TAPR #94 WINTER 2005

PACKET STATUS REGISTER

President's Corner	1
Ten-Tec to Offer TAPR Vector Network Analyzer	3
DSPx for Embedded Applications Update	4
Commercial HSMM	5
D-STAR D-Licious?	7
Confronting AX.25 V2.2	8
PIC-et Radio IV: How to Send PSK31 Data Using Inexpensive PICs	10
Eliminating Source Routing from APRS	16
Inexpensive GPS25 Offered	18
TAPR Order Form	20

President's Corner

A New Partnership

By John Ackermann, N8UR, n8ur@tapr.org

Elsewhere in this issue of PSR you'll read about the agreement just reached between TAPR and Ten-Tec for manufacturing and marketing the TAPR Vector Network Analyzer (VNA) designed by Tom McDermott, N5EG, and Karl Ireland. I've written about the VNA before and the press release describes its capabilities, so I won't repeat all that here.

Instead, I'd like to talk about the TAPR/Ten-Tec partnership and why it's a great thing for TAPR.

Many TAPR products are fairly simple to build and have modest market potential. There are plenty of TAPR kits that have sold less than 100 units, yet we count them as successes because they filled a need for both the designer and a segment of our community. A few projects, like the Pic-E, are both simple and very popular, while others, like the DSP-10, are much more complex (but still within the reach of the homebrewer) and have a more limited market because they are quite specialized.

Every now and then, TAPR does a project that's both complex and has a lot of market potential. The TNC-1 and TNC-2 were our first examples of this, and our greatest successes. The FHSS radio, had things worked out differently, would have been another. The VNA is a third.

The VNA is both technologically innovative and very, very useful. It has the potential of reaching far beyond TAPR's usual base of experimenters and into the shacks of HF operators; QPRers, contesters, DXers, and RF experimenters of all stripes will find the VNA an indispensable piece of test equipment.

And, while 90 percent of its assembly is within the reach of many of us, it does use a handful of surface mount ICs that have many tiny leads spaced very closely together. To successfully solder them requires access to a microscope and other specialized tools. It's just not practical to expect most of us to build the VNA up from a bare board. The economics

of circuit board manufacturing mean that partial assembly (e.g., putting the difficult parts on, but leaving the rest as a kit) doesn't make sense because it costs virtually as much to have six parts mounted as one hundred.

So, we were left with a great product that (a) reflects the innovation that is a TAPR hallmark; (b) has great market appeal, including to users beyond our usual ranks, and (c) is too complex to handle as a kit. What to do? Look for a partner.

The TAPR gang has gotten to know the folks at Ten-Tec quite well over the last couple of years and we decided to see if they would be willing to manufacture the VNA for us. An initial conversation showed that beyond acting as a contract manufacturer, Ten-Tec was also interested in marketing the VNA; as a leading manufacturer of high-performance HF gear, many of their customers would have an interest in a piece of gear as versatile and accurate as the VNA. Discussions started last fall and resulted in the agreement that we signed in December.

Under that agreement, Ten-Tec obtains a license to manufacture and sell the VNA. In exchange, TAPR gets some financial benefits (the details are confidential, but our development costs are covered and we will generate a long-term income stream based on the VNA's commercial success). More importantly, the VNA will carry the TAPR logo as well as Ten-Tec's and this will be an opportunity for us to reach a whole new audience.

Tom McDermott has made the software for the VNA available under a free software license and under this agreement, Tom's software will remain open source and available for experimenters to play with. Ten-Tec will have the right to do their own proprietary development if they choose, but the current software will remain available and TAPR will support further open development through our mailing lists and other services.

I can't end this report without expressing TAPR's great thanks to Tom for creating a groundbreaking design and turning it over to TAPR to run with. This isn't the first time that Tom has contributed an important piece of work to the community. He truly exemplifies the ham spirit, and we thank him.

Other VNA News

Since I'm talking about the VNA, there are a couple of further bits of information to pass along.

Several folks have asked about buying a bare PC board. While those parts with lots of tiny leads make this project too difficult for most of us, bare boards will be available through TAPR in single unit quantities for those who want to tackle it. Pricing and timelines have yet to be finalized, but we'll make an announcement when the boards are available.

Next, Tom has made a major improvement in the VNA design since the beta versions were assembled. He's figured out how to get about 30 dB more dynamic range out of the unit for gain/loss measurements and the Ten-Tec version will include that improvement (there's also a retrofit available for the beta boards). That means that the VNA will be able to measure filters with 70 dB or more of attenuation; that kind of capability is normally seen only in lab-grade equipment, so Tom's done a great job.

TAPR at Hamvention

We'll be at Hamvention in full force again this year with our usual booth location, Friday morning Digital Forum, and Friday evening BASH. We hope to see you there!

Make Your Plans for DCC 2005!

The 2005 ARRL/TAPR Digital Communications Conference will be held in Santa Ana, California, from September 23-25. It's not too early to mark your calendar and make your plans for this year's installment of one of the best technical conferences in Amateur Radio! Learn all the details at **http://www.tapr.org/DCC**.

###

Ten-Tec to Offer TAPR Vector Network Analyzer

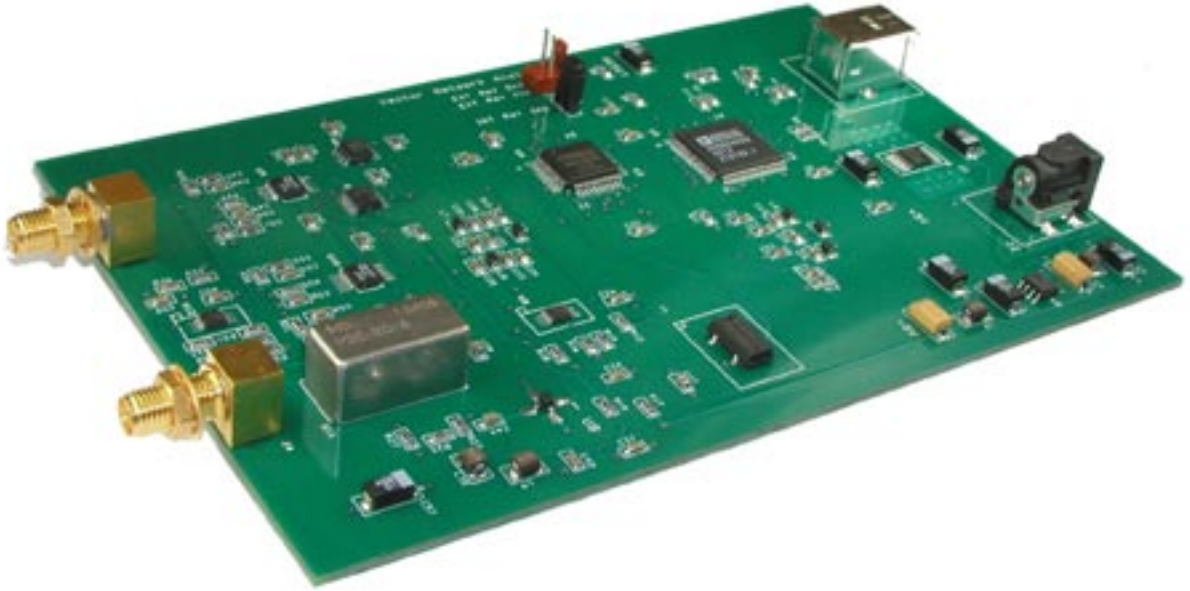
Ten-Tec and TAPR are pleased to announce an exclusive agreement to manufacture the PC-hosted 100 MHz Vector Network Analyzer (VNA) designed by TAPR member Tom McDermott, N5EG, as described in the July/August 2004 issue of QEX. The VNA is a tool for analyzing passive networks, including antenna systems.

