USER AND SWITCH PACKET DIGITAL HARDWARE

Terry Fox, WB4JFI
President, AMRAD
1819 Anderson Rd.
Falls Church, VA 22043

ABSTRACT

There has been a lot of activity in the area of digital hardware for Amateur packet radio in the last year. I see this trend continuing over the next few years. This paper describes some of the present packet digital hardware, and gives some of my thoughts on what we will be using in the future, both for the "TNC" and the network devices.

Terminal-Node-Controllers and Packet Assembler/Disassemblers

Let me first clarify what I mean by a Terminal-Node-Controller (hereafter called a TNC), and a Packet Assembler/Disassembler (called a PAD). I have altered the meaning of the TNC from its "traditional" meaning of any device an Amateur uses to gain access to the packet network. I prefer to use the term TNC from now on to describe any device that is used to provide a level 2 only connection for some non-packet device, such as Figure 1 illustrates. Now I hear some of you saying "WRONG, WRONG, WRONG!" I know this is not the "traditinal" Amateur meaning for this device, but with true networking devices on the way, I feel we need to alter the meaning of TNC to level 2 only devices.

```
                 ---------------
                 |             |
Level 2 --->     !     TNC     |  --->  User
                 !             |        Device
                 ---------------
```

Figure 1. TNC Type Device

The reason
description to level 2 only devices is that the term "Packet Assembler/Disassembler" or PAD is normally used to describe a device that makes and tears apart "packets", or Level 3 pieces of information for some non-packet device. I suggest that we adopt this PAD description for any device that interfaces a User Device to the Amateur

indicates.

```
                 ---------------
                 |             |
Level 3 --->     !     PAD     |  --->
Interface        !             |        Device
                 M--------m-----
```

Figure 2. PAD Type Device

This separation of definitions of the terms TNC and PAD will pay off in the near future when we have both Level 2 and Level 3 devices trying to hook to the Amateur network. Since most TNCs are Level 2 only devices at the moment anyway, this should be a fairly smooth change. It should be noted that a piece of hardware could actually be one or the other of these. The TAPR TNC2 is a good example of this. Using the standard EPROMs supplied by TAPR, the TNC2 functions as a TNC. If those EPROMs are replaced by ones with AX.25 Level 3 code in them, the same TNC2 would then become a PAD for the user.

TNC's and PAD's fall into two basic categories; the dedicated box, and the computer adaptor. Until about a year ago, only the dedicated box type was available for general use. The dedicated box TNC is a separate microcomputer whose responsibility is to provide a standard terminal or computer (usually via a RS-232 serial port) access to the Amateur Packet Network. In order to do this, the dedicated box has its own CPU, RAM, program in EPROM, and a couple of serial ports (one for the packet channel, the other for the User Device interface). The advantage of this type of device is that it will run with almost any terminal or computer, and will not tie up the processing power of a host computer it is attached to.

The computer adaptor is a much simpler device, at least in hardware. It's kind of
that since there's less hardware involved, more of these interfaces haven't come along. The adaptor hardware is usually nothing more than some sort of serial chip capable of supporting
drivers. The problem with the computer adaptor is that the support software for Level 2 must be written for, and must run on the host processor. This will take away processing power from the host processor. Also, if the packet operation is not the end function but rather a communications medium for another program (such as a bulletin board), a major conflict can arise between the computer adaptor software and the other program.

Today's Dedicated TNC Devices

It is interesting to note that a couple of years ago, when I designed the AMRAD PAD (which was still-born) I kept saying that the number one problem with the TNCs then available was the lack of memory, both RAM and EPROM. Boy, did I get arguments about that. Well, guess what? Todays generation of TNC boards have more memory, sometimes much more.

One example of this is the new TNC+ available from Vancouver. It has eight byte-wide sockets for memory, allowing up to 64k of memory to be used, in almost any mixture of RAM and EPROM chips. In contrast, the original Vancouver TNC (the one that started it all) had only 4k each of RAM and EPROM.

Some of the dedicated TNCs available available at the time this is being written are listed below. Note that this may not be a complete list, but it will give the reader a good indication of the types available these days.

Vancouver Amateur Digital Communications Group (VADCG) has a new version of the original TNC. The TNC+ is based on the same 8085 CPU and 8273 HDLC chip design as the original. The major improvements are the use of eight byte-wide memory sockets for up to 64k of combined EPROM/RAM, and the use of a single power supply input. It is available from Lockhart Technology at the following address:

Lockhart Technology
9531 Odlin Road
Richmond, B.C. Canada V6X 1E1

Bill Ashby and Son sells a version of the original Vancouver TNC which uses a different kind of memory devices but is otherwise compatible with the original Vancouver TNC from:

Bill Ashby and Son
Box 332
Pluckemin, NJ 07978

Heathkit still sells their version of the TAPR TNC-1, called the HD-4040. This is a 6809/WD1933 based TNC. For more information, write to:

Heath Electronics
Benton Harbor, MI 49022

Another TNC based loosely on the TAPR TNC-1 design is the Kantronics Communicator. It is a 6803 based TNC, which uses a software UART instead of the WD1933 that is in the TNC-1. Apparently, Kantronics has just introduced a new Communicator, the KPC-2. At this time, I haven't seen one of these yet, so I can't tell how different it is. For more information, contact Kantronics at:

Kantronics
1202 E. 23rd Street
Lawrence, KS 66046

Another TNC that uses a software HDLC UART is the GLB PK1-L. It is Z80 based, and the low-power version draws only 25ma at 12VDC. Contact GLB at:

GLB Electronics
151 Commerce Pkwy
Buffalo, NY 14224

I should mention that TAPR is no longer selling the TNC-2 directly. They have licensed the product to several companies to produce. This TNC uses a Z80 CPU, an SIO for both serial ports, a 32 kilobyte EPROM, 8k of RAM, and the standard TAPR design Exar chip modem.

