NUE-PSK31

A digital modem for PSK31 field operation… without using a PC!

Milton Cram, W8NUE
Austin QRP Club
9807 Vista View Drive
Austin, TX 78750
E-mail: w8nue@arrl.net

George L. Heron, N2APB
American QRP Club
2419 Feather Mae Ct
Forest Hill, Maryland 21050
E-mail: n2apb@amsat.org

Abstract

PSK31 is one of the latest communications modes to capture the interest of hams worldwide. Its inherent ability to dig out low, near-inaudible signals is ideally suited for low power QRP enthusiasts. The PSK31 digital modem engine, however, requires intense DSP processing that is only commonly available in PC sound card. Thus the PSK operator desiring portability for field operation is locked into using a laptop computer as a controller, which results in a cumbersome station. But there's hope!

This paper presents the design and construction of a standalone, battery-operated digital modem using a Microchip dsPIC microcontroller. The project includes a character display for transmit and receive text data, and a graphic display showing band spectrum and tuning indicator. Using GPL open source software, the modem can be homebrewed for less than $50 parts cost. When coupled with an SSB-capable transceiver or with a popular PSK-xx transceiver board from Small Wonder Labs, you too can have an effective portable PSK31 station.

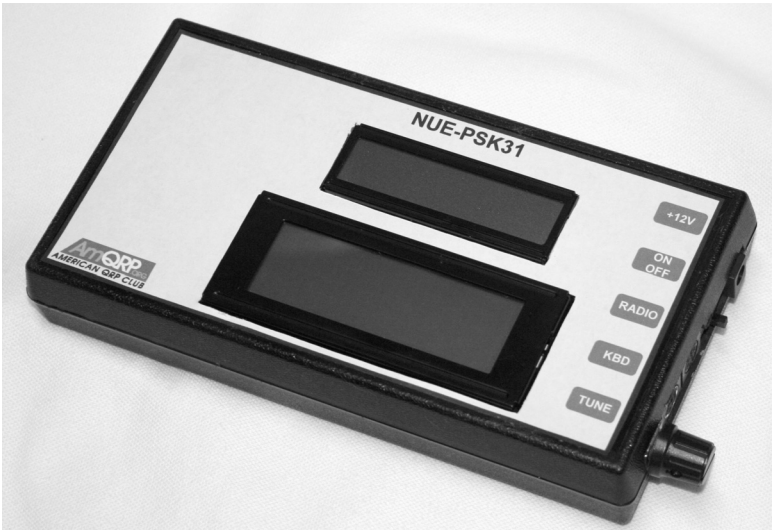


Photo 1: NUE-PSK31 in mating plastic enclosure. Powered by 9V battery contained within (~100ma current draw), with connectors and controls on right side panel.

BACKGROUND

PSK31 was introduced in 1998 to the ham technical community at large in RSGB's RadCom magazine[7]. One was able to get on the air with this digital mode using a dedicated (expensive) DSP card, a crude DOS control program for entering/displaying messages, and interface cables for connection to the

station SSB transceiver. Later, a brilliant PC program was developed (DigiPan[12]) that used a panoramic graphical display to show all signals occurring within a band segment and received messages on the PC screen. This was an astonishing improvement in the user interface for PSK31! Later in 2001, NN1G designed a single board PSK31 transceiver kit (PSK-20[16]) that required no physical tuning, and when used with DigiPan running on the PC it made a quite compact PSK31 station.

But even with these clever hardware and software designs, there is yet room for improvement. The sound card in a laptop or PC is still needed for the intense computation needs of the PSK algorithm. Even if one were to use a modern laptop for that computing power, taking an expensive and delicate computer to the field is a hair-raising experience. It is very hard to see the subtle spectral lines or the screen text data when viewing the laptop's LCD display in the bright sunlight of a mountaintop QSO. So *if* your laptop's battery lasts long enough to enjoy the joys of using PSK out in the open, and *if* you can see the laptop display in the bright sunlight, and *if* you feel like lugging that expensive laptop out into the harsh elements, you could indeed operate PSK31 in the field – but what an ordeal!

The Path to a Design

After operating for years with the challenges of using a laptop in the field, we decided that we wanted a PSK system that did not require the use of a PC in any form. We wanted something that would be very portable and compatible with QRP operations, providing many hours of operation from batteries. The project described here is a result of this desire … but it took a little time for advancing technology to pave the road.

The initial efforts to develop a "portable PSK" controller began about seven years ago with a reproduction of the original G3PLX approach described in RADCOM, but with a more current DSP card providing the horsepower for the PSK31 engine. The design also included a novel Morse user interface and tight coupling to the PSK-20 transceiver. The project was documented in the QRP literature[6] and presented at ham conferences, but ultimately it was too complex and fragile for wide-scale use.

A more recent approach considered was based on the use of low power DDS (direct digital synthesis) chips for generating audio tones with the proper phase modulation. A multiplying DAC was used for modulating and shaping the amplitude of the tones, and a microcontroller was used to demodulate the PSK and display of the resulting characters. Analog filters were used for filtering the PSK signal ahead of digital processing in the microcontroller. The analog filters, however, proved to be too bulky and difficult to design when trying to use standard-value components. Such filters also cannot provide the same level of performance as can be obtained with digital filters. Eventually this approach was also abandoned.

Success At Last

The approach that ultimately proved workable in every regard was one in which all processing is accomplished within a single microcontroller – one that is capable of performing the digital signal processing "number crunching" as well as handling all control chores. The newly-released dsPIC33 microcontroller from Microchip, Inc. is a delightfully-powerful combination of a conventional control processor with a DSP core for intense digital signal processing. Available in a small package with lots of I/O for controlling peripherals, this was just what the doctor ordered!

It was perhaps fortuitous that others in our QRP club were having similar fantasies about the same time. K5JHF was exploring the dsPIC chip family and decided they would make a good basis for a number of projects of interest to the group. He kick-started things with the design of a dsPIC33 project board that included such peripherals as a Programmable Gain Amplifier (PGA), Digital to Analog Converter (DAC), EEPROM memory, Liquid Crystal Display (LCD), quadrature rotary encoder, keyboard and interfaces for a programmer. This was enough to give birth to what we now call the NUE-PSK31 controller.

