	1	0	0	1	STRAIGHT
A	1	0	1	0	A dead B crazy
B	1	0	1	1	B crazy
C	1	1	0	0	NULLMODEM
D	1	1	0	1	A crazy
E	1	1	1	0	B crazy
F	1	1	1	1	A crazy B crazy

2.1. Sexbar FORTH Listing

The following code is designed to stand alone in a New Micros 68HC11 FORTH board. Note some comments near the beginning about shared Sexbar and Auxbar constants and words — this is the result of some code factoring between the two very similar subsystems. Auxbar-specific code has been removed from this listing, and you should have little trouble modifying this for other applications.

The code is entirely "passive" in that nothing has to run until a command is executed. If you wish to add the LED blink task shown in Section 1.5, then you also need to install a multitasker in the system. It is included as a separate listing in Section 2.2, but that is most emphatically NOT needed for basic Sexbar operation.

As in any well-designed code, all tweakable parameters are defined as constants up front. If you want to build a Serial Crossbar system with more or fewer channels, just adjust the numbers accordingly. Also, as noted earlier, you can reassign the control bits in Ports XOA and XOB (on an external parallel output board) to simple local port bits, then simplify the I/O commands slightly.

```
CR .( Sexbar software )
: B.SEXBAR ;

\ (C) 1996 by Steven K. Roberts, derived in part from Auxbar
\ code by Perry, Corley, and Chu.

\ ================================================================
\ ======= Constants and Words common to Sexbar and Auxbar =======
\ ================================================================
HEX
2D          CONSTANT   ASCII0  ( changed to - )
2A          CONSTANT   ASCII1  ( changed to * )
40          CONSTANT   ARRLEN  ( array length )
8           CONSTANT   BUSWIDTH
80          CONSTANT   HIGHBUS
4           CONSTANT   SRSTLINE
2           CONSTANT   DATALINE
1           CONSTANT   STBLINE
C031        CONSTANT   SENSPORT
1           CONSTANT   TRSENSOR
0           CONSTANT   PORTXOA ( X-Y address bits, LED b7 )
1           CONSTANT   PORTXOB ( 8816 control bits )
C7          CONSTANT   CSMASK  ( non chip select bits of XOB )
0F          CONSTANT   BUSMASK ( non-bus-bits of port XOA )
F0          CONSTANT   JACKMASK ( non-jack-bits of port XOA )

\ Counter for LCD status
VARIABLE SEXCOUNT 0 SEXCOUNT !


\ Make up for the goofy X-numbering on the 8816.
```

```
CREATE TWISTY
    00 C, 01 C, 02 C, 03 C, 04 C, 05 C, 08 C, 09 C,
    0A C, 0B C, 0C C, 0D C, 06 C, 07 C, 0E C, 0F C,
: ENCODE ( n1 - n2 ) TWISTY + C@ ;

\ BIN. output the bit-by-bit contents of the lower byte on the stack.
: BIN. ( n — )
    BUSWIDTH 0 DO 2 /MOD LOOP
    DROP
    BUSWIDTH 0 DO
        IF
            ASCII1 EMIT
        ELSE
            ASCII0 EMIT
        THEN
    LOOP ;

\ ***** General 8816 Output Words *****

\ Convert a number from a byte-long bit mask to a 3-bit number.
\ Feed in a real mask (not 00).
: DROPMASK ( mask - n )
    8 0 DO 2 /MOD SWAP IF
        I SWAP
    THEN LOOP DROP ;

\ Strobe in the input.  The bits in XOB are not affected.
: STROBE ( )