One of the companies selling the TNC-2 clone is GLB. It is called the TNC-2A.

Another company that is producing the TAPR Clone is the MFJ Enterprises MFJ-1270. Contact:

MFJ Enterprises, Inc.
Box 494
Mississippi State, MS 39762

Yet another TNC-2 clone is the PK-80, from AEA. AEA was one of the companies that sold a version of the original TAPR TNC-1, so they have been into packet radio for a while. For more info, contact:

Advanced Electronics Applications
PO Box C-2160
Lynnwood, WA 98036

Another TNC-2 clone is the TNC-200 from Pat-Comm. It is available in various forms, from bare boards to assembled units. Write to:

Pat-Comm Packet Radio Systems, Inc
4040 W. Kennedy Blvd
Tampa, FL 33609

One company, Packeterm, sells an integrated packet terminal, called the IPT. This is a whole computer terminal with a packet board included. For more info, write to:

Packeterm
PO Box 835
Amherst, NH 03031

More and more of these dedicated TNC devices should become available as packet radio continues to grow. One of the ideas I have is to design a dedicated TNC using a single chip microcontroller. Now that AX.25 Level 2 Version 2 is becoming fairly stable, I think it is time to have the protocol available on silicon. This way, anyone who wants to run the protocol can obtain one of these single-chip controllers with the code in it, and be confident that it will be compatible with others (once the initial bugs are worked out). This same device could be used stand-alone as a TNC, or as a front-end packet processor for a larger device such as a packet switch. I will be working more on this in the near future, probably based on the Intel line of microcontrollers, such as the 80C152.

The 806152 is similar to the popular Intel 8051 type microcontroller chip. It has a CPU, 8k of either EPROM or mask-programmable ROM, 256 bytes of RAM, two timers, async serial channel, and an HDLC serial channel supported by two DMA channels. This chip should allow the whole TNC to be done with about six chips, including modem.

Packet Adaptors Available

The packet adaptor devices are relatively new to the packet scene. Probably the first of these was the state machine add-on to the Xerox 820 designed by Jon Bloom, et all. Then came the FADCA/TAPR Frame Assembler/Disassembler (FAD), also for the Xerox 820. Phil Karn, KA9Q has written AX.25 Level 2 code in C for the Xerox 820. The combination of one of these two hardware mod's and Phil's software makes a good alternative to a dedicated TNC.

The first commercial packet adaptor that I am aware of is the AEA Pakratt PK-64. This device plugs into the expansion socket of a Commodore 64, and supports not only packet operation, but also RTTY, ASCII, and AMTOR. Once again, it provides a HDLC capable serial port (Zilog 8530 SCC) and modem for hardware, and canned software in EPROMs. Since the Pakratt is used only on the C-64, it can have machine-specific code in it, such as a nice menu-driven user interface.

Another packet adaptor recently announced, is one that is designed for the IBM PC and clones. It was developed in Canada, and is based on the Intel 8273 HDLC controller chip, and the Exar chip set modem. At this time, I don't have the address of where to order this device.

I have also come up with a design for a packet adaptor for the IBM PC. It uses an 8530 SCC for the HDLC channel(s). It also has room for two AMD 7910 world-modem chips, and RS-232 drivers for both channels. It is called the PC-PAD-1, and will be available from TAPR shortly.

These adaptor devices can make the entry to packet radio much cheaper if one is available for a computer you already own. As mentioned earlier, the problem with these devices is that the host computer will probably be so busy running the packet software necessary, that it won't be able to run other programs. While this is fine for an Amateur wishing to communicate over packet radio, it is not so good if the host computer must also run some other program, such as a bulletin board, or possibly some emergency database program.

Packet Switch Hardware Designs

It is likely that before long true packet switches will be replacing our present digipeaters. These packet switches will be more sophisticated devices, requiring more memory, larger program storage, and larger data storage, and more RF channels and control circuitry. For more information on what I see is needed in software, see a companion paper titled "Packet Switch Software Design Considerations" elsewhere in these proceedings.

Packet Switch Hardware Requirements

In order to properly design any complicated device, one should first look at what is ideally expected from that device. If all the functions are not necessary immediately, a graduated design may be in order. This is what I think we need to do with the design of the packet switch.

Some of the requirements for the ultimate packet switch are as follows:

Lots and lots of RAM memory for data.

Multiple Channel Operation (probably at least four channels, each potentially full-duplex).

Mass storage device(s) for routing and User directory storage, plus program loading.

Fast main processor capable of accessing large amount of memory.

Program storage either in EPROM or mass-storage.

All input and output should be under DMA control to reduce main processor load.

Service processor and control system for diagnostics and control of the switch.

Time-of-day clock, and various timed interrupts, as necessary.

-------- continued next page ---------

Local console interface for local access to switch.

Low power and/or battery back-up operation.