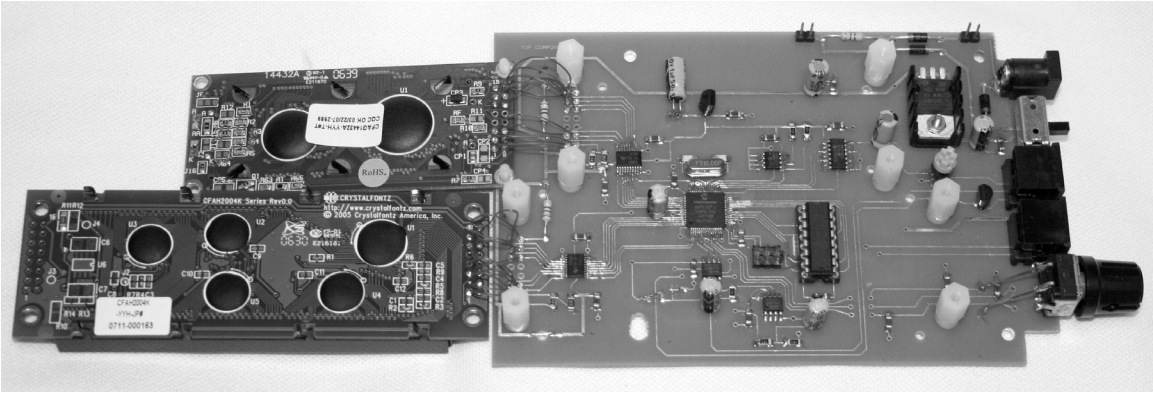


Photo 2a: NUE-PSK31 prototype pc board assembly in bottom clamshell of plastic enclosure. All components are attached to the pc board, allowing other enclosure options for the homebrewer.

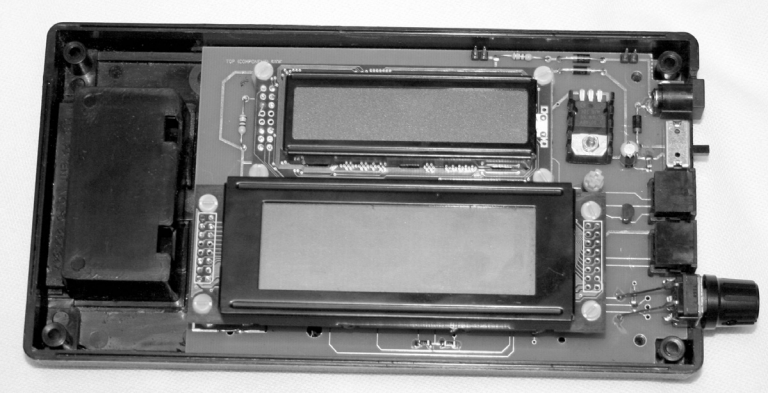


Photo 2b: NUE-PSK31 pc board assembly with graphic and character LCDs flipped over to explose dsPIC, DAC, PGA, EEPROM, and voltage translator ICs. Production LCDs will plug into mating connectors on pc board to facilitate easier assembly.

NUE-PSK31 HARDWARE OVERVIEW

As illustrated in the Appendix A: Schematic, **U1 (dsPIC33F)** is at the heart of the project design. This highly-integrated dsPIC33F device employs a powerful 16-bit architecture that seamlessly integrates the control features of a Microcontroller (MCU) with the computational capabilities of a Digital Signal Processor (DSP). The resulting functionality is ideal for applications that rely on high-speed, repetitive computations, as well as control … just perfect for our PSK31 digital modem controller project!

The dsPIC33F Central Processing Unit (CPU) has extensive mathematical processing capability with its DSP engine, dual 40-bit accumulators, hardware support for division operations, barrel shifter, 17 x 17 multiplier, large array of 16-bit working registers and wide variety of data addressing modes. Flexible

and deterministic interrupt handling, coupled with a powerful array of peripherals, renders the dsPIC33F devices suitable for control applications. Further, Direct Memory Access (DMA) enables overhead-free transfer of data between several peripherals and a dedicated DMA RAM. Reliable, field programmable Flash program memory ensures scalability of applications that use the dsPIC33F family of devices. The specific device we used contains 128 KB of program flash memory, 16 KB of RAM, nine 16-bit timers, 16 general-purpose I/O pins, a pulse width modulation port, a port designed for reading quadrature encoders, two 16-channel ADCs, two UARTS, two SPI ports, two I2C ports, and comes in a 64-pin quad surface mount flat pack package. Whew, this is some chip.

The initial prototype used the dsPIC to capture and decode signals from the **PS2 Keyboard**. This worked fine, except that on rare occasions, the dsPIC appeared to reset itself. This had the unfortunate effect of losing current operating information such as the frequency and callsigns. After reviewing all information regarding the PS2 keyboard, I decided that I didn't like the way we were capturing scancodes from the keyboard. Data was being sent synchronously from the keyboard to the dsPIC, using a clock of only roughly known frequency (~10-20 kHz). Each clock pulse caused an interrupt in the dsPIC, which then sampled the data stream. With the keyboard protocol specified by IBM many years ago, each scancode is sent using 11 clock pulses. In addition, each keystroke press and release results in three-or-more scancodes being generated. Consequently, each keystroke generated a minimum of 33 interrupts.

Apparently too much time was being wasted just processing keyboard interrupts and was the likely cause for occasional dsPIC resets. To solve this problem, I decided to use another small microcontroller to do most of the work handling the keyboard data. This second microcontroller **U5 (Freescale 68MC908QY4)** simply responds to the clock from the keyboard and gathers the bits received into a complete scancode (11 interrupts). Once a scancode is completed, the 'QY4 generates a strobe pulse to the dsPIC. Again, an interrupt in the dsPIC causes the dsPIC to capture an entire scancode on a set of port pins and place it in a buffer, or merely set a flag if the scancode is not a usable character. The ultimate effect of this division of responsibilities is that the dsPIC now responds to only $1/11^{th}$ of the total number of keyboard interrupts that were present in the first attempt.

Two LCD displays are used for the NUE-PSK31 interface: a **character display** LCD-1 and a **graphics display** LCD-2. **LCD-1** is a four line by 20 character display used for received decoded text and as a monitor for text being placed in the transmit queue. Text is displayed when in transmit and as macros are being played back. The cursor changes from steady to flashing when in transmit. Setup Menus are also displayed on the text display. **LCD-2** is a 144x32 pixel graphics display is used to display the FFT-generated spectrum of the audio passband. The lowest six rows of the display are used for the tuning cursor. Since a 512-point FFT is used with a 8 kHz sampling rate, we have 256 points for a 4 kHz passband. I chose to display only the frequency range from 500 Hz to 2500 Hz, using 128 columns of the display. As there are 144 pixels across, the last sixteen columns of the display are not used. These might be used for a vector display in the future. Alternatively, we could use the additional 16 columns to display frequencies to 2750 Hz. The data and control lines for each display are buffered by **level translators U2 and U3.** These are required to match the voltage levels between the 3.3-volt dsPIC and the 5-volt displays.

The **EEPROM U4 (24AA256)** provides local storage for the macro and user-set variables entered during modem operation. This memory device is controlled by one of the SPI ports on the dsPIC.

A computer-adjustable gain stage, the **Programmable Gain Amplifier (U7, MCP6S21)**, brings the low level audio input stream coming from the SSB receiver to the 12-bit analog-to-digital converter on the MCU. Amplifier **U8 (MCP601)** presents precisely one-half the Vdd voltage to the AC reference input of U7.