    PORTXOB XC@ STBLINE OR PORTXOB XC! ( stb=high )
    PORTXOB XC@ STBLINE XOR PORTXOB XC! ( stb=low ) ;

\ Set the data bit high.  Other bits not affected.
: DATA1 ( ) PORTXOB XC@ DATALINE OR PORTXOB XC! ;

\ Set the data bit low.  Other bits not affected.
: DATA0 ( ) PORTXOB XC@ DATALINE NOT AND PORTXOB XC! ;

\ Select a bus by bitmask (00-80)
: BUSSET ( mask - ) DROPMASK 10 * ( shift left 4 )
    PORTXOA XC@ BUSMASK AND OR PORTXOA XC! ;

\ Select which jack.  Inputs >15 cause problems.
: JACKSET ( jack# - ) ENCODE PORTXOA XC@ JACKMASK AND OR PORTXOA XC! ;

\ Send in data 1 and strobe
: SETSWITCH DATA1 STROBE ;

\ Send in data 0 and strobe
: CLRSWITCH DATA0 STROBE ;

\ Clear control and address bits
: CLEANUP
0 PORTXOB XC!
```

```
      0 PORTXOA XC! ;


\ ================================================================
\ ======================== Sexbar Code ===========================
\ ================================================================

: SREDON PORTXOA XC@ 80 OR PORTXOA XC! ;
: SREDOFF PORTXOA XC@ 7F AND PORTXOA XC! ;

\ ***** Sexbar-specific 8816 OUTPUT WORDS *****

\ Send a reset high and low to port XOB.  The other bits in
\ port XOB are unaffected.  Note use of XC@ and XC! to interact
\ with “external” output ports at C020
: S8816_RESET ( )
      PORTXOB XC@ SRSTLINE OR PORTXOB XC!
      PORTXOB XC@ SRSTLINE NOT AND PORTXOB XC! ;

\ Put the specified bits in to the chip select lines.
\ SCS has CS2=0;  ACS has CS2=1
: SCS ( chip# - ) 8 * ( shift left 3 )
    PORTXOB XC@ CSMASK AND OR DF AND PORTXOB XC! ;

CREATE SSWITCHES[] ARRLEN ALLOT

\ Update SEXCOUNT with number of buses used
: SMASKY ARRLEN 0 DO SSWITCHES[] I + C@ OR LOOP ;
: SCOUNT 0 SEXCOUNT !
      0 SMASKY
      7 0 DO DUP 1 AND IF
            SEXCOUNT @ 1+ SEXCOUNT !
      THEN 2/
      LOOP DROP
;

\ SHOW will output the contents of all the bytes in the array
: SSHOW ( — )
    CR
    ARRLEN 0 DO
        I 10 < IF .”  “ THEN I . ( makes output pretty )
        SSWITCHES[] I + C@ BIN. CR
    LOOP ;

\ Connect n1 to the bus used by element n2
: SADD ( n1 n2 — )
    DUP SSWITCHES[] + C@
    IF SWAP DUP SSWITCHES[] + C@
        IF 2DROP .” Element 1 in use.” CR
        ELSE
            SWAP 2DUP SSWITCHES[] + C@ BUSSET ( set the bus )
            10 /MOD SCS JACKSET SETSWITCH ( set chip & jack )
            SSWITCHES[] + C@ SWAP SSWITCHES[] + C!
```

```
        THEN
    ELSE 2DROP ." Element 2 is not connected." CR THEN
;
\ Remove the specified element
: SREMOVE ( n1 — )
    SREDON
    DUP SSWITCHES[] + C@ IF   ( do some error checking )
        DUP DUP SSWITCHES[] + C@ BUSSET ( set bus )
        10 /MOD SCS JACKSET CLRSWITCH ( set chip & jack )
        SSWITCHES[] + 0 SWAP C!
    ELSE
        ." Element not connected." CR DROP
    THEN
    SREDOFF
;

\ Find the first free bus.  Return a bit mask
: SFINDBUS ( — n1 )
    1 ( used by second loop )
    0 ( used by first loop )
    ARRLEN 0 DO SSWITCHES[] I + C@ OR LOOP ( OR array elements )
    BEGIN
        2 /MOD SWAP
        IF
            SWAP DUP HIGHBUS =
            IF
                2DROP 0 1
            ELSE