Lots and lots of RAM

The amount of RAM depends on how many stations are going to be using the switch, how the switch software is written, and to some degree what protocols are to be used. A full level 2 AX.25 frame uses about 275 bytes without digipeaters, or up to 335 bytes with the full complement of digipeater addresses. If one allows for a window of 7 packets in each direction (transmit and receive), this can mean from 3.6k to 4.7k bytes per user, just for actual data buffers. There will also be some small amount of storage per user at each level of protocols used. With between 4 and 5 kilobytes needed per user, one can see that the more the RAM, the better in a packet switch. Obviously, this is a worst-case scenario, and not all this memory must be reserved per user at all times. A guess on my part is that about 20 percent of the absolute amount of memory would actually be needed. If this holds true, 128k of RAM could actually support up to 150 users rather than the 32 users suggested by setting aside 100 percent of memory for all users.

Of course, this evaluation does not hold true if a lot of data latency is encountered inside the switch. As an example, say a switch receives a lot of packets from a 56 kbps channel, and finds that the majority of the data is to sent out an HF channel operating at 300 bps. This means much of the data must be buffered for a longer time while the level 2 protocol tries to shove the data over the HF link. If the switch starts to run out of memory, it may have to begin to flow-control data coming into it until the HF link frees up some buffers. This may cause back-pressure of data along the Network as more and more switches have to hold data until the next switch can accept it.

Another use for memory in switches will be to hold the operational routing tables, which can grow to rather large sizes. The reason for maintaining the operational routing tables in memory rather than a mass-storage device is due to the need for continual, fast access to them. A back-up version of the tables should be maintained on a mass-storage device (for power-failure or data corruption protection), but the main, active tables should be in RAM. This adds another, rather significant memory requirement to the switch design.

These days memory is fairly inexpensive, especially dynamic memory. It therefore makes good sense to me that a switch have as much RAM as possible, preferably as much as the CPU can easily handle.

Multiple Channel Capability

Before long, packet switches may have to monitor several different RF channels, much as the dual-port digipeater does now. Some of these channels may be operating full-duplex, especially the higher speed ones. As a guess, I think four to five channel capability should be enough for the most part, with the possibility of expanding up to six channels. The higher speed channels should either use DMA directly, or use a smaller TNC-type device to pre-process the level 2 connections, thereby relieving the main processor of some processing.

Mass-Storage Device(s)

Once the Amateur packet network begins to expand, there will be a need for some sort of mass-storage of data at each switch. One of the things needed to be stored at each switch is a routing table that the switch can access to decide how to route packets. Another thing the switch may want to store is a file containing the list of users it regularly services. Also, the switch may keep a couple versions of its operating program in case it need to re-boot for some reason.

Fast Central Processor

In order to build a switch that can process data quickly, a fast central processor (CPU) should be used. Since newer, faster CPUs are constantly being developed (which are usually very expensive at first), we may not be able to afford the "best-of-the-line". An important factor when choosing a CPU is that the development tools must be readily available for designing both hardware and software around the chosen CPU. Some of these tools include simple hardware interfacing to standard components, and Assembler and C compilers for software, at a minimum. The best CPU in the world isn't much good if one must write all code for it in hand-assembler.

Program Storage in EPROM or Mass-Storage

There should be room in the switch for a minimal amount of bootstrap code in EPROM to load the main operating program either from a mass-storage device, or possibly from a neighbor switch. Another alternative would be to have the main operating program run directly from EPROM, but this would make remote updating of switch software more difficult. since switches may be at remote locations, some method of remote loading of switch code may come in handy.

All I/O Should use DMA If Possible

As mentioned earlier, all input and output devices should operate using Direct-Memory-Access (DMA) if possible. This would include packet channels and any mass-storage devices used. The use of DMA relieves the main CPU of having to handle constant asynchronous data interrupts, or worse, polling of I/O devices, both of which can waste a lot of CPU time. DMA chips are available today at an inexpensive enough price to make a hardware design using them cost-effective.

Service Processor and Control System

When switches get more complicated, it may become necessary to provide some method of monitoring switch operation and controlling the switch if a failure is detected. One way to do this would be to provide a second, smaller micro-computer system to monitor the packet channel(s) and/or the main switch processor bus for normal operation. This separate system would have its own distinct identity in the network, and should be designed so that the main switch could not block operation of the Service Processor. If a failure is detected, the Service Processor could have several options programmed into it, from simply re-booting the main switch to taking the switch completely off the air and notifying someone of its actions and why.

This type of device is commonly used on larger computer systems to monitor the main computer, and I think we should include some version of this device in our more elaborate switches. Since the main switch and the Service Processor are inter-related, they would have to be designed with each other. At the moment this device is a luxury, but it should be kept in mind for future switch designs.

Time-Of-Day Clock and Timed Interrupts

In order for a switch to accurately control the various protocol operations, some sort of timer is necessary. Most computers use one or more timed-interrupts to control these functions. In a multiple-task computer, times cannot be regulated by software timing loops, so these timed interrupts are important. For example, the TAPR TNC2 uses a 600 Hz interrupt, while the Xerox 820 computer board uses a once-a-second interrupt. These are generally considered the two ends of the spectrum for timed interrupts, with the TNC2 being almost too fast, and the Xerox 820 being too slow.

The time-of-day clock is less of a necessity, but could come in very handy, especially if satellite or HF access is to be provided at the switch. It also may be handy for diagnostic purposes to time-stamp certain operations of the switch, such as routing table updates and last access by individual users. Once again, the cost of adding this feature is very small, so why not include it? Most time-of-day clock chips available these days also provide timed-interrupts, so both of these can be had for the price of one chip.

Local Console Interface

It is inevitable that switches are going to break down eventually. In order to work on them, or just periodically check on their operation, a local console interface should be provided. This local interface should be at a very base level of the switch (directly into the CPU if possible) to reduce the amount of potential problem points between the console and the local console while debugging.

