Two PIC-Based Keyers

Andy Palm N1KSN

My previous article on PIC chips showed how to build a ten minute countdown timer as a reminder to ID on the air. If you like CW operation or want to learn how to send Morse code, one of these projects may be just the thing for you.

There are four other files that accompany this article, two schematics in gif files and two text files with c code listings. I used the same PIC programmer and PICC-Lite software as I used for the ten minute timer project, so I won't go over these again.

The "tiny keyer" is built with a PIC12F675 chip, which has 8 pins. If you look at the c code file for this you will see that the program is pretty simple in outline. There are two subroutines, one to wait a given number of milliseconds, the other to handle a straight key. If on powering up the PIC senses that a mono plug is in the key jack (or the dah contact is grounded), then the program stays in the straight key subroutine.

The straight key subroutine uses a "state machine" to debounce the key contacts. I don't know if this is really necessary, but I did it anyway. You might want to try it without this complication and see if it works. The debouncing limits the speed of straight key sending, but not enough to be practically important. This would only be a problem if for some reason you plugged another keyer into this keyer and sent fast code.

If a straight key is not detected, then the program goes into its main loop. This loop looks to see if a dit or dah paddle contact is closed and sets a flag if this is so. If a dit or dah flag is set, then a dit or a dah is sent. The use of the flags (called "buffers" in the code) is not an unnecessary complication. If you think so, try not using them and see what happens.

You will note that the paddle contacts are each polled three times during one execution of the loop, twice together and once each separately. The separate polls were added by trial and error to make sending smoother. One thing I gained while fooling around trying to get smooth operation is some real respect for those who have created smooth commercial keyers like the CMOS IV. It's hard to describe "smooth," but you know it when you feel it.

How do you set length of the dits and dahs for a desired words per minute (WPM)? Well, the "standard" word for Morse code is "Paris" including the seven element word space. A dit is made up of one element and a dah is three elements long. The space between dits or dahs within a character is one element long. (The space between characters is three elements long, but that isn't used for these simple keyers.) "Paris" with the seven element between-word space is 50 elements, so

Element length in milliseconds = 1200 / WPM

The "tiny keyer" has a pushbutton. If this button is held down when a dit is made, the speed is increased by approximately 1 WPM. If the button is held down when a dah is made, the WPM decreases by one. This is a primitive but effective speed control. If you look at the code you will see expressions that keep the speed between 15 and 30 WPM. I said "approximately," because the math is done in 8-bit integer arithmetic, and the quotient above will not come out exactly. But it's close enough.

The other keyer is a combination of keyer and code practice or demonstration oscillator. It has two features beyond the tiny keyer, a speed control pot and a sidetone generator (and amplifier). I used the PIC16F684 chip for this keyer because it has a pulse width modulator that makes it easy to produce a squarewave sidetone. The sidetone is keyed along with the output to the rig, but it is done by changing the "tristate" register value of the sidetone pin. This switches the sidetone pin from output to input and the PWM can be left on all the time, simplifying the code.

The speed pot is read by the analog to digital converter module (ADC) on the chip. (By the way, if you want to build the tiny keyer with a pot speed control, you can because the PIC12F675 has an ADC module. But this chip does not have a PWM module.) There is a state machine at the bottom of the main program loop that starts and reads the results of the ADC. When the ADC result is obtained, it is just a number between 0 and 255, so an equation is needed to convert it to a WPM value. If you look at the code involved, you will see that there is a factor of 1020 used. This is there to allow more resolution in the pot setting under the limitations of 8-bit integer arithmetic.

This keyer has two pots, one for the sending speed and the other for the audio output level. I had two pots with on/off switches on them, so I set things up with the master on/off switch on the speed pot and the sidetone on/off on the volume pot. The sidetone was set at 600 Hz, but you can change it in the software if you like a different tone. Even though I normally use a 400 Hz sidetone (it's easy on the ears), I set this keyer up for 600 Hz because I have a resonant CW speaker for 600 Hz that can be used for code demos.

Since I had the parts, I also added an audio low pass filter after the amplifier. It is a pi filter with three elements and was originally designed for a little QRP rig. This filter cuts down the harshness of the squarewave from the PIC chip. After I built the keyer I wondered if I should have put the filter before the amp, since at higher volumes the output starts to distort to a triangle wave. You might want to experiment with this if you build this keyer.