                2* SWAP 0
            THEN
        ELSE
            DROP 1
        THEN
    UNTIL ;

\ Create a new bus and attach both elements to it
: SCONNECT ( n1 n2 — )
    SREDON
    DUP SSWITCHES[] + C@
    IF
        2DROP ." Element 2 in use." CR
    ELSE SWAP DUP SSWITCHES[] + C@ IF
        2DROP ." Element 1 in use." CR
        ELSE SWAP SFINDBUS DUP
        IF
          2DUP BUSSET
                10 /MOD SCS JACKSET SETSWITCH
                ROT DUP 10 /MOD SCS JACKSET SETSWITCH SWAP
            DUP ROT SSWITCHES[] + C!
            SWAP SSWITCHES[] + C!
        ELSE
           2DROP DROP ." All buses busy!" CR
    THEN THEN THEN
    CLEANUP   ( clear control bits )
```

```
    SREDOFF
;


\ Trigger the 8816 RESET lines and clear the array
: SRESET ( )
     SREDON
     SSWITCHES[] ARRLEN 0 FILL S8816_RESET CLEANUP
     0 SEXCOUNT !
     SREDOFF
;

\ Sexbar double-channel handling logic by SKR, 5/3/96...

\ NULLMODEM connects Rx to Tx and Tx to Rx, used when channels
\ are wired with Tx and Rx on the same-numbered pins
: NULLMODEM ( SXA SXB — )
     2DUP 20 + SCONNECT SWAP 20 + SCONNECT SCOUNT ;

\ STRAIGHT is the opposite of NULLMODEM, and connects 2 to 3,
\ 3 to 2.
: STRAIGHT ( SXA SXB — )
     2DUP 20 + SWAP 20 + SCONNECT SCONNECT SCOUNT ;

\ POLTEST is the channel polarity checker, connecting a specified
\ hardware X-line (1-3F) to Tx0 and reading the window comparator.
\ If a 1 is returned, then the channel is transmitting.  POLTEST
\ can be tested by passing ch#20, steady mark — should return 1.
: POLTEST ( ch# — flag )
     SEXCOUNT @ >R    \ minor kluge to save sexcount ;
     DUP 0 SCONNECT
     SENSPORT C@ TRSENSOR AND
     SWAP SREMOVE 0 SREMOVE
     R> SEXCOUNT ! ;

\ DSCONNECT is the big one, and connects the two specified serial
\ channel pairs either direct or swapped, depending on their
\ Tx-Rx wiring.  If both lines of either channel are zero or one,
\ indicating power off or hardware error, the error code
\ is returned.  Otherwise, the connection type code is returned:
\ 3 or C for nullmodem, 6 or 9 for straight connection.
: DSCONNECT ( SXA SXB — flag )
     2DUP 2DUP
     20 + POLTEST ( RxB )
     SWAP 20 + POLTEST ( RxA )
     2SWAP POLTEST ( TxB )
     SWAP POLTEST ( TxA ) ( stack: SXA SXB RxB RxA TxB TxA )
     SWAP 2 * + SWAP 4 * + SWAP 8 * +  ( flags -> byte )
     DUP 3 = IF ROT ROT NULLMODEM ELSE ( TxB & TxA = 1 )
     DUP C = IF ROT ROT NULLMODEM ELSE ( RxB & RxA = 1 )
     DUP 6 = IF ROT ROT STRAIGHT ELSE  ( RxA & TxB = 1 )
     DUP 9 = IF ROT ROT STRAIGHT ELSE  ( RxB & TxA = 1 )
```

```
      ROT ROT 2DROP ( return error code )
      THEN THEN THEN THEN
;


\ DREMOVE invokes REMOVE twice to nuke both physical channels
\ (Tx and Rx) of a logical channel.  Note that these may be
\ on different buses.
: DREMOVE ( SXn — ) DUP SREMOVE 20 + SREMOVE ;

\ DSREMOVE is the opposite of SCONNECT — removes all four
\ physical channels.
: DSREMOVE ( SXA SXB — ) DREMOVE DREMOVE SCOUNT ;

SRESET  ( Initialize on load )

\ auto-link Hub-Beeline on startup — Microship-specific ;
HEX
1E 1D DSCONNECT CR .( Beeline Connect: ) . CR
```