Battery Back-Up Operation (optional).

There is a rapidly growing interest in the use of packet radio by emergency groups. In order to help provide emergency communications, the packet network should be capable of operating for some period of time either on batteries, or through the use of generator power. While this isn't a definite requirement, if the switch is designed with battery operation in mind (such as the capability of running off of 12 volts DC), it will be much easier to add later.

The above items do not constitute a complete list of design goals for a switch. The radios, for example, are partially covered in a separate paper, found elsewhere in these proceedings.

With the above goals in mind, I will now turn to how I see the packet switch design evolving over the next few generations of hardware.

Packet Switch Based On TAPR TNC2

The first step of the packet switch
Thanks
to Howie, N2WX, several of us in AMRAD (as well as others) have been playing with AX.25/AX.75 networking on a local basis for several months now. The packet switch code actually resides inside a TAPR TNC2. I must point out that this is by no means a fully operational internetwork switch capable of auto-routing and lots of other fancy stuff, but it IS an actual Network switch, not a digipeater. Full AX.25 level 2 connections are made with each user, and the switch does move data from user to user based on AX.25/AX.75 level 3 connections. As a Local Network Controller for a limited number of users it does indeed work.

The amount of RAM on a present TNC2 is 16 kilobytes, with 32k of EPROM for program memory. The TNC2 uses a Z80 CPU, and an SIO for both terminal communications and packet communications. This amount of memory (especially the RAM) is sufficient to experiment with, but could hardly support the total number of users on even a semi-busy packet channel. Since there is no mass-storage device interface on the TNC2, some of this RAM would have to be used up to hold the routing table for inter-switch connections, reducing even further an already scarce commodity.

Also, since there is only one (or with modification and loss of the terminal interface, two) packet interface(s), a switch based on the TNC2 could only operate directly on one packet channel, without getting into clever data switching arrangements which would be a kludge. The maximum speed of the TNC2 serial channels is probably around 20 kbps, allowing it to be used on our medium-speed channels.

The packet channel interface operates in an interrupt mode, rather than using DMA. This is not really a problem though, since there is only one packet channel, one terminal channel, and no mass-storage devices to worry about. As mentioned earlier, the TNC2 has timed interrupts at a 600 Hz rate, relieving the software of a lot of timing overhead.

The TNC2 uses a 12 Volt DC nominal power supply, drawing from 150 to 300 ma, which makes it very good in the power department.

Packet Siwtch Using the Xerox 820

Another alternative for a packet switch from the 8-bit world involves the use of the Xerox 820 boards. These boards were surplused by Xerox after the 820 was discontinued, and a large number of them found their way into the hands of packet experimenters throughout the country. Some of the advancements in packet radio operation in the last year or so (such as the WORLI packet bulletin-board software and the dual-port digipeater) are directly related to the availability of these boards. It has on it a Z80 processor, two 2716 EPROMS, 64k of dynamic RAM, two serial channels, a parallel channel, four timers, a floppy-disk interface, a parallel keyboard interface, and a video interface that emulates an ADM-3 terminal. Many of these 820 boards are already out there as either PBBS systems, or dual-port digipeaters linking two different packet operations together.

It is natural that the Xerox 820 be considered as a possibility for the basis of a packet switch. In fact, at the September 1984 meeting of the ARRL Digital Committee, one of the requirements made for the evaluation of Virtual-Circuit network design versus Datagram network design was that they both be made to run on a Xerox 820 board.

The Xerox board's 64k memory space is better than the TNC2's 16k. One drawback to this is that a Z80 processor can only access a total of 64k of memory at a time, including both program and data space. This means that the more memory a program takes, the less that is available for data. This implies that the program used should be as small as possible to save more memory for data.

The Xerox 820 includes four timers, two of which are used to generate one-second interrupts for the 820's monitor. The other two timers are available to the user. One of these will probably be needed as a prescaler for the other in order reduce the number of interrupts per second achieved. The once-a-second interrupt could be used to build a software time-of-day clock, although it would be susceptible to inaccuracies due to periods when interrupts are disabled.

The serial ports on the Xerox 820 are interrupt driven, using an SIO chip. TAPR has an add-on daughter board available that removes the parallel interface and replaces it with two more serial channels, using an 8530 SCC in interrupt mode. This means that one Xerox 820 can support up to four different packet channels, which might come in handy at a central packet switch location.

While the 820 board does have a floppy-disk interface, it is only capable of handling single density operations. This means a total of 241 kbytes of storage on an eight-inch drive, or about 160 kbytes on a double-sided five-inch floppy. There are several daughter board systems commercially available that replace the single-density controller chip on the 820 with a double-

capacity to 390 kbytes of storage. Some of these

incompatibility has been noticed, particularly in
These incompatibilities
can be overcome without too much difficulty, and the added storage is a welcome addition.

When using CP/M on the Xerox 820, there is a wealth of software tools available for switch software development. All kinds of assemblers and higher language compilers (such as C, Pascal, Fortran, and yes, even BASIC) are available. In addition, there are a large number of debugging systems available to the programmer, so the 820 can be considered a good choice to develop code on.

I still think the Xerox 820 is a good choice for smaller switches, where there will not be a lot of users. The boards usually came built, with most of the ICs soldered in place, so the number of problems occuring with a switch based on an 820 will probably be less than some other designs. (It is interesting to note that half the problems we have experienced with the WB4JFI-5 digipeater are due to corrosion of IC socket-to IC pin connections, and that as soon as the ICs are re-seated, the problems disappeared).

