PACKET RADIO DEVELOPMENTS OVER THE LAST YEAR

Terry Fox, WB4JFI
Vice-President, AMRAD
7877 Hampton Village Pass
Annandale, VA 22003

ABSTRACT

Over the last year, AMRAD has been continuing it's work on the development of packet switch hardware and software. Our progress has been slower than we had hoped, due to our interests in other projects, such as spread spectrum. The hardware portion of our packet switch has made better progress than the software. Over the next year we hope to make more progress in software development. What follows is a brief description of what I think have been some significant advances over the last year, along with some of my current thinking in packet network development.

Packet RF Development

Since last year there has been some movement in packet RF hardware. No longer is 1200 bps the fastest speed generally available. Kantronics has been shipping their 2400 bps packet boards (KPC-2400) for a while now. While this system has met with somewhat mixed reviews by the packet community, it does represent an advance in RF technology.

AEA announced at Dayton their new 220 mHz 9600 bps RF Modem/Radio. This unit is designed to be used either as a voice radio (with included microphone) or as a 9600 bps RF modem for packet work. One nice result of this is that voice operation can be used to align the RF path and test the units out, then switched to data mode for normal operation. As of this writing, I am not aware of any units actually shipped by AEA, but I guess they will be shipping soon.

Another high-speed packet rig has been announced by GLB. Their radio also operates on 220 mHz, but can go 19,200 bps, or twice the AEA speed. Initial shipments have been made to several beta test sites so far, when the results of these test sites come in, I would expect these radios to become available.

I should point out that both of the above mentioned radios will be relatively expensive initially. In addition, it should be pointed out that both units operate on 220 mHz. It seems just as the equipment is becoming available, the FCC is considering eliminating enough of the 220 mHz band to render these radios useless. The Amateur packet community should redouble our efforts to persuade the FCC that we NEED these frequencies for digital communications. If we lose a large portion of 220 mHz, I feel our next logical frequency for building a network will realistically be 900 mHz. Most of the population centers of the U.S. already have a large portion of our 420-450 mHz allocation used, with even more pressure being placed on it if some of 220 mHz is lost. The 900 mHz band can be considered microwave, requiring paths to be line-of-sight, making network development even more difficult.

Another effort is being made in Atlanta that is even more ambitious. An amateur of some repute in Atlanta has developed a board-set that (when assembled) will run at 56 kbps. This is designed to create the digitally-modulated RF at a 29 mHz IF, which is then transverted to the band to be used. Some of these boards are being sent out now, with testing to be done in the field. This is a potentially exciting development, providing it does operate cleanly and efficiently. One note on this is that there have been some problem getting a TNC-2 to operate reliably at 56 kbps. If this is true, the shoe is on the other foot, in that the RF hardware signalling speed has surpassed the digital hardware capability!

There is also some movement on HF packet operation. There is a new movement afoot by the ARRL and others to try higher speeds on HF using minimum-shift-keying (MSK) modulation. The ARRL has asked for an STA for testing these newer HF signalling techniques. This will allow higher-speed BBS interlinking over long distances.

It appears that there has been movement over the last year in the RF portion of packet radio. Hopefully this trend will continue, with other manufacturers also jumping into the field. My other hope is that the RF systems that do come into being become compatible with each other. We do not need incompatible RF radios trying to inter-operate.

Packet Digital Hardware

With the proliferation of PC clones in the last year, the those working on developing the Amateur Packet network have been migrating toward the IBM PC and clones. At this point, most of the work being done is on the PC. Two of the boards becoming available to support this work are the PC-100 from Pat-Comm and the Eagle RS-232 board.

PC-100 From Pac-Comm

The PC-100 from Pat-Comm is an 8530 packet board for the PC that I designed over a year ago. It has an 8530 Serial Communications Controller (SCC) on it, which drives either an RS-232 connector, or on-board modems. The board is a half-slot size, and plugs into the PC bus. It can provide two different channels of packet operation, and more than one board can be chained together, allowing a switch (or user for that matter) to have many RF inputs. We in AMRAD have had versions of this board running (sending and receiving frames) for over a year now, so we feel it will contribute a lot to packet development. The PC-100 should be available from Pat-Comm Real Soon Now (RSN).

