An Inexpensive Terminal Node Controller for Packet Radio

An inexpensive processor chip forms the basis for a simple-to-build TNC.

By Bob Ball, WB8WGA

Most hams who have tuned around on the 2-meter band at one time or another have heard the infamous *blurrp* of packets being sent back and forth. Sometimes it is an informal QSO, sometimes an Amateur Packet Position beacon, or maybe someone sending info on a hot DX station. Packet radio has found a wide number of applications since its introduction in the 1980s.

Did you ever want to have an inexpensive way to monitor local packet activity? Or perhaps set up your own digipeater to get a communications link to a particular point? Or possibly, actually do some programming on your own simple Terminal Node Controller (TNC)?

If so, this simple project might be the project you have wanted. The unit can easily be built for under $25. After it is completed, the software supplied will:

- monitor local packet activity
- act as a complete digipeater
- send packet beacons at user defined intervals containing your defined text
- allow you to communicate in a round-table fashion using "converse" mode
- send APRS NMEA position reports when connected to a GPS receiver.

Note that this unit does *not* provide all functionality of the complete commercially available TNC units. It is a simplified version that captures enough functionality to get you on the air and have some fun with packet radio. If you want to expand your programming knowledge and explore AX.25 a bit, the unit described will allow you to modify the software to support just about any application you might have been considering for packet radio. This could include, for example, anything from temperature monitoring, GPS interfaces or remote control applications. The list of possibilities goes on and on. All source code for the unit is provided as a starting point and programming is possible without buying an expensive programmer. We will provide additional information on programming the processor later in this article.

I got started on this project after looking at the functionality of some of the new microprocessor chips. They just keep getting more powerful while the cost keeps dropping. After checking out a few different ones, I became attracted to the *MicroChip* series, in particular, one designated the

23 Ingerson Rd
Jefferson, NH 03583
wb8wga@arrl.net

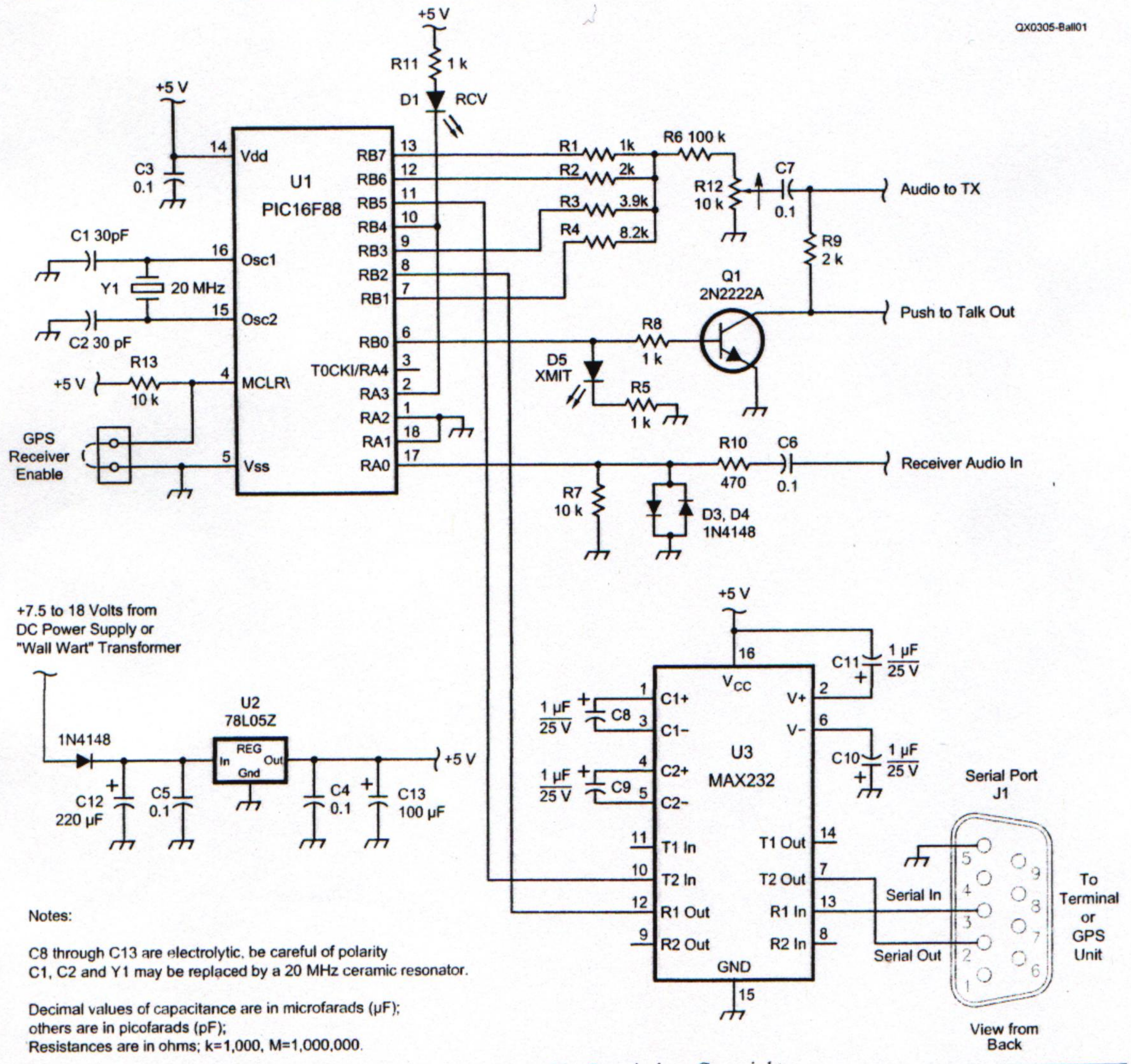
PIC16F88. This 18-pin dual-inline chip sells for about $3 in single units. It provides several analog to digital (A/D) channels, includes a UART to talk to a terminal, a programmable Flash program memory and a programmable EEPROM memory to store call signs and options. It can execute a program instruction every 200 nsec. Amazing! It is certainly not the Intel 8080 I used back in the early 70s! After looking at the unit I began to wonder if it could form the basis of a simple TNC.