Processed digital transmit audio tones are converted to a continuous analog stream by the 12-bit **D-to-A converter U6 (MCP4922).** The audio level control R4 is used to adjust the appropriate modulation level to the input of the SSB transmitter, which is generally a one-time setting for a given transmitter being used.

FET transistor **Q1 (2N7000)** buffers the Push-To-Talk control line sent to the transceiver, used to put the radio into transmit mode.

The audio input, audio output and PTT control lines are brought off the pc board using an **8-pin DIN connector J3**. This approach minimizes the number of connectors and cables normally used to connect a digital mode controller to an HF rig, as sometimes these cables can get mixed up and messy at the operating station. Further, when the NUE-PSK31 controller is used with a dedicated HF rig – e.g., a Yeasu FT-817 or a PSK-20 transceiver card – the other end of the cable may also be consolidated to a single multi-pin plug, thus providing a neat and elegant interconnect with the radio.

The design of the **power supply** accommodates multiple needs for this portable project. Since the NUE-PSK31 only requires approximately 200 ma during normal operation, portable power can be handled by using a conventional or heavy duty 9V battery. The digital modem may instead be externally powered by applying 12V power through J1. Diodes D7, D8 and D9 protect the battery when external power is applied. Provisions are also provided through R7 to trickle charge the battery in the event that common NiMH cells area used. In this case, when P5 is jumpered, the modem is conveniently charged over night while connected to a 12V supply on the bench.

The entire NUE-PSK31 project is assembled using a single 2.5" x 3.5" **pc board** – quite an improvement over the Portable PSK project done previously, as well as over the prototype hardware for this current design, as we'll show in the next section. The pc board holds all components – both LCDs, rotary encoder, power connector and radio interface connect – and may be assembled into your favorite homebrewed enclosure. The design was dimensioned for a nicely-sized hand-held plastic case from Pac-Tec, as shown in the photos of the unit. This enclosure also has a conveniently-accessible compartment on the back side that houses the 9V battery.

HARDWARE CONSTRUCTION

Before ending up with a neat and compact pc board, the NUE-PSK31 design started out as a rather large and sprawling prototype hardware layout. This is normally the case when complex projects are born as it allows the designers to try out different approaches and components, while also allowing them to easily get in to monitor and debug the design.

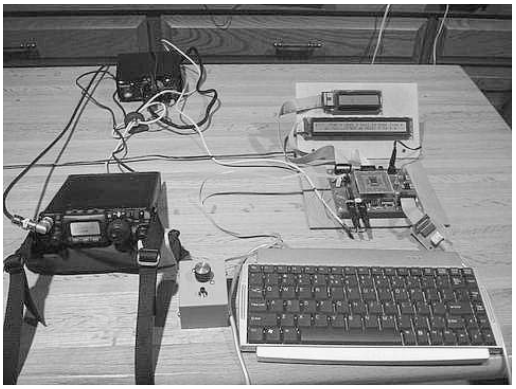


Photo 3: NUE-PSK31 Prototype System

Clockwise from upper right: NUE-SK31 displays and proto hardware, PC keyboard, rotary encoder "dial", FT-817 transceiver, and power supply.

The prototype design was built using a proto-board with plated-through holes on 0.1" centers to facilitate mounting through-hole components. The surface mount dsPIC microcontroller is mounted on a "Schmart-Board"[2]. This particular board is designed to permit attaching 32-to-100 pin SMT devices and has 0.65mm lead separation (pitch). Schmart Boards are available in several pitches and pin counts to accommodate prototyping of a range of SMT controllers. Header pins and sockets are used to connect the board to the main prototype board. The photo below shows how the headers and mating sockets interconnect on the prototype. Point-to-point circuit connections were accomplished using 30-gauge Kynar wire, and a hand-stripping tool was use to strip the ends prior to soldering to the socket/connector pins. Thus the prototype was rather easily assembled and the result was fairly solid when complete.

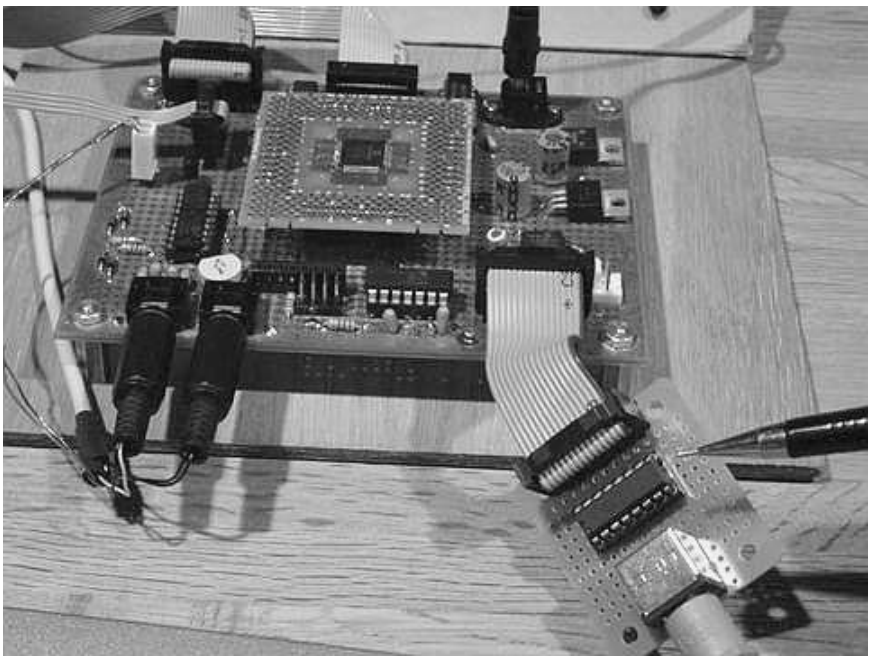


Photo 4: NUE-PSK31 Prototype

Multiple cabling and programmer connection (lower right) allow convenient access to the electronics during design shake down.

Development Tools & Getting Started in Software

While Microchip is well known in the ham community, few of us had experience with this new family of PIC chips. Microchip apparently foresaw this situation and they have provided an amazing number of application notes, specifications and guidance for designers to use in quickly coming up to speed.

Further, even the best chip on earth would be crippled without a good set of software development tools; but Microchip again came to the rescue with a C compiler and an extensive DSP library that proved invaluable to us in developing the project. Both of these were available for free, so what more could we ask!

To program the dsPIC, we discovered that the inexpensive (~$25) PICkit2 programmer[5] is entirely adequate for the job. In-Circuit-Debugging is not readily achieved with the free versions of the tools, and we seemed to manage without it.

The final essential aspect in enabling this project was a design reference for the PSK31 modem algorithm, provided by Moe Wheatley, AE4JY. His PSKcore documentation and C++ source code[3] was professionally done and graciously placed into the public domain, thus available to us. We concluded that it would be a straightforward conversion to C language so we could use our free compiler and have it work on the dsPIC33 … and we relied heavily on it.