The latest version of the PC-100 uses one 7910 world-chip modem, and optionally a TCM-3105 CMOS modem. The following is the port map of the standard PC-100:

200-203 hex	Modem Control Latch.
204 hex	8530 SCC Channel B Control Port.
205 hex	8530 SCC Channel B Data Port.
206 hex	8530 SCC Channel A Control Port.
207 hex	8530 SCC Channel A Data Port.
208-20B hex	Interrupt Acknowledge Strobe Port.

Bits 0-3 of the Modem control port set the various operating modes of the 7910 modem chip, while bits 4, 5, and 6 control the TCM-3105 modem chip.

One of the changes Pat-Commhas made is to use an external (to the SCC) oscillator for serial clock generation. This allows them to use the now familiar external divide-by-32 circuit from the still-born AMRAD PAD days, making full-duplex operation much better. Unfortunately, this also makes the PC-100 sometimes inoperable with PC clones in turbo mode.

The problem with turbo-mode clones is that the clock signal driving the clock input to the 8530 is not synchronous with the host CPU, and NOT FAST ENOUGH. The 8530 requires the PCLK (clock) input to be at least 90 percent of the CPU clock speed for proper operation. A PC running at

4.77 mHz does allow the 8530 time to cycle properly, but when running a clone at 7 mHz (for example), the 8530 no longer has enough time to function properly with the CPU. The standard PC-100 uses a 4.9152 mHz oscillator, with faster clocks and higher speed parts optionally available. This optional speed-up throws the problem in the software designers lap (which board, what baud-rate divider numbers to use? etc).

The PC-100 is interrupt-driven only. The original PC-100 used pin IRQ3 standard, with jumpers to optionally move the interrupt to other hardware vectors.

Eagle RS-232 Board

Another board has recently become available to the Amateur packet community. When Eagle computers folded, a company in California apparently bought up a large portion of Eagle's equipment. Eagle had designed a RS-232 serial board for their PC compatible computer that uses the 8530 SCC chip. Someone found out these were available, and the rush was on. This company quickly sold thier stock of functional boards to Amateurs hungry for development tools. Several of us in AMRAD were fortunate enough to obtain one or more of these boards, and as far as I am aware, we were the first to get the board to successfully send and receive packets on the air.

The Eagle board is somewhat similar to an RS-232 version of my 8530 board. It provides two channels of serial data flow, with both being RS-232. The SCC chip can operate in either interrupt mode, or can use DMA with the PC (unlike the PC-100, which is interrupt-driven only). Since there are no modems, one must be provided externally. There are two "gotchas" with the Eagle board. Fist of all, the interrupt channel used is the same one normally used by the hard-disk controller. Secondly, TxD and RxD on the DB-25 for one of the channels are backwards from "normal" (if there is a normal in RS-232). Aside from these two glitches the Eagle board does indeed operate on packet with an IBM PC. There has been enough interest in these Eagle boards that the company that purchased the rights to them is considering making more of these boards.

The port mapping for an unmodified Eagle RS-232 board is as follows:

300 hex 8530 SCC Channel B Control Port.
301 hex 8530 SCC Channel B Data Port.
302 hex 8530 SCC Channel A Control Port.
303 hex 8530 SCC Channel A Data Port.
304 hex Board Control Latch Port.

The Control Latch port is broken down as follows:

Bit 0 DMA ONLY...SCC *CTL/DATA, 1 = Data.
Bit 1 DMA ONLY...SCC Channel, 1 = Ch. B.
Bit 2 DMA Operation Gate, 0 = Non-DMA.
Bit 3 Interrupt Enable, 1 = Int Eanbled.
Bit 4 INTACK pin Strobe.

The Eagle board also uses an external oscillator for baud-rate generation. However? it does NOT have the divide-by-32 circuit required for proper full-duplex operation. The standard frequency used on the Eagle board is 3.6864 mHz. Since most of the components on the Eagle board are NOT socketed, speeding itupmay be much more difficult, making this board harder to use with a clone in turbo-mode.