2.2. Multitasker FORTH Listing (optional)

As noted earlier, you can integrate your Sexbar controller into a multitasking environment if you want to have other tasks (such as the LED status indicator or an interactive front end) running concurrently. The following code is specific to the 68HC11 FORTH boards from New Micros, and is copyright by Bill Muench... any commercial use MUST be licensed (contact Nomadic Research Labs and we'll put you in touch with him).

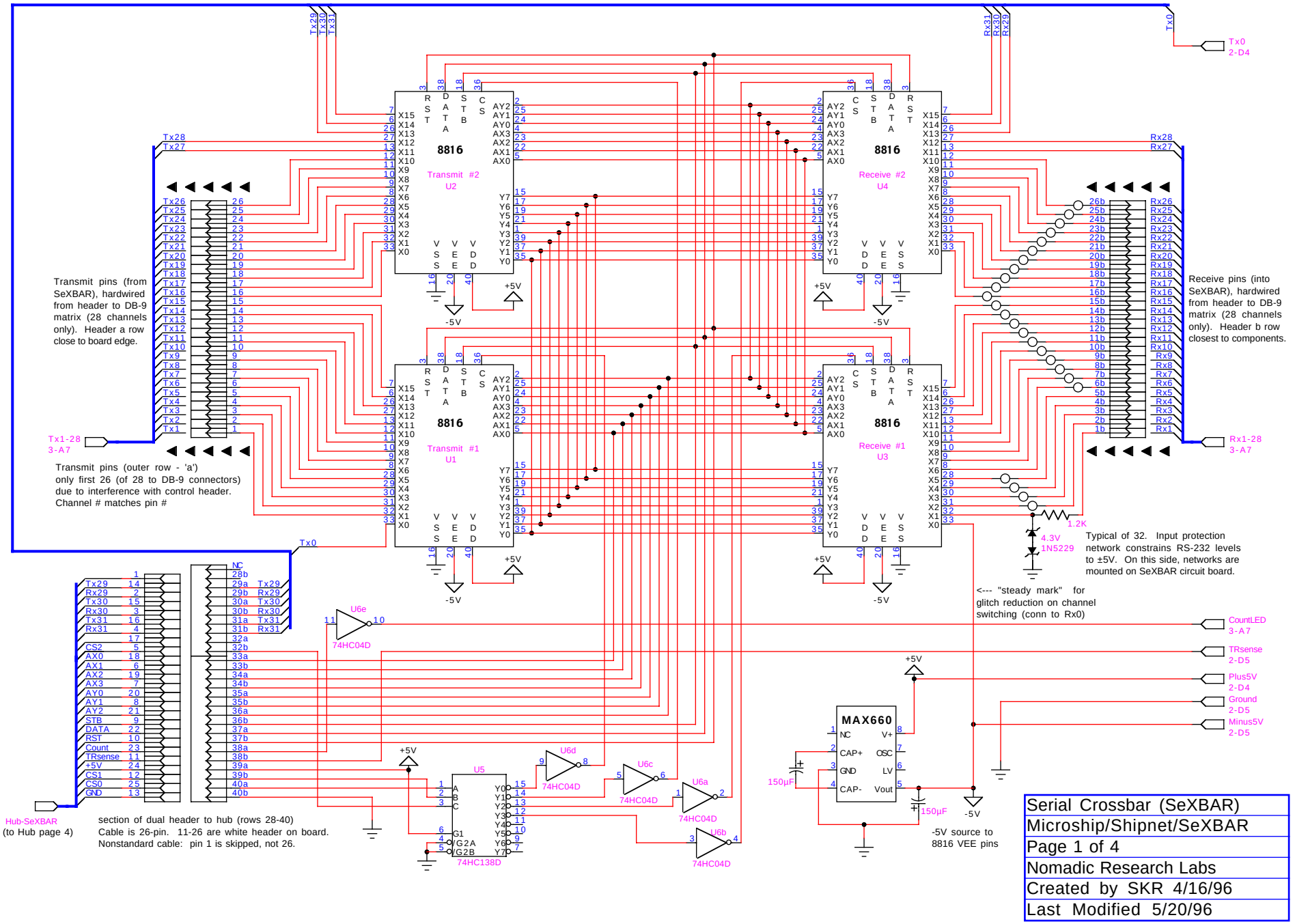
```
CR .( TASK.NMI  cooperative multitasker )
 ( .)
  CR .( Copyright 1990 Bill Muench All rights reserved. )
  ( .)
  ( 940224 code executive )
  ( 940124 update for NMI Forth 3.5E )
  ( 960703 update to add RESETTASKER )

( only MAIN task may use ?TERMINAL KEY EMIT )
( tid means task indentifier )
( tid=u/s/r may be in ROM )
( tid has pointers to RAM: user area, data stack, return stack )
( 3.5E MAIN  w/ip/up/user/—<pad/tib>—<r—<s )
( minimal task  user/12/—/16/—<s—/24/—<r )

FORTH DEFINITIONS

CREATE B.TASK ( marker )

( ========================================================== )

DECIMAL

0 USER STATUS ( either PASS or WAKE )
2 USER FOLLOWER ( next task's STATUS )
4 USER TOS ( top of stack on entry )
6 USER U1 ( free )

( ========================================================== )

HEX

0B7E CONSTANT PASS ( sev  jmp FOLLOWER )
AD00 CONSTANT WAKE ( 0 ,x jsr  FOLLOWER )

\ CODE wake ( — ) ( start next task )
\   pulx                ( addr = FOLLOWER )
\   dex dex  up stx     ( user base -> STATUS )
\   TOS STATUS - ,x lds  ( restore rp from TOS )
\   puly                ( restore sp )
\   pulx  inx inx  ip stx ( restore ip )
\   0 ,x ldx  w stx     ( save cfa in working 'reg' )
\   0 ,x ldx  0 ,x jmp   ( ITC next )
\ END-CODE
```