PSK MODULATION-DEMODULATION OVERVIEW

We will not go into great depth concerning the theory and operation of PSK31, but one can refer to the reference material[17] and Wheatley's design and technical specification[3] for additional detail. So in this paper we'll first overview the PSK31 encoding scheme, followed by the more demanding decoding scheme.

Note that while the NUE-PSK31 project focuses on the generation and decoding of PSK31, it is generally known that PSK31 is merely one-of-many modulation techniques within the "phase shift keying" family of communication. PSK31 operates at a 31.25 baud (bits/second), while other speeds may be achieved using variations to the software algorithm. PSK is perhaps more accurately termed BPSK, for bi-phase shift keying, whereby two distinct phase states separated by 180 degrees are used to convey the information. Four states may also be encoded/decoded, as is done with QPSK (quad-phase shift keying), in order to provide higher speeds and greater error correction ability.

So even though we will concentrate on the PSK31 mode, understand that some of these other modes can also be achieved with the same hardware and software of NUE-PSK31.

Modulation (PSK31 Encoding)

The PSK31 modulation algorithm is quite straightforward and could even be implemented on a conventional PIC-like device (i.e., one without a DSP core). This was done in several projects over the years within the QRP community; see, for example, the PSK31 Beacon project[18] from the NJQRP Club.

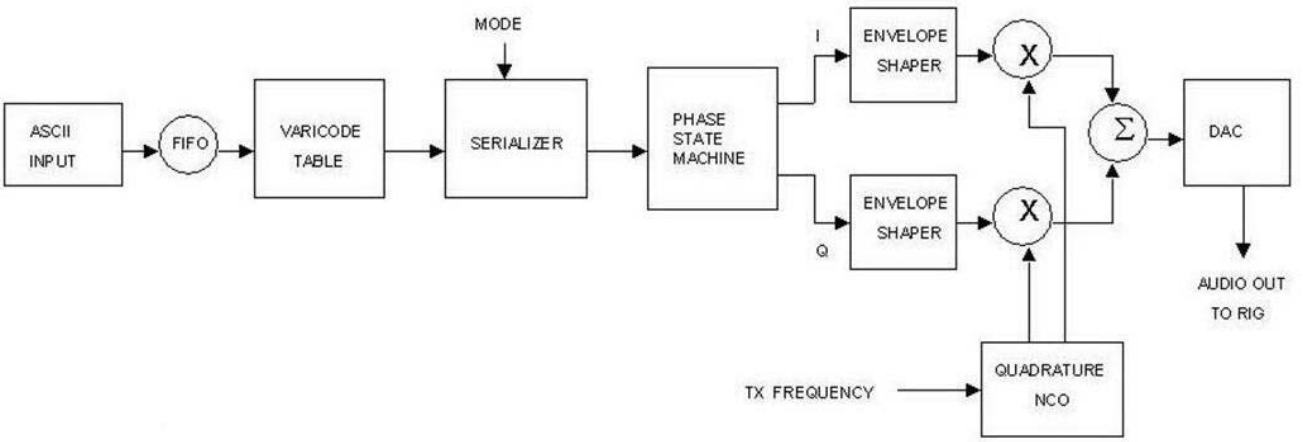


Figure 1: PSK Modulation Block Diagram

The Encoding Steps, in summary:

1) Varicode encoding of the input text character stream coming from the keyboard to create an optimized bit-representation of the text;
2) BPSK serialization of the varicode character to create the proper sequence of phase changes in the waveform based on the bits in the varicode; and
3) Form the wave shape from the combination of phase changes coming from the serializer, being careful to reduce the power level to zero when the 90-degree phase changes occur, thus reducing the bandwidth of the transmitted PSK signal.

These steps are all performed in the dsPIC processor, per the functional block diagram shown next in Figure 2. As ASCII characters are produced by a keyboard, they are first converted to Varicode-encoded characters using a lookup table. A string of binary bits, the length of which is variable (hence-"Varicode"), is generated from the table. The strings of bits are then used to drive a differential phase state machine which uses predefined tables to modulate the amplitude of the quadrature outputs (sine and cosine waveforms) of a numerically-controlled oscillator NCO. The sine and cosine numeric waveforms are derived from a lookup table to produce the NCO carrier.

The two quadrature oscillator signals are multiplied by amplitude functions as determined by the phase state machine, and the resulting channels of data are added to produce a digital version of either a BPSK or QPSK signal. Although a simpler scheme could be used for BPSK alone, this method has the advantage that it can also generate QPSK. This digital stream of data is then sent to a digital-to-analog converter (DAC) to produce an audio carrier with BPSK/QPSK modulation. The output of the DAC is sent to the audio input of the SSB transceiver for conversion to RF.

Demodulation (PSK Decoding)

Whereas the encoding process described above is pretty straightforward, the PSK *decoding algorithm* is significantly more complex and computationally demanding. It is for this reason that there have been so few standalone PSK demodulator projects in the ham homebrewing community. The PC sound card is clearly the easiest way to provide the intense DSP processing needed for decoding PSK – hence PC-based PSK31 programs abound.

This is where the standalone (PC-less) NUE-PSK project excels – it is able to independently handle the complex PSK decoding algorithm in real time, thus providing the first truly portable digital modem for hobby use.

Refer to this PSK Demodulation block diagram as we walk through the Decoding Steps next.