It would be nice to use the DMA capability of the Eagle board with the PC, but that probably won't be done much for the same reason I didn't design DMA capability onto the PC-100. The PC bus only allows for a total of four half-duplex DMA channels. Of these, one is tied up doing dynamic RAM refresh, and one each is assigned to the hard and floppy disk controllers. This only leaves a single half-duplex DMA channel for us to use (without time-sharing the others, which could be a software nightmare). Both my board and the Eagle board support two channels of operation, both of which are potentially full-duplex. In order to support these boards properly, four DMA channels per board are required. Since we only have one-fourth of that available, I decided not to use DMA at all. Trying to decide which channel would have the DMA versus the others would be too complicated with barely minimal improvement in operation. In addition, the PC bus has left out at least one crucial signal, which prevents placing DMA controllers anywhere except the motherboard (at least without serious kludging).

At this point, the rest of the hardware being used for switch development is based on the PC clones that have taken over the hobby computer market. We are working with standard PC Clone motherboards, Clone power supplies, clone floppy disk controllers with either 3.5 inch or 5.25 inch drives, and a small sized hard disk. The floppies are planned to be used for loading new code or data, or for local diagnostic and maintenance functions, while the hard-disk will be used for on-line data backup, with the actual data stored in memory. At first the switch software will be left in RAM memory, but plans are that once it stabilizes, the code will be loaded into EPROM, along with a ROM sanity monitor and boot loader. This boot loader will be capable of loading new code into RAM either from the disk drives or through a simple AX.25 LINK LEVEL ONLY connection.

The PC packet hardware being worked with so far has been both the PC-100 and the Eagle board, with both operating using the same basic software. As the switch complexity develops, more attention will have to be paid to the individual packet channels. I am guessing this will lead to a new board that will plug into the PC bus and use a DMA channel to communicate with the PC. On this board will be a complete computer system supporting the packet channels up to the Link Layer, and then passing the packets to the host PC for higher-layer processing as necessary.

Other 16-Bit Digital Hardware

Rumor has it that a group of Amateurs in Southern California have developed a new Network Node Controller (NNC) (read switch) based on sixteen-bit technology. They are using the Intel 80186 CPU with all static RAM. The packet channels will be supported with our good old 8530 SCC, of which there are at least two, and I think there will be up to three. Each 8530 will be using DMA for all four operations (Tx and Rx for both channels), which should allow for higher-speed operation. Needless to say these boards are eagerly awaited. The last I heard is that they are nearing beta-test shipment.

Eight-Bit Networking

The last vear has seen interest in the eight-bit world almost completely dry up. With the cost of PC clones coming down, virtually everyone working on network development has abandoned the eight-bit systems (some out of necessity, you can only fill a five-pound bag so full). I am afraid the Xerox 820 and the TAPR NNC are two casulties of this sudden swing. I don't think TAPR is going to be doing much about the NNC in the future, and few people are planning to work on the 820 any longer, except as individual user stations or packet BBS systems.

The one area where this is not true is surprisingly the TNC-2, and it's various clones and semi-clones. When it comes to these devices, there are two opposite trends happening over the last year. The first is to provide more sophisticated software inside the TNC-2, both for the end-user and potentially as a true packet switch. Witness the use of the TNC-2 with NET-ROM which turns the TNC-2 into a single channel packet switch. In order to operate a dual-frequency packet switch, two TNC-2 s are connected back-to-back. Also, once a TNC-2 is used with NET-ROM, it can no longer be used as a TNC, there is no support for both operations.

In the same vain, the virtual-circuit networking crowd is still planning to provide ALL end-user networking within the TNC-2. This way the Amateur packeteer does not need an IBM PC or clone just to gain access to the network. In addition, the virtual-circuit crowd is still planning to base a simple VC switch on the TNC-2 for regions that do not require the horsepower of a PC clone, similar to what Howie Goldstein, N2WX showed at the Conference last year.