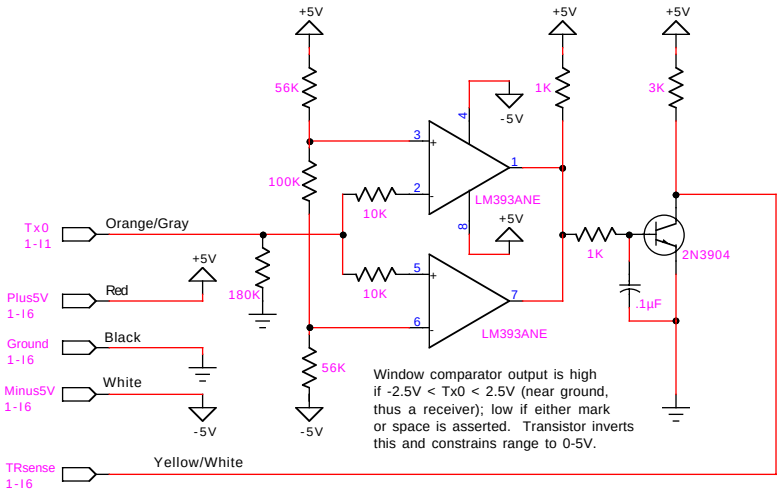
```
\ CODE PAUSE ( — ) ( allow another task to execute )
\   ip ldx  pshx             ( save ip )
\   pshy                     ( save sp )
\   up ldx                   ( user base )
\   TOS STATUS - ,x sts      ( save rp in TOS )
\   FOLLOWER STATUS - ,x ldy ( FOLLOWER = next task )
\   ‘ wake CFA @ ## ldx      ( setup for WAKE = 0 ,x jsr )
\   0 ,y jmp                 ( jump task table )
\ END-CODE

( ========================================================== )

CODE wake ( — )
  38 C,         ( pulx )
   9 C,         ( dex )
   9 C,         ( dex )
  DF C,   4 C, ( 4 stx )
  AE C,   4 C, ( 4 ,x lds )
  18 C,  38 C, ( puly )
  38 C,         ( pulx )
   8 C,         ( inx )
   8 C,         ( inx )
  DF C,   2 C, ( 2 stx )
  EE C,   0 C, ( 0 ,x ldx )
  DF C,   0 C, ( 0 stx )
  EE C,   0 C, ( 0 ,x ldx )
  6E C,   0 C, ( 0 ,x jmp )
END-CODE

CODE PAUSE ( — )
  DE C,   2 C,          ( 2 ldx )
  3C C,                 ( pshx )
  18 C,  3C C,          ( pshy )
  DE C,   4 C,          ( 4 ldx )
  AF C,   4 C,          ( 4 ,x sts )
  1A C,  EE C,   2 C,   ( 2 ,x ldy )
  CE C,  ‘ wake CFA @ , ( ‘ wake CFA @ ## ldx )
  18 C,  6E C,   0 C,   ( 0 ,y jmp )
END-CODE

( ========================================================== )

: ‘S ( tid a — a ) ( index another task’s local variable )
  STATUS - SWAP @ + ; ( PASS  TASK1 STATUS ‘S  ! )

: AWAKE ( tid — ) WAKE SWAP STATUS ‘S ! ; ( wake another task )
: SLEEP ( tid — ) PASS SWAP STATUS ‘S ! ; ( sleep another task )

: ACTIVATE ( tid — )
  DUP 2+ 2@        ( tid sp rp )
  2- R> OVER !     ( save ip on rp and setup exit )
  2- SWAP OVER !   ( save sp on rp )
  1- OVER TOS ‘S ! ( save rp-1 in tos )
```

```
  AWAKE ;

: STOP ( – ) PASS STATUS ! PAUSE ; ( sleep current task )
: COMA ( – ) BEGIN STOP AGAIN ;
: HALT ( tid – ) ACTIVATE COMA ;


( shared resources ======================================== )

( a simple semaphore is just a VARIABLE )

: GET ( semaphore – )
  PAUSE  DUP @ STATUS XOR ( owner? )
  IF BEGIN DUP @ WHILE PAUSE REPEAT ( no, wait for release )
    STATUS SWAP ! ( lock ) EXIT
  THEN DROP ;

: RELEASE ( semaphore – )
  DUP @ STATUS XOR IF DROP EXIT THEN  0 SWAP ! ( unlock ) ;

( ========================================================== )

: HAT ( u s r “name” – ) ( – tid )
  CREATE HERE >R 6 ALLOT  RAM HERE R@ ! ( user )
  >R + 01C + ALLOT HERE ( user/12/–</16/sp)
  R> 018 + ALLOT HERE ( –</24/rp )  R> 2+ 2! ;

: BUILD ( tid – ) ( sleep and link new task )
  DUP 2@ SWAP OVER - 0BB FILL ( debug tracer )
  DUP SLEEP  DUP @ ( u ) >R  2+ 2@ ( s r )
  R@ [ R0 STATUS - ( offset ) ] LITERAL + 2! ( r0 s0 )
  FOLLOWER @  R@ FOLLOWER !  R> 2+ ! ( links ) ;

( ========================================================== )

CREATE MAIN  4 ( UP ) @ , S0 @ , R0 @ , ( point to the NMI defaults )

: RESETTASKER ( – ) ( remove all TID from tasker )
     6 4 ! ( default user base address )
     STATUS FOLLOWER ! ( init FOLLOWER )
     MAIN AWAKE  ( should always be awake )
     [ ‘ PAUSE CFA ] LITERAL ‘PAUSE ! ( set PAUSE vector ) ;

RESETTASKER
```

```
( ========================================================== )

: TASKS ( — ) ( display status of all currently linked tasks )
  BASE @  STATUS  DUP
  BEGIN CR 2+ @  DUP HEX 6 U.R SPACE ( next status addr )
    DUP @ WAKE XOR IF ." PASS" ELSE ." WAKE" THEN
    ."  depth="  2DUP =
    IF DEPTH
    ELSE DUP [ S0 STATUS - ( offset ) ] LITERAL + @
      OVER [ TOS STATUS - ( offset ) ] LITERAL + @ 1+ @ -  2/
    THEN DECIMAL 0 .R  2DUP =  NUF? OR
  UNTIL 2DROP  BASE ! CR
;

: .TID ( tid — ) ( display the task table )
  2@ SWAP OVER - DUMP ;

( ========================================================== )
```