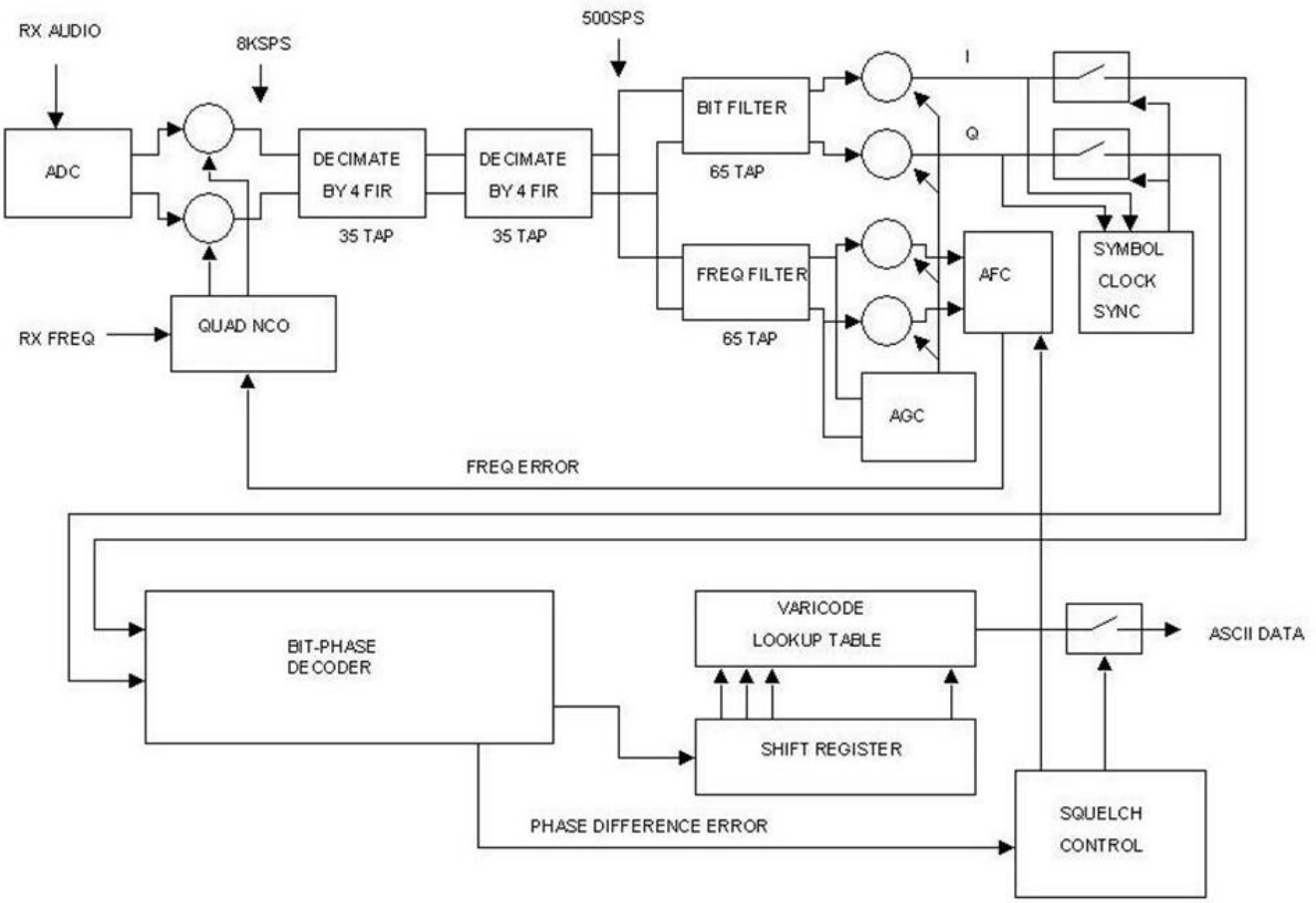


Figure 2: PSK Demodulation Block Diagram

The Decoding Steps, in summary:

1) Receiver audio is sampled at 8 kHz, creating a digital floating point representation of the audio stream.
2) Data is fed into a 512 point FFT for display, tuning and visual signal monitoring purposes.
3) The audio floating point data stream is converted to a baseband signal centered on the user's frequency. The NCO generates sine and cosine frequencies and multiplies them with the audio stream to produce I (in phase) and Q (quadrature phase) data streams.
4) The I & Q data streams are decimated by 16 to reduce the sample rate to 16 times the signal BW. The final sampling rate then is 8000/15 = 500 Hz.
5) A 65-tap "matched bit" FIR filter is applied to produce a magnitude response for best SNR for data extraction, and to minimize Inter-Symbol Interference (ISI) presented in the signal path and in the receive filter.

6) AFC is performed to lock on the incoming signal frequency by using another FIR with BW = 31.25 Hz.
7) AGC is accomplished by computing the average signal magnitude from the I & Q data streams. IIR filters are selected to provide either slow decay or fast attack settings.
8) Frequency error detection is done by scanning the FFT data within the capture range while looking for the nearest peak. A wide range AFC algorithm is also performed by calculating the slope and moving the NCO to place the peak at the center.
9) Symbol synchronization is done by finding the center of each symbol in order to sample at the optimum time. There are 16 samples per symbol at 500 Hz intervals, so each sample energy is IIR-filtered and stored in an array. The array elements with the most energy are selected as the center of the data symbol at each symbol period of 32 ms.
10) Squelching is performed by histogramming the incoming signals and considering the spread (difference angle, or atan (Q/I) between each sample) around 0 degrees and 180 degrees as a measure of signal quality. The narrower the spread, the stronger and more coherent the signal.
11) Symbol decoding is the last step, whereby we convert the I and Q signals back to two possible symbols by using the difference angle (<90 deg = 1, >90 deg = 0). The resultant symbols are shifted into a register until the inter-character mark of 2 or more zeros is found. The shift register is then used as an index into a reverse-Vericode table containing the originally transmitted characters.

SOFTWARE OVERVIEW

Most of the PSK modulator and demodulator code is derived from PSKcore. Although his code was written in C++, and was developed for use on a PC, it was not too difficult to convert it for compilation under C, for which there is a free compiler from Microchip. As part of our QRP group project, John Fisher, K5JHF provided much of the initial software for the project. His code includes most of the initialization code, a keyboard handler, a basic LCD driver, I^2C and SPI drivers, an interface for EEPROM storage, and ADC and DAC interfaces. I was able to develop the remaining code even though I consider myself a novice C programmer.

PSK31 Decoder Processing

Receiver audio from an SSB transceiver is presented to the NUE-PSK31 circuits. It is first processed by PGA whose gain is controlled by the dsPIC via an SPI (Serial Peripheral Interface) connection. The output of the PGA is then sampled by an internal 12-bit ADC on the dsPIC.

Timer1 of the dsPIC provides all critical timing. The timer is set to interrupt the software program every 125 microseconds, at which interval the ADC reads and converts the incoming audio data, corresponding to a sample rate of 8000 samples per second. The ADC samples are captured into a 2048 word buffer, and once the buffer is half full a flag is set to inform the system that data is available for processing. Only half of the buffer is processed at a time – this ping-pong buffering technique allows continuous data processing to be accomplished while the other half is being filled in real time.

The main routine of the program is an endless loop in which a number of flags are tested. When one is found to be set when queried, execution of that specific function is triggered. For example, if the ProcPSK flag is checked and found to be set, a block of data is then processed. Each sample in the buffer is multiplied by a quadrature NCO, producing I and Q signals. Each of these is then passed two

times through a 35-tap decimate-by-4 FIR filter. This provides I and Q signals that are now sampled at 500 samples per second. (If in PSK63 mode, the second filter bank will decimate-by-2, providing 1000 samples per second.) While the block of 1024 samples is being processed, the second half of the buffer is being filled with new samples under control of the Timer1 interrupts. Processing then ping-pongs between the two halves of the buffer. Using this technique we never write new data to the part of the buffer that is being processed.

The next step is to split each of the I and Q channels into two paths. One is for the processing of the bits and one path is for processing of frequency data, producing now four channels of data. Each of these channels is filtered by a 65-tap FIR. The I and Q bit channels should be optimized to minimize intersymbol interference, while the I and Q frequency channels should be optimized for fast response of the Automatic Frequency Control (AFC) loop. All of the FIR filters have responses as specified by AE4JY. Instead of using the PSKcore filtering code, I am using FIR filters from the Microchip DSP library, as these software filters are designed to take into account the special hardware features of the dsPIC. However, the results may be shown to be the same; that is, they have identical frequency responses.