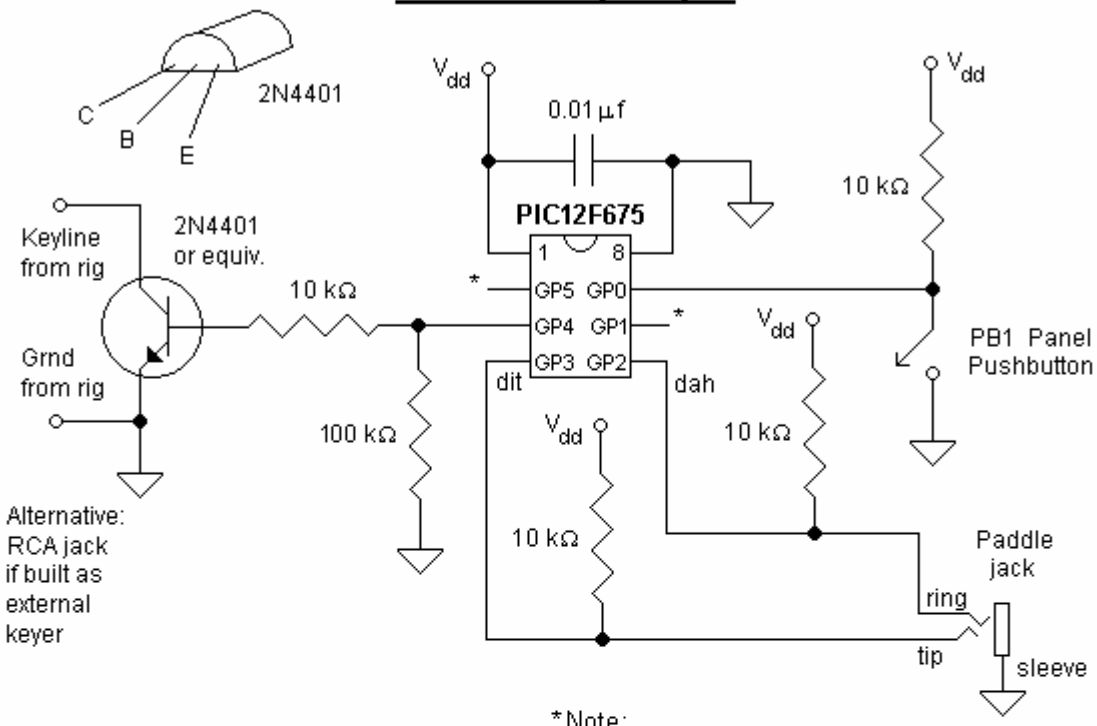
If you look carefully at the c source code for both keyers you will note that they are very similar. Except for the different features used, the only differences are in the names of some of the special registers. It was really easy to "port" the c code from one chip to the other. This is a nice feature of the PIC line of chips.

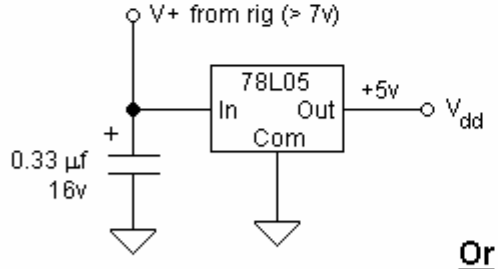
I am happy with the performance of these keyers. The tiny keyer can be a stand-alone keyer, but I designed it to be an add-in for simple QRP rigs. I will be adding it to a small QRPp 80m rig, using a 78L05 regulator to provide 5v to the PIC chip. The code practice /demo keyer is housed in a Hammond plastic case that has a special space for a 9 volt battery and an access door to it. Although these PIC chips run on 2.2 to 5.5 volts, I needed more voltage for the LM386 sidetone amplifier chip, hence the 9v battery. This circuit also uses a 78L05 to give the PIC chip 5 volts.

Acknowledgements:

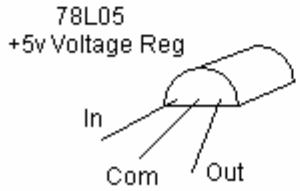
The sections of code using the timer (in the wait subroutine), the PWM, and the ADC were taken from the "Evil Genius" book I told you of in the ten minute timer article. The ideas for the straight key feature and the use of buffers when reading the paddle contacts were taken from some BASIC code I found on the internet, identified as "Keyer Version 1.0.0, Goody K3NG." However, I claim as my own the debouncing state machine for the straight key and the number and arrangement of the paddle contact reads in the main loop. I also claim as my own any problems or errors in the schematics.