Packet Switch Design Based On the TAPR NNC

A short time ago, TAPR came up with a design for what they call the Network Node

Controller, or NNC. This is the same thing as a packet switch. The NNC design is based on a Z80 like chip made by Hitachi (the HD64180). This CPU executes the Z80 instruction set, plus a few new instrucions that have been added. In addition, it includes two channels of DMA, 2 sixteen bit timers, two async serial channels, a built-in clock generator, and can physically access up to 512 kbytes of memory (through a 64 kbyte logical window). It also has a built-in interrupt controller, and dynamic memory refresh controller.

TAPR's design presently consists of three boards. The first board includes the 64180 CPU, four HDLC ports (using two SIOs), a parallel printer port, a battery backed-up real-time-clock, a SCSI bus Interface, and up to 512 kbyte of memory via byte-wide sockets, half of which may be battery backed-up.

The second board is a piggy-back floppy disk controller board. This board will allow the connection of high-density disk drives for those wishing to do software development directly on the NNC, via CP/M or similar operating systems. It will also come in handy to hold those routing tables.

The third board contains four modems of the Exar 2206/2211 variety. TAPR sees these being used as an HF modem (300 bps, 200 Hz shift) and a VHF modem (typical 1200 bps, 202 type), with two additional modems for future use. To me, these modems are still the weak link of TAPR designs. The PLL modems are inferior in operation to standard filter-type modems, and require periodic re-alignment (I just had to re-align my first TNC2). I am not sure how these modems would survive in an uncontrolled enviroment, such as where a di i eater or packet switch might be located (WB&JPI-5 operates in a metal shack part-way up a broadcast tower and the tempature inside the shack goes from about 115 degrees in the summer to below zero in the winter).

The NNC has much of what it takes to build a decent packet switch. The main limitation is still the RAM memory access. Since the CPU is based on a Z80, it can only access a total of 64k of memory at a time, both program and data. Even though the 512 kbytes of memory can be paged through in 64k windows, the amount of program space must be included in that 64k, again reducing the amount of data space directly accessible. If the program takes 32k to run, this leaves 32k total data space available without paging. When the program grows to 48k, the data window is then reduced to only 16k chunks.

This memory constriction may not be much of a problem for quite a while, since paging of data chunks can be accomplished without too much software magic. The SCSI bus allows for larger mass-storage devices than floppy-disks., such as hard-disks. The two timers will allow for timed interrupts, and the time-of-day clock is a nice addition.

Since the NNC is software compatible with the Z80, once again the whole CP/M software family is available for program developers.

The NNC is desfgned to use low-power devices, such as static RAM and SIOs instead of SCCs, the latter of which is becoming easier to use. Because of this, the NNC is a good candidate for very-low power operation, such as mountain-top switches.

I see the TAPR NNC as being a potential stepping stone on the path to the more complicated packet switch. Since it is basically software compatible with the Xerox 820 board (except at the lowest port address level), code developed for the 820 will be able to be run on the NNC. This will allow packet switches to be upgraded fairly easily when the primary reason for obsolescence is due to the lack of memory. It will also come in handy when the supply of surplus 820 boards dries up.

Packet Switch Design Based on IBM Clones

Over the last six months there has been a fairly quick conversion of the AMRAD crowd from CP/M systems (either 820 boards or S100 systems) to the IBM world. Yes, even I, the staunch S100 supporter has yielded to the tidal wave of big blue. In addition to my three S100 systems and O's, I have two IBM-PC compatible computers (one a real IBM, and the other a clone). In addition, I am trying to put together a tranp ortable PC clone, similar to my transportable 820 system (the size of a Kaypro, complete with TAPR 8530 mod and 7910 modem built-in). What has caused this wave on blueness? It's a simple matter of cost. The price of IBM-PC's and the PC clones has dropped to the point where one can build a complete computer for under $800.00, including monitor and drives. With all that software out there, how can you go wrong!

The idea of using a 16 bit CPU for the packet switch is not a new one, it's just that until recently the cost was too high. Now wait, I hear you saying that the PC is not a true 16 bit system. You are right. You are also correct in saying that even though the PC can address over a megabyte of memory, it still must do it in 64 kbyte pages. There are important differences between the PC and the normal eight-bitters.

First of all, even though the 8088 CPU looks to the outside world like an eight-bit CPU, it thinks inside with a full sixteen bits. This means instructions can be executed quicker. In addition, it's instruction set has been improved, allowing some functions to take fewer instructions to execute, thereby speeding up processor throughput.

Secondly, even though the 8088 can only work in a maximum of 64k banks, it can use different 64k banks for program and data. In fact there are four independant 64k banks that can be anywhere in the one megabyte address space. One of these segments is used for program operation, another is used for data storage, a third is used for the stack, and a fourth is provided as an extra segment. Since each of these segments is completely independant of the others, adding to the computer program does not reduce the amount of data memory directly accessable. This can be an important point when dealing with larger numbers of packet users.

In order to use the PC or PC-clone on packet, either a separate TNC type device must be used, or a HDLC capable board must be added to the PC system. We have designed a PC-compatible board that plugs into the bus that has an 8530 Serial Communications Controller (SCC), which has two programmable serial channels. Each of the two channels can be strapped for either RS-232 operation or can be used with a AMD 7910 World-Modem chip. This board (called the PC-PAD-l) fits in a short-slot position, so it is usable in almost any PC-compatible computer. The PC-PAD-l should be available shortly from TAPR.