The bit channels are processed as described in the PSKcore specification to determine the proper time for determination of the phase changes that are employed in PSK. Since the bit rate of PSK31 is 31.25Hz, each bit extends for 32 milliseconds in time. We have a sample rate of 500Hz at this stage of processing, so there are 16 samples for each bit. The point in time for proper synchronization of the phase detection process is based on an analysis of the average energy in each of the 16 samples when averaged over several bits. Without going into the mathematical details, suffice it to say that the maximum energy always occurs at the boundary between successive bits. This fact is used to establish synchronization in the bit detector.

I used the free WinFIRDesigner software, with parameters obtained from the AE4JY code to calculate the FIR filter coefficients. As noted above, the frequency responses obtained with these coefficients are identical to those published by AE4JY.

A processing block takes the four filtered signals and proceeds to:

1) obtain a digital AGC control;
2) calculate frequency errors;
3) correct the Numerically Controlled Local Oscillator;
4) determine bit boundaries;
5) determine whether a 1 or a 0 is being received;
6) collect the decoded 1's and 0's into a Varicode pattern;
7) convert the Varicode pattern into ASCII characters; and finally
8) display the resulting characters.

The PSKcore routines were used to perform AGC, bit synchronization, character decoding, etc. In addition, I added code that will perform a 512 point FFT on the samples (8k sampling rate) that are provided to the FIR filters. The processed FFT is then converted to magnitude, and then to a logarithmic scale. The 500-to-2500 Hz portion of the spectrum is displayed on a 144x32 pixel graphics LCD. This display is essential for tuning. More about this later.

The final output of the demodulator processing is a decoded ASCII character. These decoded characters are displayed on a 20-character by 4-line display. The display driver includes a line buffer so that once a

line of characters is filled, it is scrolled up and new characters are inserted at the beginning of the second line. This approach was chosen so that printed characters remain fixed longer for easy reading, as opposed to all characters being in constant motion (scrolling horizontally) when data is being received.

PSK31 Encoder Processing

As mentioned earlier, the encoding process is considerably less-intense as compared to the decoder operations. ASCII characters are accepted from the keyboard, converted to Varicode characters, and the binary string represented by the Varicode is used to modulate the phase and amplitude of an audio carrier – the PSK audio signal.

Although PSKcore code creates a block of data to be sent to the PC's soundcard, I chose to generate a single sample of output signal for each and every 125 microsecond timer interrupt. This minimizes data memory requirements. The method of generating the desired phase and amplitude modulation is that developed by AE4JY with the exception that the tables used reside in program memory instead of data memory. The use of these tables eliminates the time consuming calculation of sine and cosine signal components. The choice of placing these tables in program memory was made because I had plenty of program memory with the dsPIC, but not a lot of spare data memory. The calculated data samples are then scaled for output to a 12-bit DAC. The DAC output, after capacitive coupling, is then routed to the audio input of a SSB transceiver.

As each interrupt occurs the code steps through the tables, providing modulation values for the I and Q signals, resulting in either BPSK or QPSK phase modulation. The modulated I and Q signals are added together prior to the DAC.

USING THE NUE-PSK31 DIGITAL MODEM

To enable easy operation of the NUE-PSK31 digital modem, numerous hot keys (short cuts) have been incorporated into the software. For example, these features make it easy to record macros, and change various set-up features like mode, AFC On/Off, CW Id, etc. The EEPROM memory is used for storage of setup information and macros so that they will be retained when power is removed from the modem. The following is a list of hot keys that have been implemented:

- Play Macros: Function Keys **F1 to F7**
- Record Macros: **Ctrl-Fn** Initiates recording. Enter keystrokes. When finished, Press F9
- Erase Macros: **Alt-Fn** to delete Macro associated with Fn
- **F8** toggles TUNE mode. May be accessed only in RX or TX. (Not in Setup, or Macro Recording)
- **F11** displays the first few bytes stored in EEPROM
- **F12** toggles between RX and TX (again, not in Setup, or Macro Recording)
- **F10** displays the main Setup Screen. (Accessible only in RX mode)
- A numeric selection from the Main Menu, selects a submenu, which is then displayed on the LCD. Another numeric selection activates your selected parameter
- **Ctrl-K** clears the keyboard buffer (in case errors made) before entering callsigns
- **Ctrl-M** saves keyboard entries into a fixed location in EEPROM (for recording your callsign, for use in Macros)

- **Ctrl-T** saves keyboard entries into a RAM location (for recording the other station's callsign—also for use in Macros)
- **Alt-M** enters a control character into a Macro, that when played back, will insert your callsign
- **Alt-T** does the same as Alt-M, but it forces the entry of the other station's recorded callsign into the macro playback
- **Ctrl-F** saves the current frequency into EEPROM so that it can be restored at the next power-up
- **Alt-F** retrieves the saved frequency and makes it the current frequency
- **Ctrl-Tab** displays the current frequency (audio) on the character LCD

A few parameters are selectable by hot keys:

- **Ctrl-A** Enable AFC
- **Alt-A** Disable AFC
- **PgUp** Increase PGA gain
- **PgDn** Decrease PGA gain

- **Ctrl-L** clears the Character LCD
- **Ctrl-B** clears the internal buffers

- **Ctrl-Q** inserts a TX-OFF control character in the TX buffer, or Macro

Tuning

Changing the operating frequency is done by one of two methods. The first is by using the rotary encoder dial to change frequency in either of two increments (25 or 50Hz, selectable from the menus). As the frequency is adjusted, the cursor moves on the Spectral Display LCD. The lock sequence is initiated by aligning the cursor with a signal peak and pressing the pushbutton on the rotary encoder.

The second way of tuning is to use the keyboard. The right and left arrow keys increase and decrease the frequency by either 25 or 50Hz, as selected by a menu option. The up and down arrow keys increase or decrease the frequency by 100 or 250Hz. When the cursor is approximately aligned with the spectral peak of the desired signal, pressing the END key initiates the lock sequence.

Use of the encoder pushbutton or the keyboard END key is optional. A timer has been included so that a lack of frequency change after a certain time interval (presently set for 1 second) causes automatic initiation of the lock sequence.