At the opposite extreme we have the TCP/IP crowd. They are taking the TNC-2 and gutting out all of the AX.25 and user interface, leaving behind what they call the "KISS TNC". The reasoning behind this is that their protocol software barely runs on a PC, it does not run on a TNC-2. They are running packets using unconnected Link Level frames (using only UI frames) with the datagram information packed into the Information field of the UI frame. They rely on the Transport Layer (in this case TCP) to handle ALL error recovery. This "Keep-It-Simple-Stupid" operation allows the PC software to handle all aspects of packeting, except actually generating the envelope around the datagram, which is left to the TNC, along with gaining access to the RF channel. The datagrammies feel this provides much more flexibility, since they can now run unconnected datagram service at both the link AND network layers. This author does not necessarily agree with them, and somehow the picture of eight ganged-together digipeaters from Washington DC to Boston (with 5 minutes and 3 disconnects/reconnects just to say hi) starts painfully coming into focus from the recess of my mind, but that's life. A square wheel is better than no wheel, I guess.

Packet Switch Software Development

While others have been forging ahead designing and writing packet switches, AMRAD has been taking a somewhat slower approach to this task. I personally feel **that** for the packet switch to work efficiently many software design issues must be decided BEFORE the first line of code is written. I have been hearing a lot about writing ALL code in C versus Pascal versus Assembler versus whatever languageof-the-month comes up. My opinion is that there is no ONE SINGLE answer to the language issue. There have been two instances where a software designer/programmer has written virtually all software in a higher-level language (including almost all the interrupt processing). When the software didn't run fast enough, this programmer claimed the hardware was not capable of running any faster, and the only answer was DMA operation. In fact, the hardware could run MUCH faster if the interrupt processing had been done in assembler instead. The point of this is that the software should be optimized for the task. Taking 2 years to write a full switch in assembler can be just as bad as the above mentioned example. Software language optimization for each specific task should be done before proceeding with that task. In addition, some attention should be paid on standardizing how tasks are supposed to intercommunicate, and how shared resources are to be requested and allocated.

Last year I wrote a paper on how I felt the various functions of the packet switch should be broken down. Since that paper was published, I have felt that most of the comments made there are still valid, in fact I feel even stonger that the support services should be standardized within the switch. We in AMRAD have been passing around a lot of ideas regarding switch softtware design, and while not a lot of code has been written so far, I feel much better about the direction we are taking than if we had just bolted off into the ether writing madly away.

Some of the modules I have identified over the last year are as follows:

Shared Resource Modules

SOS	Switch Operating System.
BQM	Buffer and Data Queue Manager.
CTM	Clock and Timer Module.
ECR	Error Control and Recovery.
DBI	DOS and BIOS Interface module.

Internal Switch Modules

LCI	Local Console Interface.
SCC	Switch Command and Control.
AUT	Authorization Module.
UDB	User Database Manager.
RDB	Routing Database Manager.
RTB	Round Table Operations Manager.
PAD	Packet Assembler/Disassembler

Protocol Machines

SSM	Switch Session Module (Layer 5).
ETM	End-Point Transport Module(Layer 4)
NET	Switch Network Module (Layer 3).
LINK	Switch Link Module (Layer 2).
LARM	Link Address Resolution Module.
PHS	Physical 'Hardware Support Module.

Each of these modules has their own distinct function, with attendant trade-offs in usage/size/speed/complexity. Therefore, I feel each module should be studied fairly independantly before deciding how it should be written, and in what language.

Switch Operating System (SOS)

Since the Switch Operating System is responsible for almost all inter-task communication, it will be used a lot. Because of this, it should be lean and mean and not cause much additional delay 'between processes. I feel in the long run, the SOS should be written in assembly language for each target processor.

The operating system we have decided to use is a fairly simple one, often called the HUB operating system. Mike O'Dell wrote a paper for last years conference which describes it in more detail. He also wrote a version of the HUB Operating System in C, which we have had running both on Xerox 820's and IBM PC's. Despite what I said above regarding assembly language, since he already has it working in C, I think we will use **his** version, at least until one of **us** has **time** to re-code it in assembler.

Buffer and Queue Module

The Buffer and data Queue Module is responsible for handling both the data buffers and the data buffer queues. Most of the actual switch processing on received/transmitted data will be done inside these data buffers. In order to provide smooth data buffer flow between processes, data queues will be implemented.