The VNA allows measuring the forward and reverse gain and phase response of a circuit, and the input and output reflection properties (complex impedance). VNAs are used to measure and adjust filters, coaxial cables, amplifiers, antenna input impedance vs. frequency, just to name a few uses.

A unique feature of the PC-hosted VNA is that it is controlled completely from a PC via USB interface. Software for Windows(tm) 98SE or later Windows OS is included with the VNA. An open-source version of the software is also available from TAPR and TAPR will host mailing lists and other resources for users and developers.

The illustration below is for informative purposes; finished unit will be sold complete with enclosure. Production of the Ten-Tec/TAPR VNA is scheduled for late spring 2005. Price is estimated at $650. For further information contact **sales@tentec.com**

###



Ten-Tec TAPR Vector Network Analyzer (VNA)

DSPx for Embedded Applications Update

By Lyle Johnson, KK7P, kk7p@wavecable.com

In May, 2003, the KK7P DSPx signal processing module was released at the TAPR booth at the Dayton Hamvention. This module was designed for experimentation and self-education in digital signal processing (DSP).

The KDSP-10 kit was introduced to support the DSP-10 Software Defined Radio designed by W7PUA and kitted and sold by TAPR. In addition to its duties providing a suitable interface between the DSPx and the DSP-10, the KDSP-10 serves as a development platform for the DSPx.

Sound like old news? Two things have changed in January 2005 regarding these TAPR products.

DSPx

The DSPx includes flash memory. The earliest units had a 64k byte memory, later units come with 512k bytes of memory. However, there was no support for allowing the DSP to self-program its flash or for it to run any other application at power-up unless the flash chip was removed and re-programmed. All that has changed.

Available for free download on the DSPx web site **(http://www.kk7p.com/dsp.html)** is a program called FLASHUTL. This program allows you to selectively erase and program the flash memory on the DSPx.

Also available on the same web site is a new *Monitor*, which has an additional command to boot from flash, as well as a provision to recognize a jumper on the DSPx. If the jumper is in place at reset, the DSPx will run an application program rather than the *Monitor*. In addition, a new command, $BF, allows you to jump to the application program in flash from the *Monitor* prompt.

Numerous applications have been programmed into flash and booted just to test this. This includes almost all software from *Experimental Methods in RF Design*, including the 18-MHz transceiver. Complete directions for loading and using *FLASHUTL* to reprogram the DSPx with the new *Monitor* are on the web site.

KDSP-10

A new version of the KDSP-10 kit has just been released by TAPR. This is compatible with the existing one in every way, but has two additional features.

The first is an on-board 5V buffer with high drive capability. This buffers four output signals used by the DSP-10. Until now, KDSP-10 owners have often had to patch in an inverter section or two to get sufficient drive to reliably command the shift registers on the DSP-10 board for programming the synthesizers, etc. That is no longer required.

In addition, the board supports (but does not include) a 6-bit DAC. This is used by the G3XJP "Pic A Star" transceiver project.

Where is the DSPx being used?

There are lots of applications for embedded DSP in Amateur Radio. Hundreds of DSPx boards have been sold to amateurs around the world. G3PJT built the EMRFD 18-MHz transceiver and ported the DSP code to the DSPx.

A pair of hams in Perth, Australia, have created a 160m radio using the DSPx and code based on the 18 MHz transceiver code ported by G3PJT.

A ham in Italy has built a Pic A Star using code ported to the DSPx.

Elecraft **(http://www.elecraft.com)** is using the DSPx in their KDSP2 option for the K2 transceiver.

The American QRP Club **(http://www.amqrp.org)** has incorporated the DSPx into their antenna analyzer project.

Where will you use yours?

###

Commercial HSMM

By Don Rotolo, N2IRZ, n2irz@arrl.net

Not long after 802.11 gear became popular, we heard about High Speed Multimedia (HSMM) systems. The idea is that radio amateurs could use commercial 802.11b/g gear under Part 97 instead of Part 15 of the FCC rules, building an 11 Mb/s (or faster) data network on 2.4 GHz.

The advantages of Part 97 include a somewhat higher power limit, allowing for networks with reasonably spaced network facilities on the order of a few miles, instead of hundreds of feet. Hams can also use antennas different from those approved by the FCC for Part 15 use. Part 97 operation also adds some complexity aside from ID-ing and security - you need to find higher power gear, and get some better antennas - not always an easy or cheap option.

The reality is that no major HSMM networks have been built, to my knowledge, despite the ready availability of good, cheap equipment. (If you know of one, please write!). In this short article, we'll look at why that is, and examine a commercial implementation of what is essentially a HSMM network.

Line of Sight

The greatest hindrance to widespread deployment of an amateur HSMM network is the propagation characteristics of 2.4 GHz radio waves. Any RF link of reasonable distance must be optical line of sight, with hardly any RF-attenuating materials between antennas. A thin layer of brick or wood siding hurts a little, but a thousand feet of leaves - even if they total only a thousandth of the path length in thickness - has a devastating effect upon 2.4 GHz path loss. For most hams, at least in the major population centers of the east and west coasts, tall trees reduce the viability of HSMM networks to almost zero.

Even with an antenna above the trees, feedline losses at 2.4 GHz are large, so equipment generally needs to be located near the antenna. Such equipment is not terribly expensive, but it's more costly than the consumer-grade gear so readily available.

Aside from line of sight (LOS) issues, we have HTS. Just like AX.25 Packet, 802.11 does not tolerate the Hidden Transmitter Syndrome well. HTS is when two or more stations are trying to communicate with a central station. The central station can hear each remote station, but the remote stations cannot hear each other. The stations often transmit at the same time - 'doubling' - and both transmissions are damaged at the central receiver. The result is very poor upstream throughput.

Another significant issue is timing inherent to 802.11. It simply was not designed for long-haul links, so the few microseconds of light-speed delay from a 10-mile link becomes significant, again affecting throughput. The 802.16 (WiMax) protocol addresses the timing issue well.

There are more reasons why we don't have a decent HSMM network, but these are the big ones in my view. Not that these cannot be solved, but it does add a level of difficulty to a widespread deployment.

Now let's have a look at how one clever company got around these issues.

Sky Pilot Network, Inc (**http://www.skypilot.com**) has taken small piles of commercial off-the-shelf (COTS) 802.11a gear (5.8 GHz), added some controlling circuitry and antennas, and developed an easy-to-deploy high-speed network. The main disadvantage is the cost - a well-equipped network for a dozen widely-spaced users (or clusters of users) might cost $10,000. The advantage is nearly drop to deploy technology - simply feed the network element some power and it's ready with minimal configuration necessary.