A follow-on project to the PC-PAD-l board is also in the works. Due to limitations of the PC (lack of slots, interrupt lines, and DMA channels), and since the PC-PAD-l only occupies a short slot, I felt it would be a good idea to expand on the basic board. In order for a switch to perform added functions such as acting as a gateway between networks on different frequencies, it will probably need more than two HDLC channels. Therefore, the next generation board will have more SCC chips (at least three), it's own interrupt processor chip (slave to the PC's interrupt processor), and at least an additional 7910 modem. How much the board will end up having on it is based on how much we can stuff onto a standard PC card. I have already heard from a couple sources that a minimum of six separate HDLC channels would come in handy. The planned name for this board is the PC-PAD-2.

Ideally, it would be nice to have the HDLC ports interface to the PC through DMA channels, thereby reducing CPU overhead. Unfortunately, the PC only provides for a total-of four half-duplex DMA channels, and this cannot be expanded without cutting/kludging motherboard. In addition, one of these four DMA channels MUST be used for dynamic memory refreshing. Two of the remaining DMA channels are reserved for mass-stora e devices, if present. Since the switch should %ave some sort of mass-storage, at least one of these (and most likely both of them) will be used up. This leaves only one half-duplex DMA channel available on the standard PC, which is essentially worthless for packet work. In order to turn the one half-duplex channel around, and arbritrate which device is using it, more processing overhead would be required than to simply use interrupts in the first place. The bottom line is that DMA is not real useful for packet I/O processing on the PC.

Another limitation in the PC is in the area of interrupts. The motherboard has an interrupt processor capable of arbritrating up to eight interrupts. All of these are potentially reserved in the PC for certain devices as follows:

IRQ0	System board Timer 0.
IRQ1	Keyboard Scan Hardware.
IRQ2	Optional Clock Module.
IRQ3	Secondary Serial Channel (COM2).
IRQ4	Primary Serial Channel (COM1).
IRQ5	Optional Hard-Disk Controller.
IRQ6	Floppy-Disk Controller.
IRQ7	Printer Port.

Since they are all reserved, but we still need at least one, something has got to go. What I have done with our SCC board was to use the secondary serial port interrupt first, followed by the primary serial port. The only other interrupt that could be "obtained" without leading to conflicts would be the printer port line. I wouldn't want to take up either disk interrupts, and I also think a clock module is going to be an essential part of the packet switch.

Once an interrupt line is made available from the master interrupt controller on the PC motherboard, it is possible to add one or more slave interrupt controller chips to that line. In order to provide multiple HDLC channel devices (such as more than one SCC), the use of a slave interrupt controller will help to figure out which device caused an interrupt. Rather than polling each device, the interrupt controller is polled to see which device caused the interrupt. Since the slave interrupt controller will be on a plug-in card, as many of the interrupt sources as possible should also be on that card, which will reduce or eliminate the need for inter-card wires. The use of slave interrupt controllers is not the best way to implement intyerrupts, but in the PC enviroment it is the best available, in my opinion.

The power consumption of the PC-compatible computers varies with what is plugged into the bus, and what type of mass-storage device is being used. Most of the IBM computers were sold with 65 watt switching supplies, which provide enough power until the computer is full of RAM, has both floppies, several additional cards, and an internal hard-disk. This tells me that a typical packet switch computer based on the PC would probably draw about 50 watts on the high-end. While this is higher than some would like for a mountain-top switch, it is still fairly low power-consumption when one considers the computing power involved. The replacement of standard TTL devices with high-power CMOS, and the removal of local console devices (such as keyboard and video board) would help to reduce this load.

Packet Switch of the Future

Down the road, it would be nice to have a packet switch designed by the Amateur community. What follows is some of my thoughts/ideas/leanings on what a computer specifically designed as a packet switch might look like. Some of this is taking shape now, while other parts are just in the initial idea stage.

One of the most important concepts I want to push from the outset of this section is that eventually a single CPU will most likely become overloaded with work in a switch, especially a switch that covers a metropolitan area, or a switch that interfaces between several different subnetworks. The method I propose to overcome this situation is to break up the workload of the switch into several, smaller pieces. Some of theses pieces would then be taken care by smaller, slave processors to the main switch processor. This way the switch can handle its functions more effectively without being interrupted to do the more mundane work. An example of this whould be a slave board that would handle one or more HDLC channels. This board would be very much like a TNC, and would be in charge of maintaining the AX.25 level 2 connections. It would pass data to/from the main switch only after it had processed all the level 2 work necessary. This relieves the switch processor from having to do ANY Level 1 or Level 2 processing.

Central Processor Used

In order to take advantage of the development tools we presently have, I decided that my design for the next generation packet switch should be based on the Intel line of microprocessors. My feeling is that it should be a true sixteen-bit processor, not an eight-bit external one. While it would be nice to use either the 80286 or 80386, both the cost of theses chips, and the added complexity of the hardware involved detracts from their use. Because of this, at this point, I think the Intel 80186 CPU is the best compromise to be used in the packet switch. Most of the C compilers and assembler packages for the PC now include 80186 support, so the development enviroment is there.

General Switch Computer Architecture

There are as many different approaches to the design of a computer as there are designers. In my opinion, one of the foremost requirements is that the switch be designed to use a motherboard with a bus to plug special function boards into as needed. This allows the switch to change as needs change, without obsoleting the hardware. It also allows for easier troubleshooting, since function boards can be swapped-out until the defective one is found. As to which bus should be used, at this point I am leaning toward a modified version of the IBM PC AT bus. It is both eight and sixteen bits wide, allowing present PC boards to be used for the more mundane operations, such as video and disk controllers. The AT bus also has twice the interrupts and DMA channels available as the PC. The reason I say a modified PC AT bus is that the AT still does not allow for expansion of the DMA channels to the function cards. The switch should provide the necessary signals on the bus to allow DMA controllers on the plug-in boards in addition to what is used on the motherboard. Since the AT bus does not have these signals, some method of supporting them would have to be added. It appears to me that this is a small price to pay for all the support that using the AT bus would bring.