Once again, since the Buffer and Queue Manager is controlling a valuable shared resource, this module should be optimized for speed, implying assembly language. Getting memory for a data buffer from a C language heap each time you need a buffer can take up a lot of extra valuable time.

I have implemented the Buffer and Queue Manager for the IBM PC in assembly language, and this code has been checked out and de bugged. Incidently, it has also been checked out and works properly when called from C routines.

Clock and Timer Module

The Clock and Timer Module is responsible both for maintaining the time-of-day functions and the timer requests from other modules, such as the protocol machines. Once again, it is a shared resource, and should probably be written in assember, to reduce routine latency.

We have decided both the time-of-day and the timer routines should keep a linked list of timer requests, with each request going in order of time. This means only the shortest timer actually needs to be counted down, and once it expires, the requesting routine gets notified, and the next shortest timer becomes the active request. This method is much more efficient than trying to update many timers at each "tick" of the system clock. We are just starting work on **this** module as of this writing.

Error Control and Recovery

The Error Control and Recovery Module will be responsible for either trying to "heal" minor errors, or disable the switch if major malfunctions are encountered. It will also be responsible for accumulating totals for "soft errors" that may occur during normal switch operation, for later software tweaking. This module is not really necessary at first, and can be written in a higher level language such as C.

DOS and BIOS interface

This module will be used to gain access to the various hardware available in the PC, primarily the disk drives. It is not crucial for switch operation at the *beginning, and will probably be written in a higher level language, such as C.

Local Console Interface

The Local Console Interface will be responsible for allowing local access to the switch either for "tweaking" or for maintenance. Some form of this module will most likely be required when the switch is first activated, but it will probably be expanded upon as the need arises. Once again, it could be written in a higher level language.

Switch Command and Control

The Switch Command and Control module will be responsible for interpreting and acting upon the commands given to it either by the Local Console Interface, or by a packet connection to the switch via the Authentication Module. Obviously, some form of this module must be operational to alter operation of the switch. It has not been written, and will most likely be written in a higher level language.

Authentication Module

As described by Hal Feinstein at last year's conference, some sort of message authentication must be provided if the switch internals are to be accessed via packet. Hal has been working on this aspect of the switch over the last year, with most of his effort being written in C.

User Database Module

The User Database will become increasingly important as more users show up on the network. This module has received almost no attention at this point. It is not needed for basic switch operation, but will be added at a future date. It could be written in a higher level language.

Routing Database

The Routing Database is used by the Transport and Network protocol machines to get the proper information on network connection routing at call setup time. This database is verp important for automated network routing. envision the actual database being held in RAM memory, with periodically disk compares/saves for backup purposes. This software has not been written yet, and depends on routing and naming conventions not yet decided upon. It could be written in a higher level language, since connection-oriented networks only need to access it at call setup time, not on every packet like most datagram networks.

Roundtable Module

One of the neat functions Howie N2WX provided in his second network level software was something that has been sorely lacking in Amateur Packet Radio, the roundtable. It allowed for multiple stations to connect to the switch, and all of them talk to each other, ala a voice roundtable. I have felt for a long time this was needed and was glad Howie showed it could be done. This module has not been worked on at this time, but will probably be written in a higher level language.

PacketAssembler/Disassembler

One of our firm committments is to make our switch downward compatible. In order to do this, we must provide some method for older, non-network capable TNC's to gain access to network functions. Once again, Howie showed how this could be done in his first release of network code for the TNC-2, as described in a paper presented at last year's conference. This is another task easily written in a higher level language.

Switch Session Module

The Switch Session Module will be responsible for handling the various internally terminated network connections. These include connections to the Switch Command and Control Module, the Roundtable Module, and the PAD module. It could be written in a higher level language.

End-Point Transport Module

The End-Point Transport Module will be responsible for assuring the user data made it through the packet network without errors or problems (if that was requested). It is based on X.224 Version TP-1, and has been described in other papers at the last two conferences. I feel it could be written in a higher level language.

Network Protocol Machine

The Network protocol machine will use the X.25/X.75 Network Layer protocols. Some background work has been done on writing this module, but no code has acutally been written. I feel the Network layer could be written in either assembler or a higher level language such as C.