The central element is a "Sky Gateway", about $2500. This is the central controlling element for

the network, and the link to the Internet (think commercial applications). You only need one, but it's not optional. Around that, you deploy any number of "Sky Extender" devices ($500), which are essentially intelligent digipeaters. Last, for each user (or cluster of users connected through some other means - 802.11b/g, Ethernet, packet...) you need a "Sky Connector" ($350), which is essentially a patch antenna that is pointed at either an Extender or the Gateway.

These prices might seem expensive for ham gear - they are - but compared to my cable connection at $45/month, I'd break even in well under 2 years, even contributing heavily to the cost of a Sky Extender and a bit for the Gateway. Note that the Gateway can support more than a thousand Extenders/Connectors.

Although 802.11a supports up to 54 Mb/s raw data rate, the Sky Pilot system claims to offer about 3 Mb/s actual throughput on a sustained basis, in both directions. That's a bit better than my cable modem. Range for true LOS is up to 20 miles, while Near LOS range is about 4 miles. These long rages are due to three factors: high-power transmitters, high-gain antennas (152.5 dBm link budget) combined with a type of Demand Assigned Multiple Access (DAMA) protocol to ensure no HTS exists.

The antennas are the key. There are about 16 high-gain, highly directions antennas arranged in a circle. The extender or gateway can transmit or receive on more than one antenna at a time. Using directional antennas allows relatively high power (about a watt, not entirely sure) under Part 15. The directionality allows a few transmitters (but not 16) to be in use at the same time without mutual interference. It also makes a DAMA scheme easier, since you have to switch to the right antenna for a given direction anyway.

The network software is fairly advanced. Although you can control and monitor everything - remember this is intended as a commercial system to compete economically with Cable and DSL - the network is self configuring (if you want it to be), routes like FlexNet (measuring paths to find the best), and has loads of features for access control, network topology, and so on.

While expensive, the Sky Pilot system could conceivably be deployed as an amateur HSMM network. The point of this article, however, is to bring attention to the fact that all of these technologies already exist in the Amateur Radio world, and perhaps those wanting to deploy a very fast radio data network can look to the Sky Pilot system for some good ideas on putting these technologies together into a workable whole.

I would imagine that even the development of a compatible replacement for the Gateway to be used by amateurs would bring the cost of such a network well within the reach of even smaller clubs and organizations. Even some of the ideas - like multiple antennas for point-to-point omni-directional coverage (not an oxymoron) - might work well for HSMM deployments, especially under Part 15.

HSMM networks using standard 802.11 gear sure sounds like a good idea, but there are some problems. Even though these problems are not new and have already been solved in the amateur world, 802.11 gear is less flexible than, say, a TNC. The result is that I have yet to find a large HSMM network. A small start-up in California has gone a step further and created the elements of such a network, designed to rival cable, DSL and WiMax deployments. Amateurs with a lot of cash can simply buy an HSMM network or use the ideas to build something similar. If only I didn't have 75-foot trees (and a 50-foot tower - sigh), I'd be the first one in the neighborhood to put up a 54 Mb/s user port on 13 cm.

###

D-STAR D-licious?

By Ed Woodrick, WA4YIH, ewoodrick@ed-com.com

The waiting begins

At the 2002 Dayton Hamvention, Icom whetted our appetite with the D-STAR line of radios, specifically the ID-1. Well, I waited and waited and alas, at the Dayton Hamvention in 2003, I gave HRO a lot of money to reserve me two of the radios when they came out in a few months. Well, a few months passed, no radios. Dayton 2004 even passed with no radios. However, just as 2004 ended, Icom America shipped the long anticipated ID-1 digital voice and data radios.

Analog Voice - USB Control

What is the ID-1? It is a 1.2-GHz radio with 1 and 10 watts output. It looks like any other mobile. However, one of the first differences is the USB pigtail in the back. The USB port allows for complete control of the radio without the control head. That means that you can plug the microphone directly into the main unit, add power and an antenna, program it by computer and you are on the air. No control head is required for operation, great for those situations where you do not want any knob twiddling!

Digital Voice and Data

There is another use of the USB port: a low speed data mode at 2400 bps. While not exactly blindingly fast, the 10-ms TR delay means that you can get better throughput than you can on 1200 or even 2400 bit/s packet implementations.

With the ID-1 in Digital Voice and Data mode, an AMBE (2.4Kbps) CODEC provides for what Icom is referring to as "Toll Quality" voice. I will admit that it sounds good, a lot better than some of the other Amateur Radio implementations, even better than many cellular providers.

Digital Data

Looking at the rear of the ID-1, there is yet another pigtail. It looks like a standard microphone RJ-45, but in reality, it is an Ethernet RJ-45! Cool!!!

This is where the ID-1 stands out from just about any radio: the ability to transmit Ethernet protocols. At 128 kbit/s, the ID-1 provides "faster than the phone modem" connectivity. In addition, when connected to the Internet, it is possible to browse web sites at decent speed.

ID-1 Applications

The ID-1 provides a basic building block that many different types of applications may use. Short message services in the Digital Voice and Data mode allow for Instant Messenger types of applications. The ID-1 control head will even display messages.

The killer application for the ID-1 has to be Ethernet. It provides a transport that is compatible for just about anything. I have used Echolink, sent e-mail, received weather maps, sent pictures, and even read *Packet Status Register*. This radio has the promise to bring Amateur Radio into the 21st century.

The ID-1 is only one of the radios that Icom is producing that is a part of the D-STAR system developed by JARL and Icom. There is a lot more in the pipeline.

What can it do for you?

###

Confronting AX.25 V2.2

By Jim Wagner, KA7EHK, wagnerj@proaxis.com

Two years ago, I mused about some of the problems of packet radio and whether or not software could help [1]. Recently, I've started trying to turn some of those ideas into working code. Now, I have come to the point where I must confront the specification for ax.25, V2.2 [2].

I see some really big problems with that spec and I wonder whether or not anyone else has solved them. The biggest seems to be the inclusion of modulo-128 sequence numbers. Unnumbered frames contain no sequence numbers, so there is no change with U-frames. Supervisory frames can now have a control field containing 1 or 2 octets, but there is no PID field or Information field, so the size of the control field can be uniquely determined without reference to any preceding packets.

Not so for I-frames. I have been unable to identify any feature of an isolated I-frame that will allow determination of the structure of its control field. When monitoring packet frames, it now appears necessary to maintain a history of packets between station-pairs in order to determine what the I-frame format is. Rob, PE1CHL, proposed in 1995 [3] that one of the address field "R" bits be used to designate which control field format is in use. That seems like an eminently practical idea but it didn't seem to make it into the specification. So, my first question is: Has anyone actually put modulo-128 packets on the air (in an environment containing both modulo-8 and modulo-128 sequence numbers) and, if so, how is monitoring managed? Or, am I missing some key fact?

A second problem has to do with the new "Parameter Negotiation". Section 6.3.2 of the specification says "Parameter negotiation occurs at any time"? Does this mean any time after making a connection? Or, does this mean exactly as written - any time? For example, A connects to B, and asks for modulo-128 sequence numbering; B says "OK" and the exchange proceeds, ending in a disconnect. On the next connect, A might logically assume that the previous negotiation is still valid while B may have determined that modulo-8 is in order. Is it B's responsibility to renegotiate? Or, is a new negotiation expected with every new connection? Is there any consensus as to which initiates a negotiation - the station initiating the connection, perhaps?