The switch motherboard should also have the hooks needed to add a "Service Processor" board. The Service Processor is kind of an advanced watchdog device that should continually monitor the main switch operation for abnormalities. Since it will need direct access to certain points on the motherboard that are not on the bus, some additional connections will need to be provided.

HDLC Channel Processor Plug-In Boards

There are actually two different types of HDLC channel boards I see being used in the packet switch. The first version would be similar to the PC-PAD-2 described above. The main difference would be that the board would use a dedicated DMA controller for data transfer to the switch rather than interrupts. This would greatly relieve the central processor from byte-by-byte interrupts from each HDLC controller. I would recommend that the HDLC chip used be the 8530 SCC chip. I feel it is a superior device, and performs well with the Intel line of chips.

The second and more advanced type of HDLC controller board would be a slave processor that would use an on-board CPU such as a Z80 to process the data of the HDLC channel(s) it supports, and use DMA to pass the data to the central processor of the switch. As mentioned above, this slave processor would actually handle all of the AX.25 Level Two connection overhead, further relieving the switch. This is needed as the HDLC channels go to faster and faster speeds. There comes a point where one processor will become bogged down, no matter how fast it is. This delegation of responsibilities to slave processors will become a necessity eventually, and if the switch is designed to accomodate it from the beginning, so much the better.

Both types of HDLC boards should have a watch-dog timer on every HDLC channel they provide. It would also be handy if the watch-dog timers had outputs that fed the switch processor, the HDLC Channel Processor, or a Service Processor, that indicated when they had timed-out.

Mass-Storage Devices Support

If the PC AT bus is used, standard IBM compatible mass-storage devices could be used, relieving the designer of both hardware, and more importantly software work. Both floppy-disk and hard-disk support should be available. One point is that at least the floppy-disk system should probably NOT use a high-density recording method. In order to make the floppy more reliable, lower density storage methods should be used, and if more storage is needed, either more drives should be added, or a hard-disk should be considered. The system should also be designed so that as much data is maintained in RAM memory as possible, with the mass-storage used primarily as a data back-up.

Memory Requirements

The memory requirements for a packet switch falls into three catagories; program memory, data memory, and control and variable storage memory. Each of these types of memory have different needs, so they will be discussed separately.

The program memory would be responsible for holding the program(s) that the switch executes. It may hold the actual executing code itself, or it may contain a copy of the code that is read out of the program memory into another part of the switch memory for actual execution. I anticipate what will eventually happen is that a small "bootstrap" version of the opperating program will be kept permanently in the program memory in case the switch needs to be completely re-booted. The actual operating software will then be loaded into the switch, either from a mass-storage device, or over the radio from another switch or switch control operator. Since this boot code must always be available, it should be stored either in EPROM (prefered) or battery-backed RAM. Once the real operating software is loaded into the computer and is executing, this EPROM code could then be phantomed out, reducing the amount of memory space occupied.

The data memory will occupy the most address space of the switch. The simple rule regarding data memory is: The more, the merrier. Since there will be a lot of data memory, it should be fairly inexpensive. I am leaning toward using dynamic memory for this part of the switch. These days, a megabyte of memory only costs about $120.00. If dynamic memory is implemented, I would also recommend that a dynamic memory controller chip be used, rather than the DMA refresh approach that IBM used. The switch will be busy enough already, without wasting additional overhead doing memory refresh. The cost of a dynamic memory controller is not much these days, so it would be money well spent.

The dynamic memory support should also have some method of detecting memory failures. One method is to add a simple parity check scheme similar to what the PC implements. Another method is to use one of the more advanced dynamic memory controllers that include failure detection. These more advanced controllers can not only detect failures, but "hide" the bad section f memory from the host processor, something which might come in real handy for remote, mountain-top switches.

Another idea associated with the main data memory is to have some spare amount of it available. If a memory failure is detected, this spare memory could be placed into the memory map of the host processor instead of the bad memory. The size of this spare memory could vary from a spare bank of chips on the motherboard to a completely separate, full amount of data memory with it's own memory controller on a add-on board, in stand-by. Obviously the latter is an extreme, but it would be advantageous where the switch is at a real remote location with very limited access.

At this point in time, the chips to use would seem to be the 256k by 1 devices. They are cheap, and have a very high density. a recommended dynamic RAM controller is the Intel 8208, since it works well with the 80186.

The third type of memory is used to contain the important variables and control information used by the switch, such as callsigns, link connection information, switch status flags, etc. This memory should be battery backed-up, and it should have a checksum which is updated whenever the data in the memory is changed, and is periodically checked. This memory is important enough that an added measure of safety might be required, that of maintaining a second copy which is also updated and checked. This way, if there is a failure in the primary memory module, the back-up could be switched into operation. Since this memory is battery backed-up, it should use a very low amount of power, such as a CMOS device. Fortunately, the amount of this memory needed is relatively small.