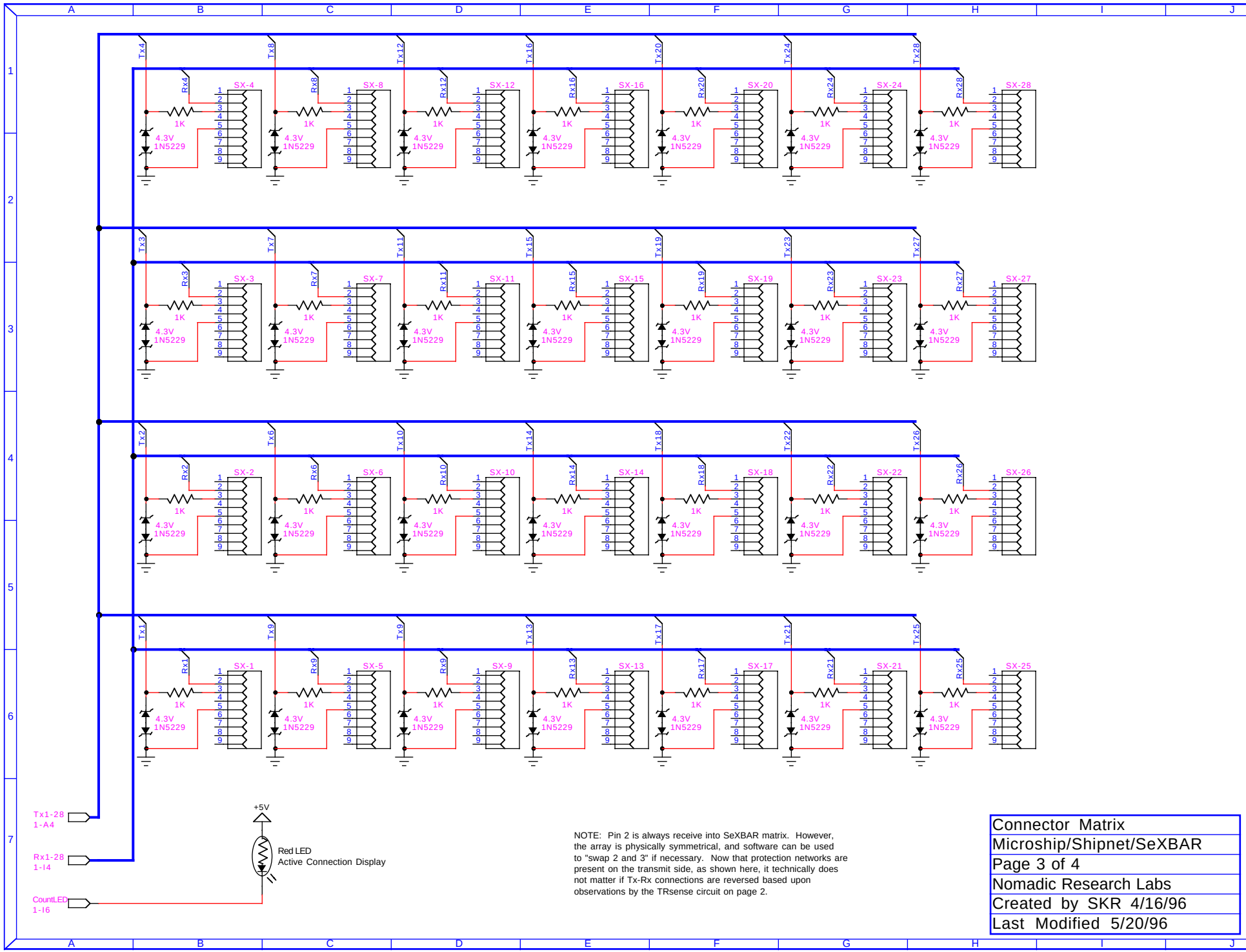
on following page


Transmit pins (from SeXBAR), hardwired from header to DB-9 matrix (28 channels only). Header a row close to board edge.
Transmit pins (outer row - 'a') only first 26 (of 28 to DB-9 connectors) due to interference with control header. Channel # matches pin #
Receive pins (into SeXBAR), hardwired from header to DB-9 matrix (28 channels only). Header b row closest to components.
Typical of 32. Input protection network constrains RS-232 levels to ±5V. On this side, networks are mounted on SeXBAR circuit board.
<--- "steady mark" for glitch reduction on channel switching (conn to Rx0)
Hub-SeXBAR (to Hub page 4)
section of dual header to hub (rows 28-40)
Cable is 26-pin. 11-26 are white header on board.
Nonstandard cable: pin 1 is skipped, not 26.
-5V source to 8816 VEE pins
Transmit #2 U2
Receive #2 U4
Transmit #1 U1
Receive #1 U3
8816
74HC04D
74HC138D
MAX660
Tx0 2-D4
Tx1-28 3-A7
Rx1-28 3-A7
CountLED 3-A7
TRsense 2-D5
Plus5V 2-D4
Ground 2-D5
Minus5V 2-D5
Serial Crossbar (SeXBAR)
Microship/Shipnet/SeXBAR
Page 1 of 4
Nomadic Research Labs
Created by SKR 4/16/96
Last Modified 5/20/96