One of the things I discovered is that many of the existing PIC processor based packet TNCs use the MX614P modem chip. Modem chips are great building blocks but do add to power consumption and cost. In addition, they can be difficult to obtain. Most of the existing TNC project designs use processor chips that have been replaced by more powerful and less expensive units.

This article will describe how to use the enhanced internal capability of these new processors to code and decode the digital signals. While the project described here is designed for packet radio, the project board and modemless techniques described should work for just about any digital mode (RTTY, Pactor, etc). So, once you have built this unit for packet use, with a little work and ingenuity, it can be reprogrammed for enhanced packet radio or other amateur digital modes.

As with many ham projects, I was able to build on the published work of many others. In particular, Mike Berg, NØQBH,[1] has done some work on modem-less receiver design using an external comparator. A number of PIC programmers[2] have documented resistor ladder network designs for older technology PIC processors that gen-

[1]Notes appear on page 25.



Notes:

C8 through C13 are electrolytic, be careful of polarity
C1, C2 and Y1 may be replaced by a 20 MHz ceramic resonator.

Decimal values of capacitance are in microfarads (µF); others are in picofarads (pF); Resistances are in ohms; k=1,000, M=1,000,000.

Figure 1—Schematic of the inexpensive TNC.

erate a clean sine wave needed to transmit Audio Frequency Shift Keying (AFSK) tones for transmit. I have combined this existing receive and transmit work, added a command interface which is patterned after the Tucson Amateur Packet Radio (TAPR)[3] terminal node controllers, and made use of the newest PIC technology to produce a simple TNC. It provides the following features:

- A command interface is provided that allows configurable options to be set and stored in EE ram and restored on subsequent resets.
- A monitor feature is included to allow all packets, no packets, or just packets addressed to my station to be displayed on the terminal attached to the serial port.
- Digipeating of packets (up to 255 bytes/packet) is supported.
- Alias capability allows the unit to function as RELAY type digipeater.
- Beacon capability with user set parameters is supported.
- Converse mode operation to transmit text typed from the serial port supporting "round table" chat type operation is provided.
- Support is provided to transmit APRS position beacons when a GPS unit is attached to the serial port.

Circuit Description

A schematic of the TNC is shown in Figure 1. It contains two integrated circuits, one for level conversion of the RS232 signals and the microprocessor. The microprocessor is the 18-pin DIP version of the PIC16F88. The popular MAX232 chip does the RS232 conversion.

Connections to the radio consist of audio input from the speaker jack, audio to the microphone circuit, and a connection to the push to talk circuit.