The third problem is with the state diagrams now included within the ax.25 specification. There is a primitive for Layer 3 to request a connection, "DL-CONNECT Request." There is a primitive for the Data Link to report that the connection request has been satisfied, "DL-CONNECT Confirm." But, no indication when the Data Link has tried and failed; Layer 3 would really like to know! Similar situations appear with several other inter-process primitives. How do programmers deal with this sort of thing? Do you just add primitives to suit your needs? Or do you ignore the state diagram and just charge on through?

A fourth problem has more to do with understanding than anything else. As I started reading and dissecting the specification, I was sure that the Link Multiplexer would be used to manage the distribution of outgoing packets to various radio ports; that is, after all, the essence of "multiplexing"!" The more I read, the more I understand that this is *not* what is intended for the Link Multiplexer. But, I do not understand what it *is* supposed to do, nor how the issue of radio ports is to be managed. On the surface, it would appear that the issue of multiple radio ports was ignored in the specification; am I missing something?

With all of these issues, I wonder about some of the less obvious changes from V2.0 to V2.2. For example, what would be the consequence of violating the new limit of two digipeater addresses? The state machines never seem to test for the number of address fields in packets passed through via digipeating. Nor is the number of address

fields tested in a packet addressed to that station; is a packet containing more than 2 digipeater addresses to be rejected? So, it would seem only to affect packets generated by that station. Beyond the new TEST and XID packet, which seem relatively benign, we have changes in the handling of FRMR and the new SABME. Has anyone put any of these into practice (and care to comment about performance)?

Next is the inter-relationship between KISS hardware and the ax.25 state machines. Clearly, KISS resides somewhere below the Data Link. But, which functions of the Physical State Machine are implemented within the KISS hardware and which, if any, need to be implemented in host software remains a bit of a mystery. Among other things, the KISS interface provides none of the reverse messaging implied in the ax.25 specification. How have other software writers dealt with the KISS interface issue?

Speaking of KISS, there are a number of implementation challenges with respect to KISS that are not really part of ax.25. One such challenge is management of the message rate into the KISS TNC to prevent needless overrun of TNC transmit buffers. Certainly, there is something better than just throwing data at it as fast as the serial port allows. When I have contacted other developers, the answer is usually something along the lines of "Gee, I really don't remember. Why don't you look at the source code?" There has to be something better!

Certainly, I am not the only person to have questions about protocol and implementation matters such as these. Yet, I can find no discussion forum, no archive, nothing that can be referred to! While programming activity is not very high, particularly in the U.S., it must be happening. Why not open a forum (list), perhaps following the lead of APRS developers? TAPR should be able to host such a forum and it would provide an invaluable exchange. If someone needs to be the moderator of such a list, I will offer my services and if TAPR isn't interested, then a medium such as Yahoo Groups ought to work. If we don't get our arms around the issues, we are going to be stuck with old software that does not meet user's expectations and packet use will continue to wither. If you are interested in such an exchange, or just want to provide some response to questions, please contact the author at **wagnerj@proaxis.com**.

[1] Revitalizing "Plain" Packet, Jim Wagner, KA7EHK, TAPR *PSR* #87, Spring 2003, pp 6-8.

[2] *AX.25 Link Access Protocol for Amateur Packet Radio, Version 2.2, Revision: 11 November, 1997*, William A. Beech, NJ7P, Douglas E. Nielsen, N7LEM, Jack Taylor, N7OO, Tucson Amateur Packet Radio Corporation

[3] **http://www.ir3ip.net/iw3fqg/doc/ax25ext.htm**

###

PIC-et Radio IV: How to Send PSK31 Data Using Inexpensive PIC Microcontrollers

Initial Draft

By John A. Hansen, W2FS, hansen@fredonia.edu

Author's note: This paper is the fourth in a continuing series of papers that explore the possibilities of using microcontrollers to implement digital communication technologies. The first three papers in this series were originally presented at various TAPR/ARRL Digital Communication Conferences and can be found on the "documentation" page at http://www.tnc-x.com. This paper describes an ongoing project and is subject to revision as that project is completed.

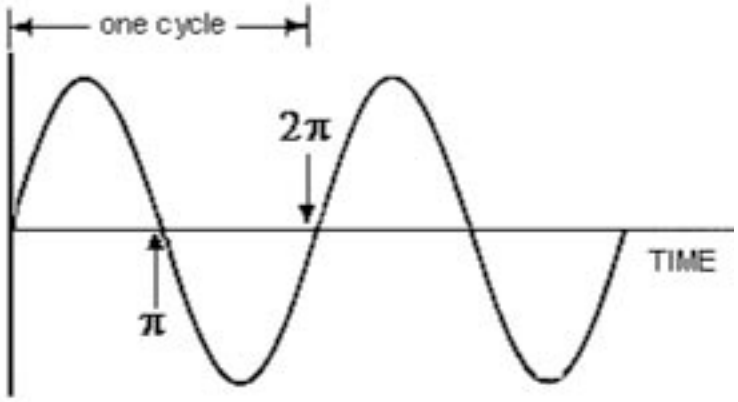
PSK31 is a wildly popular digital mode used on the HF bands mostly for keyboard-to-keyboard communication. It was developed by Peter Martinez, G3PLX (see **http://psk31.com/G3PLXarticle.pdf** for details). Software is available to operate this mode on most PCs using PC soundcards to do the digital to analog and analog to digital conversion. The mode is a natural for low power transmitters because it is so efficient solid copy can be obtained with extremely weak signals. Relatively simple (even indoor) antennas have also been used for reliable communication using this mode. Given this, it would seem to be a great mode for QRP portable operation with, say, a Yaesu FT-817 or ICOM 703. However, it does require a PC with a sound card and to date, there are no handheld PC options available that meet this requirement. Thus it would seem that a laptop at least would be required. However, It seems to me that lugging a laptop along could take a lot of the fun out of QRP portable operation.

Interestingly enough, the AOR TDF-370 (see picture) is capable of receiving PSK31, but for some reason they didn't include the ability to transmit it. This seemed rather odd to me since I'd thought that receiving the data would be the hard part, not transmitting it. I think the TDF-370 may have severely limited its market by not including this feature. To be honest, my longer-term goal in this project was to develop a terminal that would both send and receive PSK31, perhaps using an inexpensive Palm Pilot running a terminal program. But I decided to start with the transmitting side since at least there was a non-laptop option for receiving already available.

PSK31 works by sending a single tone. Data is indicated by either doing a phase shift of the tone by 180 degrees to indicate a digital zero or not doing this shift to indicate a digital one. Martinez developed a "varicode" encoding scheme for text where each letter is represented by a string of ones and zeros that is between 1 and 10 bits long. Shorter strings of bits are used to represent the more common characters. The beginning of a transmission is indicated by a string of zeros and the conclusion is marked by a string of ones. The bits are timed to be sent at a rate of 31.25 bit/s. Peter says he picked this rate because it could easily be derived from the 8-kHz sample rate used in many DSP systems. Since this was not going to be my approach to the problem, I had to determine a frequency to use that could be easily adapted to this bit rate. I wanted the phase shift (if there was going to be one) to occur at the point where the audio sine wave crossed the zero point. Thus, I wanted

to pick a tone that would have a whole number of cycles in 31.25 seconds. Thus, each bit period should be 1 / 31.25 = 32 ms. Now, if I selected a 375-Hz tone, each cycle of the tone would take 2 2/3 ms., so 12 cycles would take exactly 32 ms. So if I transmit a 375-Hz tone and shift the phase of it (or not, depending on whether a zero or one is sent) every 12 cycles, I'll be sending PSK31. Similar calculations can show that if I transmit a 750-Hz signal and change the phase (or not) every 24 cycles, I'll also be sending PSK31.