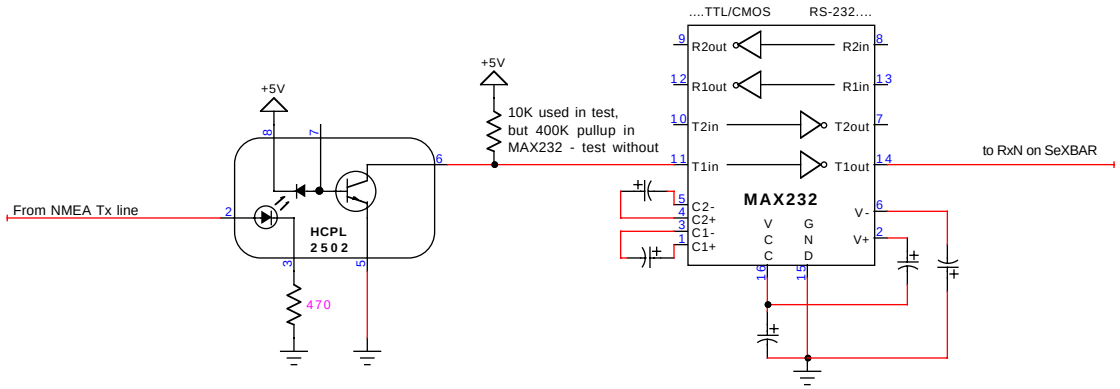
NOTE: T-R sensor is a perfboard subassembly piggybacked on the SeXBAR matrix board.

T-R Sensor
Microship/Shipnet/SeXBAR
Page 2 of 4
Nomadic Research Labs
Created by SKR 4/26/96
Last Modified 5/20/96

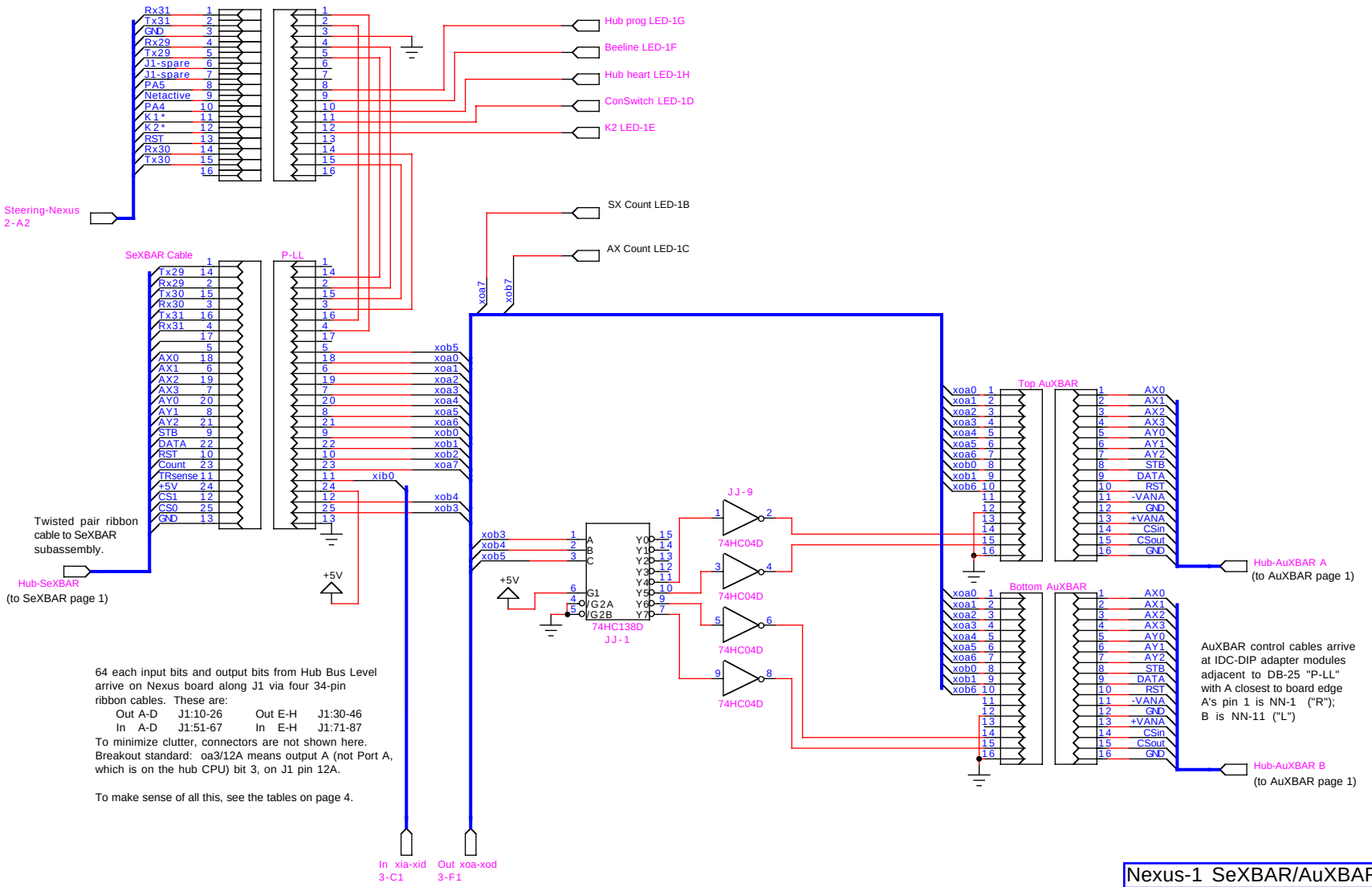


NOTE: Pin 2 is always receive into SeXBAR matrix. However, the array is physically symmetrical, and software can be used to "swap 2 and 3" if necessary. Now that protection networks are present on the transmit side, as shown here, it technically does not matter if Tx-Rx connections are reversed based upon observations by the TRsense circuit on page 2.