The serial port is used to connect to a standard terminal program or to a GPS receiver that outputs NMEA sentences. The terminal program is used to display received packets, send text while in the CHAT mode, and do some initial setup of the unit. Jumper J4 is used to specify what is connected on the serial port.

Note that after initial setup is complete and the station parameters are set in EERAM, the unit will run as a super simple remote digipeater without the MAX232 and serial port connections. With this configuration, it becomes a one-chip TNC.

Power to the unit can be anything from 7 to 18 V. Power from the 12 V radio can be used, or an inexpensive *wall wart* will do the job.

Receiving Packets

Packet radio, in common with many digital transmission techniques, uses *Audio Frequency Shift Keying* (AFSK) to send the stream of *ones* and *zeros*. It uses two frequencies, 1200 and 2200 Hz to represent a change in the value of the bit stream. For receiving purposes, this TNC uses a comparator to find zero crossings of the AFSK sine wave. The TNC calculates the interval between zero crossings to determine the frequency present. The software then converts the changes in tones to a data stream. To imagine this, think of a sine wave at either 1200 or 2200 Hz. If we clip the sine wave with a couple of diodes (D1 and D2 in the schematic), we get a square wave. The circuit is shown in Fig 2.

If the resulting square wave is fed to a zero-crossing-detector, a pulse is generated at every crossing. If the processor internally times the interval between these pulses, it can determine the frequency of the sine wave. The F88 processor has timers and an internal comparator circuit so all that decoding work can be done inside the chip. The software inside the PIC can also do some digital filtering and throw out frequencies outside of the passband of the audio. The resulting modem-less design is extremely sensitive and rivals the performance of the commercial modem chips like the MX614. My units routinely decode packets that are less than 1 S unit on my 2-meter radio from stations over 75 miles away.

It should be noted that the above technique could be used for sampling the frequency of any sine wave. It should work for any of the digital modes that use AFSK or for any applications that need to detect an audio tone of a certain frequency.

Transmitting Packets

To transmit, the software needs to generate either a 1200 or 2200 Hz sine wave. The frequency is changed to indicate the transmission of a digital one or zero.

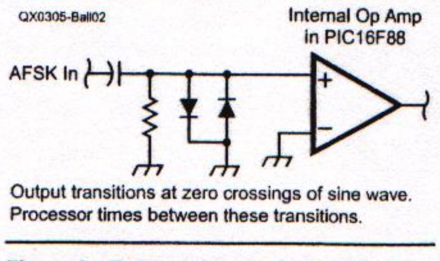

QX0305-Ball02
Internal Op Amp in PIC16F88
AFSK In
+
−
Output transitions at zero crossings of sine wave.
Processor times between these transitions.

Figure 2—Determining the frequency of a sine wave using a zero crossing detector.

A sine wave is no more than a varying voltage level over time. If the processor can change the voltage level in the right amount and time frame, a sine wave with low distortion can be produced. The TNC must only generate a single tone at a time, although production of multiple simultaneous tones (i.e. DTMF) is possible. MicroChip has documented this technique in one of its application notes if you are interested in more details.

This generation can be done in software by controlling a resistor ladder network that has the correct values to generate a sine wave. If the voltage is changed often enough (say 32 times per audio cycle) at the right value, a sine wave is produced. Figure 3 shows the voltages that are generated at the wiper of R12 when the Ports RB7, RB6, RB5 and RB1 are taken through a binary sequence by the program at the correct frequency. Increasing the number of samples decreases the distortion of the sine wave but increases the real-time requirement. A sine wave adequate for this application can be generated using 32 points.

Software

Approximately 3000 words of program instruction reside on the PIC chip to do the job of taking the received packet, storing it, checking it for validity, sending the info to the serial port, and re-transmitting the packet as necessary. If you are not interested in studying or modifying the provided software, a copy of the working assembled software is provided that can be downloaded into the processor (see next section).

If you are interested in learning about PICs or how packets are formed, etc, you might want to modify the code to meet your needs. While the complete software structure will not be described here, a copy of the commented source code is available[4] and can be used as a starting point for further experimentation. The software is organized by functional processes (ie receive packet, digipeat, command processor, etc) and scheduled in a round robin manner to make modification easy. The terminal interface is implemented as a table driven function, so that additional input commands can be easily added.

The software in this unit uses published snippets of code from many hams who have shared their work on the Web. References and credits are given to all the authors in the source software. Please remember to maintain these credits if you modify the