The first step down this road was to get the PIC processor to send a sine wave. Fortunately many people have been down this road before me. Some have chosen to use the microcontroller's Pulse Width Modulation (PWM) capabilities to do this. In the case of the PIC that I'm using, this would only allow me to obtain 32 different voltage levels over the course of an audio sine wave. While this is certainly adequate for 1200-baud packet, I decided to start with something that would at least allow the possibility of producing a better approximation of a sine wave. I did this because the usual mechanism for sending PSK31, a PC sound card, is capable of achieving a much better approximation than this. A packet signal only requires that the receiver discern whether the audio frequency is 1200 or 2200 Hz. PSK31 has to detect the phase at specific points in time and, knowing relatively little at this point about how PSK31 was decoded, this seemed to me to be a more difficult task than determining the frequency. Of course, I could be wrong about this, and I intend to do some more experimentation to how robust a system is needed.



Sine Wave

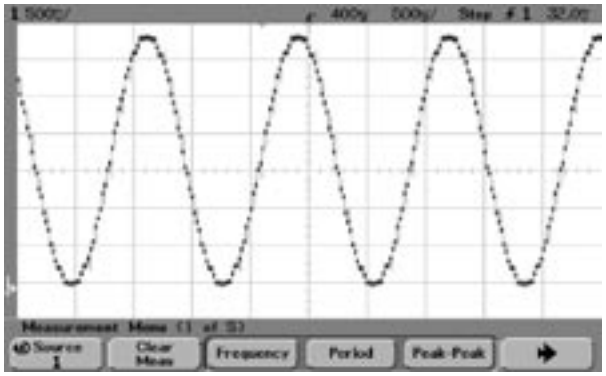
A second approach for generating a sine wave is to use a "resistor ladder." Byon Garrabrant, N6BG, uses this approach, for example, in the TinyTrak. Within this general approach, there are two possibilities. Byon uses four PIC pins and connects resistors to them such that they have values that double when moving from pin to pin. Thus he connects a 1k resistor to the first pin, a 2k resistor to the next, and so on. The idea is that turning on different combinations of pins will result in 16 different voltage levels. As I noted above, I was concerned that this would not provide me with enough different voltage levels to give a sufficiently close approximation to a sine wave for PSK31. So, using this approach I would need more pins. It gets hard to find the appropriate resistor values to keep doubling for more and more pins, so I used a second approach that was outlined in a Microchip Application Note (AN-655 available on www.microchip.com). This approach uses twice as many resistors (two for each pin), but only two different values are needed altogether. These values are readily available in 16-pin DIP packages, so if I ever decided to do a commercial design based on this, I could simply use two of these DIPs (one for each value).

I decided to use seven PIC pins, which would produce 128 different possible voltage levels. I settled on seven because I wanted to use a relatively small, inexpensive PIC (one of the 18 pin models). Eight of the pins on this chip constitute something called "Port B." You can set the state of all 8 pins at one time because a single register in the chip controls them all. Thus, only one clock cycle is required to set all the pins. However, one of those pins is connected internally to the receiver on the

chip's serial port. I wanted to use the hardware serial port in the chip to receive data from the Palm Pilot (or other terminal) so that pin would not be available for the resistor ladder. This left me with seven pins.

The next step was to figure out how many points along the sine wave that I wanted to set for a single cycle. I suppose the absolute minimum would be four, marking the top of the cycle, the bottom and the points where the cycle crossed zero. This way one could in theory figure out where the zero crossing was and determine whether the cycle continued on in the same direction (indicating a one bit) or reversed direction and headed back toward the top or bottom (indicating a zero bit). However, since I really didn't know exactly how PSK receive systems worked, I was almost certain that this wouldn't be enough data. I arbitrarily picked a value 64 for the number of points that I would use to specify the proper voltage level.

The next step was to figure which pins to turn on at each of the 64 points on the cycle. To do this, you first divide the sine wave into 64 parts and then calculate the sine for each of those points. Remembering my high school mathematics, it is a lot easier to do this using radians than it is using angle degrees. An entire sine wave is 2π in length so each 64th part is $\pi/32$. So, to start, one calculates the sine of each of the following values: 0, $\pi/32$, $2\pi/32$, $3\pi/32$ and so forth up to $63\pi/32$. Using either a calculator, a sine table or your trusty slide rule, this will give you a sine value that ranges for -1 (at $48\pi/32$) to 1 (at $16\pi/32$). The voltage produced by the PIC and resistor ladder will not range from -1 to 1 however, so it is necessary to rescale these values so that the peak corresponds to the highest voltage you can get out the ladder (where all seven pins are turned on) and the bottom corresponds to the lowest voltage you can get out of the PIC (where all seven pins are turned off). Then it is necessary to figure out the pin configuration that will produce a value very close to the value on the re-scaled sine wave. With 128 different voltage possibilities it is possible to get pretty close! Of course there is a DC bias to this signal because the PIC pins cannot produce negative voltages, but this can be removed by running the output signal through a capacitor. I took a look at the resulting waveform with my oscilloscope and it looked as follows:



Clearly, we've got a pretty good approximation of a 750 Hz sine wave here. In terms of the programming code, I created an array (called wave) of the values that needed to be written to the Port B pins in order go from the top of the sine wave to the bottom. It would have been redundant to also include the values that were necessary to go from the bottom back to the top, since these values were identical to the first set. What I needed to do was simply to step through the array from start to finish. When I reached the end, I then stepped through the array backwards to form the other half of the wave. I used a variable called point to keep track of where I was in this process. Since the peak and bottom values are both included in this array, it has (64/2) + 1 = 33 values. The code to accomplish this is pretty simple:

```
PORTB = wave[point];
if (point == 32) up = true;
if (point == 0) up = false;
if (up) point--;
    else point++;
```

A variable called 'up' keeps track of whether we are moving up the sine wave or down. The first line of code sets the voltage on the output. If we've reached the last point of the array, it means we've

reached the bottom point on the sine wave. So 'up' is set to true. If we have reached the first element of the array, it means we must have reached the peak of the sine wave, so 'up' is set to false. When up is false we move forward through the array; when 'up' is false, we move backward.