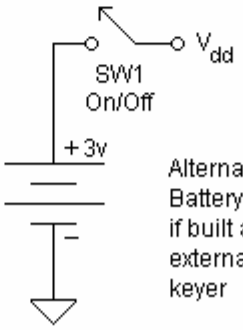
N1KSN Tiny Keyer





Or

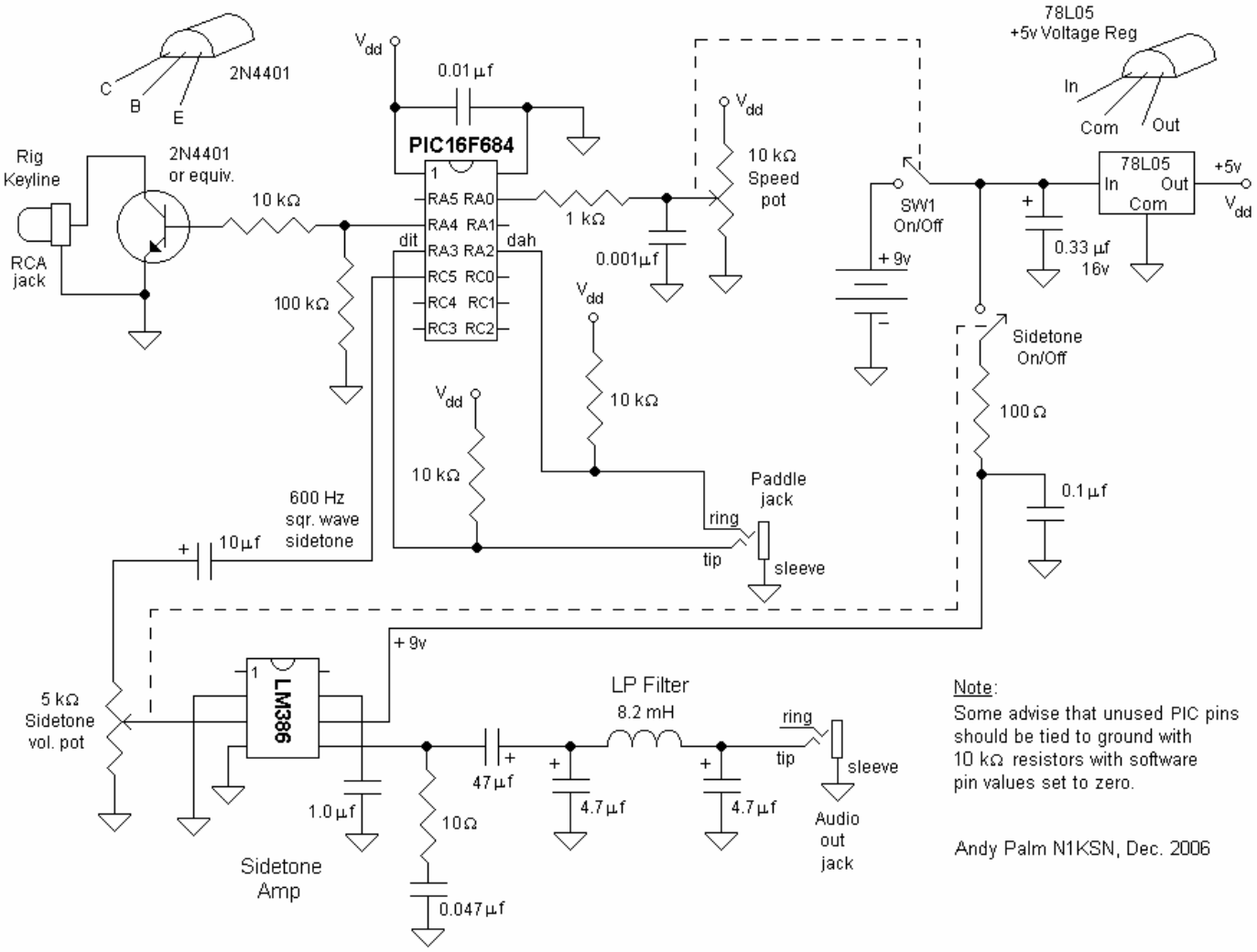




*Note:
Some advise that unused PIC pins should be tied to ground with 10 kΩ resistors with software pin values set to zero.

Andy Palm N1KSN, Dec. 2006

N1KSN Simple Keyer and Code Practice/Demo Oscillator



Note:
Some advise that unused PIC pins should be tied to ground with 10 kΩ resistors with software pin values set to zero.

Andy Palm N1KSN, Dec. 2006