Connector Matrix
Microship/Shipnet/SeXBAR
Page 3 of 4
Nomadic Research Labs
Created by SKR 4/16/96
Last Modified 5/20/96



NMEA-RS232 Interface
Microship/Shipnet/SeXBAR
Page 4 of 4
Nomadic Research Labs
Created by SKR 4/16/96
Last Modified 5/20/96



Nexus-1 SeXBAR/AuXBAR
Microship/Shipnet/Hub
Page 5 of 5
Nomadic Research Labs
Created by SKR 5/1/96
Last Modified 7/21/97

A. ACKNOWLEDGMENTS

The crossbars were, collectively, a huge project that has its roots in the BEHEMOTH bicycle era (1991) and later evolved with the aid of FORTH experts, UCSD students, sponsors, and other volunteers. They deserve a lot of the credit for their contributions to the design in this book...

Auxbar Volunteers and Sponsors

BEHEMOTH ERA:

Steve Sergeant — analog audio system design and testing
Amber — Model 3501 noise and distortion measurement system
Bob Lockhart — printed circuit CAD artwork
Mesa Reprographics — printed circuit films
Sun Circuits — circuit board manufacture
Joe Dunn — board stuffing volunteer
Mike Perry — initial FORTH control code

MICROSHIP ERA:

Jason Corley — UCSD student design team: FORTH code
Isaac Chu — UCSD student design team: wiring and testing

Sexbar Volunteers and Sponsors

David Wright — zener level constrictor idea
Perry and Corley (see above) — FORTH code ported from Auxbar
Dan Sebald — UCSD student team: board fabrication
Jeff Simon — UCSD student team: board fabrication
Jim Rees — start of NMEA interface board
Chris Burmester — NewtonScript front-end macro tool

Vixbar Volunteers and Sponsors

Delon Levi — UCSD student engineer: FORTH code & wiring

Sponsors Supporting All Three Subsystems

New Micros — 68HC11 FORTH boards and accessories
Bill Muench — multitasker and multidrop software
Mitel Semiconductor — 8816 and 88V32 crosspoint chips
Halted Specialties — miscellaneous electronic parts

B. REFERENCES

Mitel 8816 data sheet: Mitel *Digital/Analog Communication Handbook*, Issue 9, page 7-51. --->http://www.mitelsemi.com/products/pdf/dataserv.cgi?mt8816

The Art of Electronics, Second Edition, Horowitz and Hill. Window comparator circuit on page 669.

LM393 datasheet: National Semiconductor *General Purpose Linear Devices* databook, 1989, page 5-61.

MAX660 (ICL7660) datasheet: Maxim 1989 Integrated Circuits Handbook, page 6-117. The 660 is spec'd at 100mA.

MAXC001 Low-ESR Aluminum Electrolytic Capacitor datasheet.

The Sexbar

- **networking low-level serial devices**
- **random interconnections under software command**
- **distributed sensors and navigation instruments**
- **hacking environments**
- **industrial control**

This NRL technical monograph covers one of the most useful spinoffs of Microship system development -- the serial crossbar network. Presenting a simple command-line interface on one end and a matrix of DB-9 connectors on the other, this handy gadget eliminates the usual nightmares of random RS-232 connections... swapping pins 2 and 3, fiddling with gender changers, and assembling chains of cables to get something to work. If the plugs fit and the baud rates match, any two devices will talk — the Sexbar even figures out which pins are transmit and which receive, then connects them accordingly!

The design included in this book provides 32 channels (of which 31 are actually available), and up to four simultaneous bidirectional connections can exist among any of them. This is easily scalable if your application calls for more or less capacity.

It may seem anachronistic, in these days of ubiquitous and cheap 10 megabit/second ethernet, to be building networking tools for plodding old 9600-baud RS-232 devices. But we are surrounded by simple gadgets that depend on vanilla serial interfaces, from the COM ports on PC's to a whole population of GPS receivers, sensors, device controllers, packet radios, speech synthesizers, printers, standalone LCDs, security systems, battery monitors, embedded micros, and so on. None of those things have the horsepower to run TCP/IP protocol stacks and support 10baseT connections. So if you find yourself surrounded by serial widgets and are constantly fishing for the right cables and adapters to make them talk, then you need the Sexbar.

Your purchase of this book gives you license to construct a unit from our design and use the included software for any non-commercial application. (If you want to sell them or include the design in a product, we welcome licensing opportunities!) We would appreciate your registering informally by sending email to wordy@qualcomm.com

Please contact NRL for current information on other technical monographs, and ask for our free *Microship* project electronic updates...

Steven K. Roberts
Nomadic Research Labs

www.microship.com
wordy@qualcomm.com

Tel: 360-387-1440
Fax: 360-387-2429