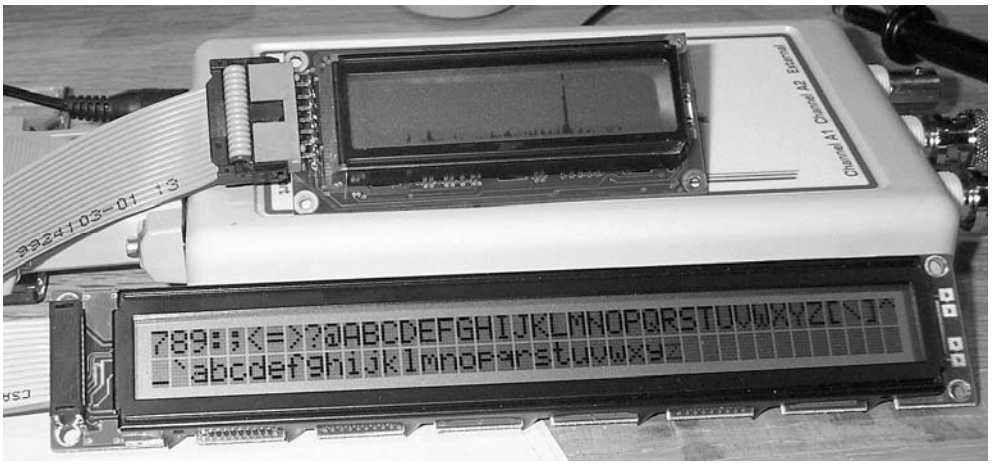


Photo 5: NUE-PSK31 Displays

Graphics LCD (upper) showing received spectrum with a PSK31 signal prominently displayed, and text display (lower) showing the range of characters supported. (NOTE: The NUE-PSK31 supports either the 2x40 character display shown here, or the 4x20 character LCD shown in the enclosure.)

FUTURE ENHANCEMENTS

Consideration of software improvements and additional capabilities are already in the works.

1. A radar-like vector scope would be a very convenient tuning aid to help the user to see the current spread of the PSK31 tones and thereby have an indication of the incoming signal quality. This vector scope can fit nicely on the right side of the graphics display.

2. Depending on the cost of using a larger graphics display, it would make sense to eliminate the character display and also use the graphics display to show the characters from the typing buffer, transmit buffer and receiver buffer. The ability to display stable text (for easy reading) along the bottom of the single graphic LCD will be another consideration for making this change.

3. Updating the graphics LCD to display current spectral information consumes a considerable portion of the total processing time in the dsPIC. If all LCD display routines were off-loaded to a small microcontroller, which we already have in the system, there would be more time available for processing PSK63 and perhaps other digital modes.

4. Additional dynamic range would be possible if an external ADC with 16 to 24 bits were to be employed. The Austin QRP group is currently evaluating ADCs and codecs that might be used in this application.

CONCLUSIONS

The NUE-PSK31 digital modem is used regularly by both authors, has seen some intense operation during some QRP contests this year, and was recently demonstrated at the Austin Summerfest Convention. In short, it is a great pleasure to be able to use the PSK31 digital mode out in the field with such little reliance on a conventional PC or laptop. All the benefits previously mentioned are truly helping in enjoying the overall communications experience.

The project is easy to homebrew, with all software being freely available (under the GPL) and with the use of commonly-available components. To ease the effort of implementing the NUE-PSK31 digital modem, the American QRP Club has graciously agreed to kit up the parts, pcb and enclosure for a very reasonable price[1].

We are indebted to the pioneering efforts of others before us here in the field of PSK31 (see the extended Reference section), and especially to AE4JY. Without Moe's groundbreaking work in creating and documenting PSKcore, the process of getting the PSK31 algorithms working in such a revolutionary microcontroller as the dsPIC would have been much longer in coming.

We hope you enjoy building up your own NUE-PSK31 digital modem. Let us know how it works out and we'll be looking for you on the air!

REFERENCES

1. The NUE-PSK31 digital modem controller is available as a kit from the American QRP Club. All project technical information, kit pricing and availability can be found at http://www.amqrp.org/kits/nue-psk31.

2. SchmartBoards www.schmartboard.com See part 202-0011-01 (32-100 pin QFP, 0.5mm)

3. Moe Wheatley, AE4JY, "PSKCore" www.qsl.net/ae4jy/ . For his source code, download PSKCoresrc.zip. For the full technical specification, download "PSKCore Interface Specification and Technical Description Ver 1.40"

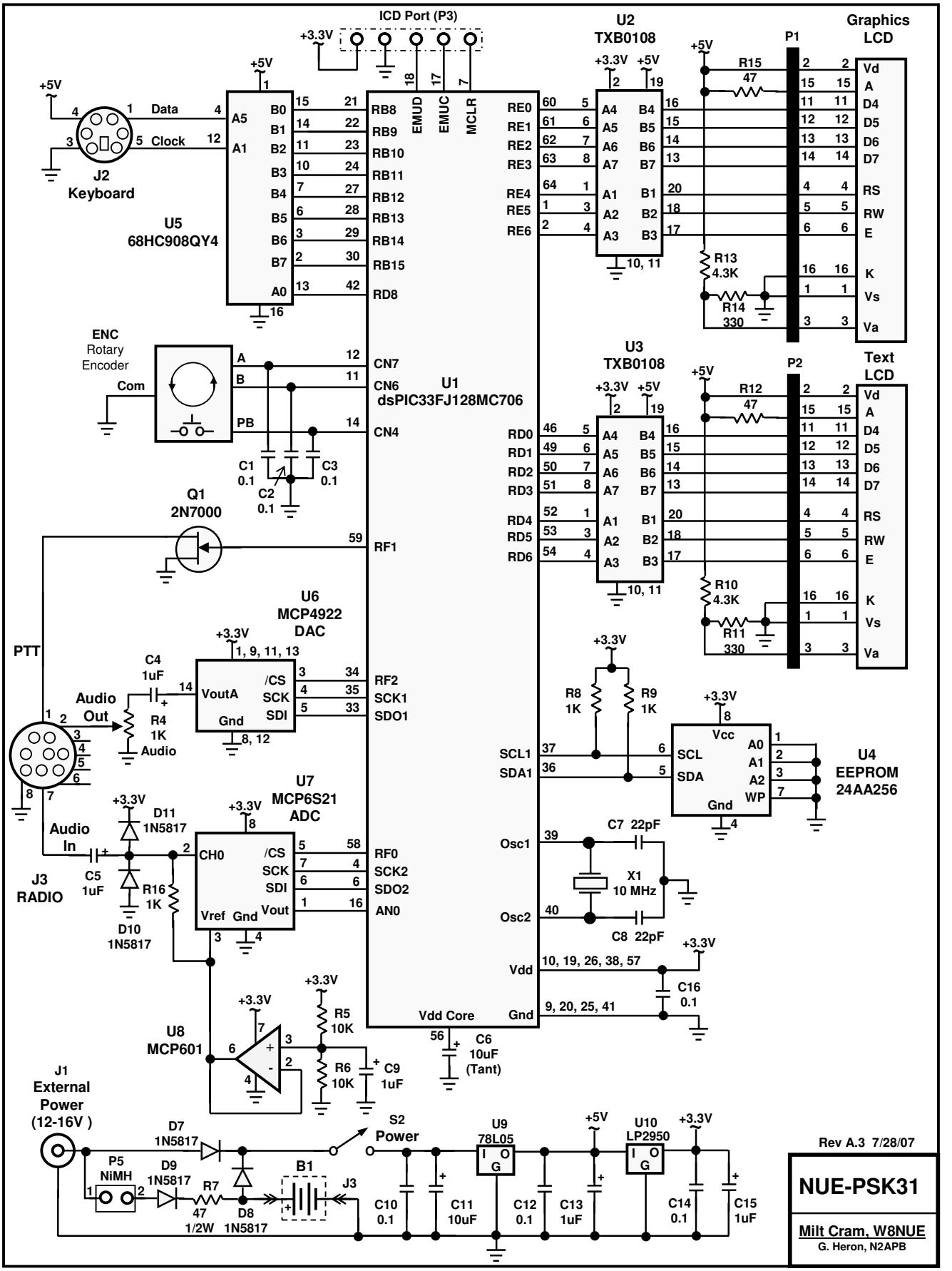
4. Microchip www.microchip.com. Technical documentation for the entire line of Microchip microntrollers is available. The MPLAB Integrated Development Environment, and Student Edition C compiler are available for free download.