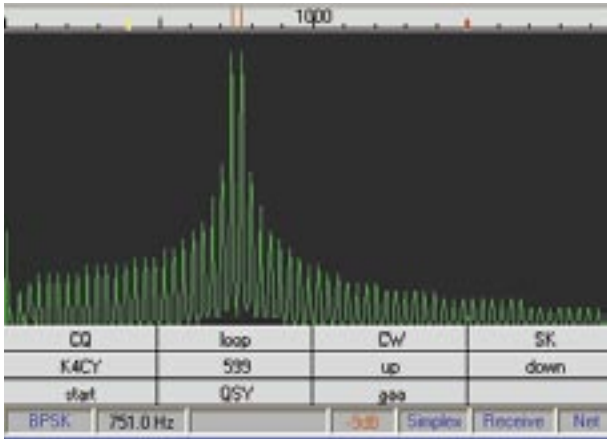
The only thing lacking here is the timing mechanism. If we ran the above routine without any delay, it would produce a frequency much higher than the desired 750 Hz. So this code was placed in an interrupt service routine. I arranged for the PIC to fire an interrupt in such a way that the resulting wave was 750 Hz.

But that doesn't send any data, it's just a sine wave. In order to send a string of zeros, it is necessary to change the phase of this signal by 180 degrees every 24 cycles. Such a signal would look like this on an oscilloscope:

Note that this type of phase shift can be accomplished by simply reversing the direction of the movement through the data array at the halfway point. The modified interrupt service routine that does this looks something like this:

```
portb= wave[point];
if (point == 32) up = true;
if (point == 0) up = false;
if (point==16){
    cycle++;
    if (cycle == 48) {
            if (flip) up = !up;
            cycle = 0;
    }
    flip = true;
}
if (up) point--;
    else point++;
```

The variable 'cycle' counts the number of zero crossings that have occurred since the last phase change was made. A zero crossing occurs half way through the array, at data point 16. Since there are two zero crossing per sine wave cycle, we need to get 48 of them in order to complete 24 sine wave cycles (for a data rate of 31.25 bit/s). A variable called 'flip' determines whether the phase shift should be made or not. If a phase change is to be made, flipping the 'up' variable causes the direction through the array to be reversed. At that point the cycle count is also reset to zero. I've designed this program so that the default data bit sent will be a zero, so each time through zero crossing routine the 'flip' variable is set to true. If the program needs to send a one instead, it simply changes the value of this variable to false and the next time 'cycle' equals 48 no phase change will be accomplished. The nice thing about doing it this way is that the main program itself can have



virtually no code in it all and the unit will idle by sending a series of zeros.

Here is a spectrogram (made with the Zakanaka PSK31 program) of the resulting signal when the unit idles by sending zeros. Note that the IMD

figure on the display is -9dB. This is pretty bad and indicates that some additional work is needed. It is recommended that transmitter produce IMD values of at least -23 dB. However, the current program is a first hack only. When I changed the timing so that a 775 Hz wave was produced, for example, the IMD rose to almost -23 dB. However, this fouls up the bit rate (it's no longer 31.25 bit/s) and so copy of the data became pretty spotty. I'm hoping that some tweaking of the sine wave will resolve this issue. It's worth noting, however, that despite the poor IMD figure, the receiver is able to perfectly copy this signal, even at very low signal levels.

In order to actually send data, it is necessary to have the PIC receive data over the serial port and encode the data in the phase shifts of the sine wave. To hold the data that needs to be sent, I created another array that has room for 80 bytes (called 'text[]'). PSK31 is mostly used for keyboard-to-keyboard QSOs. As a result, large amounts of data do not generally need to be held in memory waiting to be transmitted. I could have allowed a buffer larger than 80 characters, but to do that, I would have had to either select a much larger PIC or add a memory chip (the latter was the approach I took with the TNC-X project). Currently, I am using a PIC16F628A chip for this project. It's extremely cheap (under $2) and it has a built-in hardware serial port.

The data buffer is configured as circular buffer, so when the end of the buffer is reached, it wraps around to the beginning. Two variables are used as pointers that indicate the next place in the buffer that data should be added and the next place in the buffer that data should be removed for transmission. A third variable keeps track of the number of bytes awaiting transmission. This variable is not strictly necessary, but it makes the program simpler. In the mainline of my program, every now and then I call a routine that checks to see if there is a byte of data on the serial port waiting to be received and if there is, it moves it to the next location in the buffer. The code that does this is relatively simple:

```
    if (bit_test(PIR1,5)){
       text[receivepoint] = getc();
       receivepoint++;
       bytes++;
       if (receivepoint == 80)
receivepoint = 0;
    }
```

Bit 5 of the PIR1 register (PIR1,5) is true if there is a byte to be processed. If so, it is placed in the text array at the receivepoint, the receivepoint is incremented, and the number of bytes received ('bytes') is incremented. The last line wraps the end of the buffer around to the beginning.

When there is data to send (byte>0), the character must be translated into the Martinez varicode system and then clocked out on the sine wave. A lookup table handles the translation to varicode. Two bytes are used to return the varicode value because it is 10 bits wide. Martinez designed the varicode so that two zeros in a row never occur within a character itself, but two consecutive zeros mark the end of each character. This solves the problem of not knowing how many bits should be transmitted (since the length of the varicode can range from 1 to 10 bits). The program simply keeps transmitting bits until it runs into two zeros in a row. Then it knows that the end of the varicode character has been reached. The code to do this looks like this:

```
if (bytes > 0){
   current = translate(text[sendpoint]);
   while ((current & 1) || (lastbit)){
      lastbit = (current & 1);
      while (cycle != 47);
      if (lastbit) flip = false;
      current = current >> 1;
      while(cycle !=0);
   }//end of while
   while(cycle != 47);
   while(cycle != 0);
```

```
    sendpoint++;
    if (sendpoint == 80) sendpoint = 0; //don't overrun the array
    bytes--;
}
```

The variable 'sendpoint' contains the index in the text array that contains the byte to be sent. The variable 'current' holds the varicode value of this byte, which is produced by the translate function. 'Lastbit' contains the bit that was previously sent. This allows the program to determine when two zeros in a row have been sent. The loop that begins with the line "while ((current & 1) | | (lastbit))" allows the program to continue to process the bits until two zeros in a row are located. The phase shift is supposed to occur, if needed, on the 48th zero crossing, so we pause the program until we get to the 47th zero crossing. Obviously we don't want to move on to process the next bit until the current one has been sent! If the bit to be sent is a 1, then the phase shift does not occur (flip = false), otherwise it is left at the default value of true.

When I constructed the varicode lookup table, I reversed the order of the bits so that the first bit to be sent was the rightmost bit. This allows me to simply right shift the value of 'current' to get to the next bit. It is necessary to wait until the last bit has been sent (cycle = 0) before moving on to process the next bit. PSK31 specifies that a pair of zeros should be sent in between each character. One of these is sent by the loop that sends the bits themselves, but the second zero is sent by the pair of while statements after the loop. After the character and two zeros have been sent, 'sendpoint' is incremented to move it to the next character that is due to be sent and if necessary this value wraps around to the beginning of the array. Finally, since a byte has been sent, we decrement the value of 'byte'.

So far, so good. The transmitter produces output that is perfectly readable when routed into my PC soundcard and decoded by Zakanaka. The only remaining problem is the IMD figure, which clearly needs work. It may be that sine wave needs some massaging, or it may be that I need more than 64 data points or 128 different voltage levels to produce an adequate signal. But I think that this initial experiment shows that transmitting PSK31 will be possible with an extremely inexpensive PIC-based system. ###

Eliminating Source Routing from APRS

By Pete Loveall, AE5PL, pete@ae5pl.net

Source routing has been part of AX.25 since its inception. It provided a low-cost method for TNC owners to connect from point A to point B via multiple digipeaters. At the time of the creation of AX.25, router hardware and software were expensive, both from a financial standpoint and from a resource standpoint. This source routing is one of the reasons that networks of AX.25 digipeaters have all but disappeared in the US and much of the rest of the world.