Link Layer Protocol Machine

The Link Layer Protocol machine has been started. It is based on the AX.25 Level 2 protocol, and is being written in assembler. I feel the Link Layer protocol machine has enough time-crucial elements that to implement it in a higher level language would add too much delay, especially since we are starting to see higher speed packet operation.

Link Address Resolution Module

I have wedged a module in between the hardware support and the Link layer protocol machine, which I call the Link Address Resolution Module. It will take our Amateur callsign address kludge out of the received packets before passing them to the link module, and add the kludge to all transmitted packets from the link module. In addition, it will handle the digipeating function instead of the Link module. This separation will allow the Link Layer protocol machine to be more of a generic X.25 machine, reducing it's complexity.

This module is being written in assembly language, and is almost finished. This will allow testing of the switch shared resources modules as a simple digipeater until the Link, Network, etc modules are finished.

Physical Hardware Support Module(s)

The Physical hardware used to send and receive packets must have some software to support it. This is the job of the Physical Hardware Support Module(s). As implied, there can be more than one of these modules. In fact, there will be one of these modules for each hardware board in the switch. So far, both of the boards being used in our switch development use similar hardware (the 8530 SCC), so the PHS modules for these boards are also very similar. As stated earlier in this paper, both the boards have been on the air sending and receiving packets. All that is necessary is to modify the software drivers slightly to interface properly with the shared resources modules. The PHS modules are written in assembly language for the PC, using interrupts.

As can be seen from both the above description, and my paper from last year, our effort at developing a packet switch is probably more detailed and studied than most other implementations. I feel this will aid both ourselves in the development process, and others who follow after us in understanding what it takes to implement a complex function such as a packet switch.

Off On a Tangent

1 thought I might bring up something that has nothing to do with any of the above, but I feel should be mentioned somewhere (since I don't have

time to write another paper on it, here we go). I have noticed that a large portion of the packet activity revolves around packet bulletin boards. I generally agree with a good part of this, but I would like to point out something that might save quite a bit of channel activity in the long run. If more Amateurs were to put up their own bulletin board systems rather than using a central, community BBS, then other Amateurs could send mail directly to them, rather than a BBS they happen to frequent. This will lead to a reduction of overall packet activity, since the message can be automatically forwarded directly to the destination Amateur, reducing the chances of being read multiple times, and showing up on every message summary request. Something to think about for now.

Conclusion

There has been some movement over the past year in packet hardware, both digital and RF. Admittely, after the initial surge last year by the Virtual-Circuit crowd, the Datagrammies have caught up. I hope to have more time over the next year to finish our connection-oriented packet switch project, and start placing them into service. I still hope for an experiment where both approaches are placed side-by-side for a period of time and compared. I still feel that the Virtual-Circuit approach will be the long-term winner by a wide margin.

References

Fox, T., "RF, RF, Where is My High Speed RF?", 5thARRL Computer Networking Conference, ARRL, 1986

Fox, T., "Packet Switch Software Design Considerations", 5th ARRL Computer Networking Conference, ARRL, 1986

Fox, T., "User and! Switch Packet Digital Hardware", 5th ARRL Computer Networking Conference, ARRL, 1986

O'Dell, M., "An Introduction to the HUB Operating System", 5th ARRL Computer Networking Conference, ARRL, 1986

Feinstein, H., "Authentication of the Packet Radio Switch Control Link", 5th ARRL Computer Networking Conference, ARRL, 1986

Goldstein, H., "A Packet Assembler/Disassembler for Amateur Packet Radio Networking", 5th ARRL Computer Networking Conference, ARRL, 1986

Beattie, G., "Proposal: Recommendation AX.224, Class 1 Transport Protocol Specification for the Amateur Radio Network"., 5th ARRL, Computer Networking Conference„ ARRL, 1986

Fox, T., "CCITT X.224 Transport Layer Protocol basic Description", 4th ARRI, Computer Networking Conference„ ARRL, 1985

Rinaldo, P., "Introducing the Packet Adaptive Modem (PAM)", 2nd ARRL Computer Networking Conference, ARRL, 1984

Fox, T., "A New Packet-Radio-Controller board", 2nd ARRL Computer Networking Conference, ARRL, 1984