5. PICkit2 – A low-cost development kit for programming many of Microchip's dsPIC33 Flash memory microcontrollers. http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en023805

6. George Heron, N2APB, "Portable PSK" project http://www.njqrp.org/portablepsk/
7. Peter Martinez, G3PLX, "PSK31: A New Radio-Teletype Mode", QEX, July/August 1999, pp 3-9 (reprinted from RadCom, Dec 1998 and Jan 1999 issues).
8. Don Urbytes, W8LGV, "A PSK31 Tuning Aid," QST, Dec 1999, pp 35-37.
9. Steve Ford, WB8IMY, "PSK31 - Has RTTY's Replacement Arrived?", QST, May 1999, pp 41-44.
10. Steve Ford, WB8IMY, "PSK31 2000," QST May 2000, p 42.
11. Howard "Skip" Teller, KH6TY, and Dave Benson, K1SWL, "A Panoramic Transceiving System for PSK31," QST, June 2000, pp 31-37.
12. DigiPan software, v1.2 is available at http://members.home.com/hteller/digipan. DigiPan stands for "Digital Panoramic Tuning" and brings the ease and simplicity of panoramic reception and

transmission to PSK31 operation. DigiPan provides a panoramic display of the frequency spectrum in the form of an active dial scale extending the full width of the computer screen. Depending upon the transceiver IF bandwidth, it is possible to "see" as many as 40-to-80 PSK31 stations at one time. DigiPan was developed as freeware by Howard (Skip) Teller, KH6TY/4 and Nick Fedoseev, UT2UZ.

13. Dave Benson, K1SWL, "The NJ Warbler - a PSK-80 Single Board Transceiver for PSK31," QRP Homebrewer, Summer 2000, pp 15-21.
14. Johan Forrer, KC7WW, "Using the Motorola DSP56002EVM for Amateur Radio DSP Projects," QEX, Aug 1995, pp 14-20.
15. ARRL website collection of PSK31 articles, links, literature and products: http://www.arrl.org/tis/info/psk31.html
16. PSK-20 Transceiver Kit for PSK31, Small Wonder Labs, Dave Benson, K1SWL, 80 East Robbins Ave, Newington, CT 06111. Email: dave@smallwonderlabs.com, website at: http://www.smallwonderlabs.com/
17. The "Official" Homepage of PSK31 is at http://aintel.bi.ehu.es/psk31.html
18. PSK31 Audio Beacon, George Heron / NJQRP Club, http://www.njqrp.org/psk31beacon/psk31beacon.html


ICD Port (P3)
+3.3V
EMUD
EMUC
MCLR
U2
TXB0108
Graphics
LCD
P1
+5V
R15
47
Vd
A
D4
D5
D6
D7
RS
RW
E
K
Vs
Va
J2
Keyboard
Data
Clock
U5
68HC908QY4
R13
4.3K
R14
330
ENC
Rotary
Encoder
Com
U1
dsPIC33FJ128MC706
C1
0.1
C2
0.1
C3
0.1
U3
TXB0108
Text
LCD
P2
R12
47
R10
4.3K
R11
330
Q1
2N7000
PTT
U6
MCP4922
DAC
C4
1uF
Audio
Out
R4
1K
Audio
U4
EEPROM
24AA256
R8
1K
R9
1K
U7
MCP6S21
ADC
Audio
In
J3
RADIO
C5
1uF
D11
1N5817
D10
1N5817
R16
1K
C7 22pF
X1
10 MHz
C8 22pF
C16
0.1
U8
MCP601
R5
10K
R6
10K
C9
1uF
C6
10uF
(Tant)
J1
External
Power
(12-16V)
D7
1N5817
S2
Power
U9
78L05
U10
LP2950
P5
NiMH
D9
1N5817
R7
47
1/2W
D8
1N5817
B1
J3
C10
0.1
C11
10uF
C12
0.1
C13
1uF
C14
0.1
C15
1uF
Rev A.3 7/28/07
NUE-PSK31
Milt Cram, W8NUE
G. Heron, N2APB

PARTS LIST

Designator	Part	Qty	Description
U1	dsPIC33FJ128MC706	1	Microchip DSC, 64-pin QFP
U5	MC68HC908QY4	1	Freescale microcontroller, DIP-16
U7	MCP6S21	1	Programmable Gain Amplifier
U8	MCP601	1	Op Amp
U6	MCP4922	1	Dual-DAC MCP4922
U4	24AA256	1	Microchip EEPROM
Q1	2N7000	1	NFET
U2, U3	TXB0108	2	Octal Level Shifting Buffer
U10	LP2950	1	3.3V regulator (TO-92)
U9	78L05	1	5V regulator (TO-92)
LCD-1	4x20 character LCD	1	Primary character display
LCD-2	144 X 32 graphics LCD	1	Crystalfontz CFAG14432A-YYH-TT
P1, P2	16 pin pinheader	2	Pinheader, 2x8 (cut 4 from 2x36 strip)
P5	2 pin header	1	Pinheader, 2x1 (cut 36 from 2x36 strip)
P4	8-pin Mini-DIN plug	1	8-pin Mini-DIN plug
J2	6-pin Mini-DIN	1	6-pin Mini-DIN
J3	8-pin Mini-DIN	1	8-pin Mini-DIN
J1	DC power connector	1	DC power connector
X1	crystal	1	crystal
R8, R9, R16	1k ohm resistors	3	resistor
R5, R6	10k ohm resistors	2	resistor
R4	1k ohm potentiometer	1	resistor
R7, R12, R15	47, 1/2W	3	resistor
R11, R14	330 ohm	2	resistor
R10, R13	4.3K ohm	2	resistor
C6	10uF Tantalum	1	capacitor
C9, C13, C15,	1uF ceramic	3	capacitor
C1, C2, C3, C10, C12, C14, C16	0.1uF ceramic	11	capacitor
C7, C8	22pF	2	capacitor
	IC socket	1	16-pin DIP IC socket
D7, D8, D9	schottky diode	3	Diode, Schottky, 1N5817, DO-41
D10, D11	schottky diode	2	Diode, Schottky, 1N5711, SMT
ENC	Rotary Encoder /w PB	1	rotary encoder
	PCB	1	
	Overlay Label	1	
	Enclosure	1	