What is source routing? Source routing is where the originator of a packet specifies the route the packet will take to get to the destination. This requires the network users to know the topology of the underlying network. It also requires the users to adjust their paths according to their desired destination.

Robert Bruninga, WB4APR, recognized early on in the development of APRS that specifying specific paths would be cumbersome, if not impossible, for mobile and portable operations using the UI portion of the AX.25 protocol. So the generic aliases of RELAY, WIDE, and TRACE were created to allow simple program-it-once-and-forget-it path settings for the mobile and portable operator. However, there were problems.

APRS is a protocol based on the AX.25 UI packet type. This packet type is unconnected or broadcast. The communication, in essence, is one-to-many. One station's packet is heard and decoded by everyone that receives it. As such, APRS is designed as a tactical form of communications providing local area data reporting and SMS (short messaging) capabilities. Examples of data that is reported are position, weather, telemetry, etc. Hams, being the experimenters that we are, immediately began experimenting with paths to see how far we could expand this "local" area. The result was a mess of packets ping-ponging back and forth through the network.

So WB4APR came up with another solution: UI flood and UI trace. The basic concept is to have one via represent up to seven digipeater hops. Digipeaters implementing these n-N protocols would check for duplicate repeats and therefore, eliminate the ping-pong effect. Initially, this protocol was implemented in the uidigi EPROM for the TNC-2 and in the Kantronics KPC-3 and KPC-3+. Unfortunately, the duplicate check algorithm in the Kantronics TNC is buggy and only looks at the UI flood and UI trace constructs. This means that they still ping-pong if there are other aliases used in conjunction with the UI flood and UI trace constructs. The most common mobile setting, RELAY,WIDE2-2, can cause the KPC-3+ to digipeat the packet multiple times due to call sign replacement of RELAY and no duplicate checking until the WIDE2-2 is repeated.

This was usable, however, in the early development of APRS because even large metropolitan area APRS frequencies were not saturated. This has changed. The APRS frequencies in many parts of the world are saturated. A close examination of the packets on those frequencies shows that on average, over 90% of the packets are from digipeaters. So it is painfully obvious that the way to improve frequency utilization is to eliminate unnecessary digipeats.

What are "unnecessary digipeats?" The answer to this question lies in the base design of APRS: "APRS is designed as a tactical form of communications providing local area data reporting and SMS (short messaging) capabilities." Source routing causes reliance on individual operators to know what "local area" is and then to program their radios accordingly. The "new n-N paradigm" does nothing to address this underlying problem, makes travel between areas more difficult for the traveler with preprogrammed trackers, and simply gives hams new ways to create destructive paths.

OK, so how do we address this problem with the equipment that we have now? We can't unless we can get Kantronics to add a no-source-route option. If that can be done per the specifications below, then slowly but surely, source routing will disappear from APRS and it will become a usable protocol again. The digipeater specification in this paper is easily implemented in a TNC-2 EPROM as well as with any software digipeater such as Digi-Ned and javAPRSDigi.

The basic digipeater algorithm is as follows:

1. The digipeater repeats everything it sees directly (no digipeated packets), stripping the entire path away and replacing it with just the digipeater's call sign with the H-bit set.

2. The digipeater repeats any packets it sees digipeated by digipeaters on its "ok" list. This allows remote areas to make it into the "metro" LAN as deemed proper by the digipeater sysop. The digipeater will modify the path by simply appending its call with the H-bit set after the "ok" list digipeater call.

RELAY can be on the "ok" list allowing people to set up low-level RELAY alias digipeaters. Packets with RELAY in the path would only be digipeated if RELAY is in the first position and no place else.

3. The digipeater does full dupe-checking (CRC or checksum) based on from call, unproto, I field length, I field data. The digipeater will not digipeat any packet where its call sign appears before or including the call with the H-bit set in the path. The depth of this dupe check would only need to be about 30 packets long.

This is a very simple algorithm. The decision of "what makes up the local area" would now lie in the hands of the wide-area digipeater sysops. The users could still use a path, for instance, of RELAY,WIDE2-2, for areas not covered by such a digipeater yet they would be properly digipeated in areas where these types of digipeaters would exist. For simplicity sake, let's call these digipeaters UI no-source digipeaters. One of the biggest benefits to the UI no-source digipeater: *no* user ever needs to know the network topology again. The digipeater sysops take care of this just as the Internet service providers make it so no user of the Internet needs to know the actual network topology.

What would this mean to an area like the DFW Metroplex (north Texas, USA) where the frequency is saturated? The Collin County wide-area digipeater would digipeat anything it hears directly or directly digipeated by the wide-area digipeater in the southwest portion of the county (we have big counties). Tarrant County wide-area digipeater would only digipeat what it hears directly. Etc. etc. etc. All of a sudden, we go from 150 to 200 stations competing for the frequency down to 10 to 20 stations competing for the frequency.

What about when we "need" to communicate farther? There are two options:

1. APRS-IS - APRS-IS IGates provide inter-LAN connectivity worldwide, nationwide, statewide, and area-wide. This is the most available option and most usable on a day-to-day basis.

2. In an emergency where APRS-IS might not be locally available, the digipeater sysops modify their "ok" lists. This requires no action by the individual users (important during an emergency) and reduces the number of changes implemented to an absolute minimum.

It is interesting to note that APRS-IS has always been free of source routing, even though some have tried to get implemented different source-routing variations. It has been important to the integrity and usability of APRS-IS that it does not implement any type of source routing.

This paper is designed to provoke thought, not as an answer to all of the source-routing ills of APRS. I am sure there are modifications and tweaks that can be done. However, this is a start towards simplifying the use of APRS for everyone while making it more usable everywhere per its original intent.

###

Inexpensive GPS25 Offered

By John Koster, W9DDD, w9ddd@tapr.org

Garmin has offered TAPR the opportunity of selling one of their products, the GDL-47, at an especially attractive price. The Garmin GDL-47 is a small plastic box that contains a GPS25LVC and an associated embedded processor board.

There is no documentation available for the GDL-47 itself, however, for the price, you can pull out the GPS-25LVC and throw the rest away. The GPS-25LVC is well documented and is very similar to the GPS-25LVS that TAPR has sold for a number of years.

The difference between the GPS-25LVS and GPS-25LVC is the serial ports. The LVS meets RS-232 specifications; the LVC has CMOS logic levels. This should not be a problem in most cases. Most serial devices will interface with the LVC without problem. One known exception is the 1995-vintage IBM ThinkPad.

There is one difference between this GPS-25LVC and the standard Garmin GPS-25LVC product. Instead of the antenna connector being soldered to the PCB, there is a short (1.5-in) coax pigtail terminated in a bulkhead MCX connector.

These are new units and carry the same warranty as our original GPS-25LVS. For general information on the GPS-25, see the GPS-25 page at **http://www.tapr.org**.

Ordering Information

The price for the GDL-47 is $72.00 US for members of TAPR, $80.00 US for non-members plus shipping and handling. You may order online at **http://www.tapr.org**.