Service Processor Board

The Service Processor is an optional board that should have a separate connection to the motherboard. The function this board provides is to perform sanity checks on the packet switch. Exactly how interconnected the main switch motherboard and the Service Processor board are depends on how many sanity checks one feels are needed. At a minimum, the Service Processor should receive periodic "pings" from the switch processor via a common I/O port and also monitor data and/or address lines for a "stuck" processor. In addition, the Service Processor might also monitor each HDLC channel for 'stuck" transmitters, possibly indicating that HDLC channel is not being properly serviced.

Whenever the Service Processor detects an error condition, it should be capable of taking one of three actions; if the error is easily recoverable the Service Processor should attempt to correct the problem, if the Service Processor cannot easily correct the situation it should perform a hard reset to re-boot the switch, if the Service Processor determines that the problem is a major one, it should disable the switch from transmitting on any of its channels. In addition, the Service Processor should store the symptoms of the failure for later retrieval during servicing. Also, the Service Processor should have a distinct packet address, capable of sending and receiving connection requests. If the Service Processor detects a switch failure, it should notify neighbor switches or a service point.

In order for the Service Processor to attempt to "heal" the packet switch during minor problems, it should be able to control certain parts of the packet switch. It should be able to change or disable certain parts of memory if it thinks there is a memory problem. It should also be able to cause the packet switch to cold-boot if necessary.

An obvious potential problem can be if the Service Processor itself goes bad. In order to prevent a bad Service Processor from messing up the switch, some sort of hardware interlock should be provided on all interfaces to the switch. This might include a timed window during which the Service Processor can send commands to the switch. Any commands that fall outside this window would be ignored by the hardware.

This timed-window could be designed such that in order for the Service Processor to gain access to a critical area of the switch hardware for alterations, it must first send a byte having a certain bit pattern to one of it's own ports. Hardware at the output of that port compares the received byte to what it expects, and if they match, unlocks access to the requested switch area. This access is only allowed for a short period of time (using a hardware timer), and if the Service Processor fails to send it's command before the time ends, the door is closed, and the Service Processor's request is ignored.

Additional Slave Processor Boards

There might be the need for additional slave processor boards in the switch to perform functions such as User directories, and more importantly routing database upkeep. Once again, this might best be handled by having a separate slave processor, such as a Z80, with a lot of RAM and a canned database program. This way, when the switch needs to obtain the necessary routing information, it can ask the slave Route Processor for the route, then continue with other tasks while the Route Processor looks up the route. Another advantage is that the routing information is stored in RAM for faster access, but still won't take up valuable switch memory space.

Power Consumption of the Advanced Switch

Even a quick look at the above will indicate that our packet switch has become a power hog. In order to reduce the amount of power used, the switch should be designed using CMOS devices wherever possible. There should also be some way of shutting down selected portions of the switch when they are not needed (such as the local console). Even so, the switch will draw more power than some would prefer in certain situations. For this reason the programs written for this switch should allow for smaller versions of the switch, with the software testing to see if certain parts of the switch are present.

An example of this would be if a switch to be placed on a mountain top will only have a couple HDLC channels coming to it. Since the switch is powered by batteries (trickle-charged by solar cells), it is decided that the main processor can perform all its duties and also support the HDLC channel processing required. This means the HDLC Channel Processor will be removed, and a simple HDLC Controller card will be put in its place. Now, if the switch goes down and gets re-loaded from a neighbor switch, the newly loaded code must check to see what hardware configuration is being used before it starts bopping out packets. This is one example of how hardware "pruning" might affect software operation. Anytime the hardware is reduced, one should look for situations to develop in other areas. Sometimes it might be better just to increase the amount of batteries and solar cells and live with the added drain.

Advanced Switch Hardware Conclusions

While it is beyond the scope of this paper to set down the absolute design of the next generation of Packet Switch hardware (especially since we don't have much of a feel for what is actually needed yet), I think the above ideas will aid in the design of the future packet switch. One point I must emphasize is that the switch can be designed using the most sophisticated processor chip in the world, but if the development tools (both hardware and software) aren't there to support the device, no one will use it. This is the basic reason I decided to push the 80186 at this time. If other computers based on other CPU's such as the 68000 become a dime-a-dozen like the IBM PC and it's clones have, then I would be in favor of using those CPU's. For now, with the PC being a clear winner in the popularity contest, I am keeping to it's family of chips.

Another important point is that the switch be based on a bus structure. This will allow easier switch expansion and troubleshooting.

The third important point I want to leave the reader with is that even though right now it might be difficult to imagine how a single processor might get bogged down in a switch, that time will come all too soon. If the switch is designed from the beginning to off-load as much specialized processing as possible (ala HDLC Channel Processors and Route Processors), it will take a lot more to overload the switch with work. This will become a very important point in the next few years.

I hope to be working on the more advanced packet switch design over the next year. So far, my design is based on the 80186 CPU, a megabyte of dynamic RAM supported by an Intel 8208 dynamic memory controller, some EPROM, ten expansion slots, a timer chip, expandable DMA capability, and expandable interrupts. The system is NOT up and running at this time, but I hope it will be shortly. The first add-on board will be a DMA-supported HDLC board, with multiple SCC chips.

Conclusion

Developments in the area of digital hardware for Amateur Packet Radio are progressing smoothly. New designs are coming along, for the end-users, digipeaters, and the packet switches. Hopefully, in the next year the software and RF hardware parts of the hobby will catch up with the digital hardware strides made in packet radio.

References

Fox, T., "Packet Switch Software Design Considerations", Fifth ARRL Amateur Radio Computer Networking Conference, ARRL, 1986

Fox, T., "RF, RF, Where is My High Speed RF?", Fifth ARRL Amateur Radio Computer Networking Conference,, ARRL, 1986