###

Packet Status Register

#94 Winter 2005, ISSN: 1052-3626

Published by
TAPR
8987-309 East Tanque Verde Road #337
Tucson, AZ 95749-9399 USA
phone 972-671-TAPR (8277)
fax: 972-671-8716
e-mail **tapr@tapr.org**
URL **www.tapr.org**
TAPR Office Hours
Monday – Friday, 9 AM – 5 PM Central Time

Postmaster: Send address changes to TAPR, P. O. Box 852754, Richardson, TX 75085-2754. *Packet Status Register* is published quarterly by TAPR, 8987-309 East Tanque Verde Road #337, Tucson, Arizona 95749-9399 USA. Membership in TAPR, which supports the electronic publication of the *Packet Status Register*, is $20.00 per year payable in US funds.

TAPR is a community that provides leadership and resources to radio amateurs for the purpose of advancing the radio art.

Submission Guidelines

TAPR is always interested in receiving information and articles for publication. If you have an idea for an article you would like to see, or you or someone you know is doing something that would interest TAPR, please contact the editor **(wa1lou@tapr.org)** so that your work can be shared with the Amateur Radio community. If you feel uncomfortable or otherwise unable to write an article yourself, please contact the editor for assistance. Preferred format for articles is plain ASCII text (Microsoft Word is acceptable). Preferred graphic formats are PS/EPS/TIFF (diagrams, black and white photographs), or TIFF/JPEG/GIF (color photographs). Please submit graphics at a minimum of 300 DPI.

Production / Distribution:

Packet Status Register is exported as Adobe *Acrobat* version 5 and distributed electronically at **www.tapr.org**

PSR *Packet Status Register* Editor:

Stan Horzepa, WA1LOU
One Glen Avenue, Wolcott, CT 06716-1442 USA
phone 203-879-1348
e-mail **wa1lou@tapr.org**

Item	Price	Member Price	Qty	Total	Kit Points
TAPR MEMBERSHIP					
New		$20.00			0
Renewal, Enter Membership Number here:		$20.00			0
KITS					
DSP-10 2-Meter Transceiver	$329.00	$299.00			56
KK7P DSPx DSP Module	$99.00	$99.00			16
KK7P DSP10 Adapter Kit	$39.00	$39.00			16
PIC-E(ncoder)	$65.00	$58.50			16
Motorola EVM56002 Interface	$150.00	$135.00			16
Compact FlashCard Adapter (FlashCard not included)	$49.00	$39.00			16
DAS (DTMF Accessory Squelch) (as seen in December 1995 QST)	$68.00	$61.20			8
Bit Regenerator (for regenerative repeater operation)	$10.00	$9.00			1
Clock Option (for regenerative repeater operation)	$5.00	$4.50			1
PK-232 Modem Disconnect (to simplify external modem connection)	$20.00	$18.00			2
PK-232MBX Installation Kit (for 9600-bit/s modem installation)	$20.00	$18.00			2
XR2211 DCD Modification	$20.00	$18.00			2
State Machine DCD Modification	$20.00	$18.00			2
State Machine DCD Modification with Internal Clock (for KPC-2)	$25.00	$22.50			2
FIRMWARE					
TNC2 Version 1.1.9 with KISS EPROM (includes command booklet)	$15.00	$13.50			4
TNC2 Version 1.1.9 command booklet	$8.00	$7.20			2
TNC2 WA8DED EPROM (ARES/Data standard 8-connection version)	$12.00	$10.80			2
TNC1 WA8DED EPROM	$12.00	$10.80			2
TNC2 KISS EPROM	$12.00	$10.80			2
TNC1 KISS EPROM	$12.00	$10.80			2
PK-87 WA8DED EPROM	$12.00	$10.80			2
TrackBox EPROM	$15.00	$15.00			2
MX-614 Modem IC	$8.00	$8.00			2
PUBLICATIONS					
Digital Communications Conference (DCC) Proceedings					
2002 DCC No. 21 (printed copy)	$20.00	$18.00			8
2001 DCC No. 20 (printed copy)	$10.00	$9.00			8
2000 DCC No. 19 (printed copy)	$15.00	$13.50			8
1999 DCC No. 18 (printed copy)	$15.00	$13.50			8
1998-2000 DCC Nos. 17-19 (CD & available printed copies)	$50.00	$45.00			4
1998-2000 DCC Nos. 17-19 (CD only)	$33.00	$30.00			4
1992-1997 DCC Nos. 11-16 (CD & available printed copies)	$33.00	$30.00			4
1981-1991 DCC Nos. 1-10 (CD & available printed copies)	$33.00	$30.00			4
Earlier DCC Proceedings (printed copies):					
Circle desired nos.: 1-4 5 6 7 8 9	$6.00 ea.	$5.40 e			8
Circle desired nos.: 10 11 12 13 14 15 16 17	$6.00 ea.	$5.40 e			8
TAPR Spread Spectrum Update	$18.00	$15.30			16
TAPR Software Library CD	$20.00	$18.00			4
Wireless Digital Communications	$39.99	$36.00			28
Packet Radio: What? Why? How?	$12.00	$10.80			8
BBS SYSOP Guide	$9.00	$8.10			8
Packet Status Register Vo. 1 (Nos. 1-17, 1982-85)	$20.00	$18.00			16
Packet Status Register Vo. 2 (Nos. 18-36, 1986-89)	$20.00	$18.00			16
Packet Status Register Vo. 3 (Nos. 37-52, 1990-93)	$20.00	$18.00			16
Packet Status Register Vo. 4 (Nos. 53-68, 1993-97)	$35.00	$31.50			16
OTHER					
TAPR Badge with Name and Call Sign	$10.00	$10.00			0
TAPR 11-oz. Coffee Mug	$11.00	$10.00			0
GPS EQUIPMENT					
TAC-32 Software Registration	$55.00	$55.00			0
Garmin GPS-20/25 Interface/Power Kit	$40.00	$36.00			8
Garmin GPS-20/25 Data Cable	$15.00	$15.00			2
Garmin GA-27 GPS Antenna (w/MCX conn., mag. & suction mounts)	$75.00	$67.50			8
Oncore GT+ GPS	$149.00	$129.00			28
Motorola Antenna 97 (w/MCX connector and magnetic mount)	$30.00	$27.00			8
MCX Right-Angle Connector with Coaxial Pigtail	$15.00	$15.00			2

TAPR Order Form

SUB-TOTAL

SALES TAX (TEXAS RESIDENTS ONLY, 8.25%)

SHIPPING:

TOTAL ORDER AMOUNT

SHIPPING:
1-7 Kit Points = $6
8-15 Kit Points = $7
16-27 Kit Points = $8
28-54 Kit Points = $9
55 or More Points, Contact

TAPR Business Office
P.O. Box 852754
Richardson, TX 75085-2754
Phone (972) 671-8277
Fax (972) 671-8716
E-mail tapr@tapr.org
Internet www.tapr.org

Check Enclosed __ or Charge My Credit Card: VISA __MasterCard __

Credit Card Account Number_____ Expiration Date___/___/___

Signature_____

Name_____ Call Sign_____

Street Address_____

City – State – ZIP Code_____ Country_____

Phone Number_____ E-Mail Address_____