DesignerNotes II AA0ZZ 11/16/2009 (updated 10/7/2013)

In 2009 the Four States QRP Group asked me if I would be willing to develop a PIC-based CW keyer that they could sell as a fund raiser for OzarkCon. I wanted to do a keyer for a long time - to implement one with exactly the features I wanted - but never got around to it. This was a good excuse for doing it, and a good cause, so I agreed.

I developed the code and made a printed circuit board for them and we have a completed product now. The Four States QRP Group announced its availability in November, 2009 and many EZKeyer kits have been successfully built and used.

I thought some of you might be interested in hearing about the development process. I developed this CW keyer entirely on my PIC-EL III board. In addition, I'm programming all of the production PICs on my PIC-EL board. It is working great.

From the very beginning I really wanted to have three pushbuttons for three messages. From my contesting experience I think three messages will handle the basic exchanges without the need to reprogram the messages during a contest. (Typically, I like to have my call in message 1, exchange (599 X - or whatever) in message 2 and CQ message as message 3.) I also wanted to put the PIC into SLEEP mode when the keyer was not operating, eliminating the need for a power switch. These two issues were closely intertwined. Here's why. In order to use SLEEP mode, you obviously need to have some trigger for waking it up. The PIC's port-monitoring feature is the obvious method to use, where any activity on these ports would wake up the PIC. I wanted to make it wake up on a paddle press or a pushbutton press. However, there are only four port pins (RB4-RB7) that the PIC looks at for this wake-up functionality. Two were obviously reserved for the paddles, so I had just two port pins for three pushbuttons. I resolved the issue with a little "trickery" in the way I implemented the three pushbuttons. Two of them are configured as normal pushbuttons on PIC ports but the third pushbutton is implemented with a couple of diodes. Here, pushing the third pushbutton activates the ports for BOTH Pushbutton 1 AND Pushbutton 2. Now I just had to implement code to watch for both pushbuttons simultaneously as the trigger for sending message number 3. (See schematic for details.) Yes, it works great. This software could have been cumbersome to handle but, in this case, it was a function that was not difficult to handle cleanly.

I decided to run the keyer with the PIC's internal 4 MHz oscillator. It is not as accurate as a crystal but, for the purposes of a keyer, it is very sufficient. This reduced the overall cost of the keyer, since a crystal would have been one of the more expensive parts. Yes this can still be done on the PIC-EL with the crystal connected. You just configure the internal oscillator (with a different CONFIG statement) and the PIC simply ignores the crystal connection.

I decided to use the same basic command structure that Steve Elliott, K1EL, used in his K8 keyer and that we ported (with his permission) to the PIC-EL. I started out with the standard PIC-EL hardware connections (paddles, speaker, keyed output) that we used in the K8-ported keyer. This configuration had some challenges but was sufficient to get the basic keyer mechanism working.

The main problem was the port connections for the pushbuttons. Looking at the PIC-EL schematic you will see that two of the pushbuttons share ports with the paddles and the third pushbutton shares a port with the speaker. I solved the problem by disabling the Master Clear function (another CONFIG change) so now I was free to use the Reset pushbutton (PIC-EL PB4) port as a dedicated pushbutton port. I configured the Reset pushbutton to be Pushbutton 1 in the keyer code and I simply did not have a functional Pushbutton 2 or Pushbutton 3 while running on the PIC-EL. To test the Pushbutton 2 and Pushbutton 3 functions on the PIC-EL I made use of "stubs" - short temporary pieces of code to force execution of areas of the code that I couldn't get to "naturally". This was sufficient to debug the code.

In the end, when I had the hardware PCB, I ended up putting two #DEFINE statements in the source code for "PICEL" and "STANDALONE". This was done so I could move the paddles and the three pushbuttons to RB4-RB7 so I could use them for the wake-ups. Then I could simply change one line in the source code, re-assemble it, and change from one configuration to the other. Two different sets of PORT definitions are coded and, with #ifdef and #ifndef statements around the affected code segments, I could turn on the right code at the right time. It works like a charm.

I must share one more trick that I used in developing the CW keyer. Obviously, in any keyer the timing is critical. I had to figure out how to make the speaker operate at the desired sidetone frequency of 600 Hz. This is a bit lower than the 800 Hz tone that Steve implemented in the K8 but 600 Hz is often thought to be optimum for hams as they get older and have more trouble hearing high frequency tones. I designed timing loops that toggled the speaker port with the right timing to make the 600 Hz sidetone.

How did I check it? Two ways. First I loaded the code into a PIC simulator. I really like the PIC simulator by OshonSoft (www.oshonsoft.com). I put breakpoints in the code at appropriate locations and counted the instructions for one complete tone cycle. Knowing that the time to execute one instruction was 1 uS I was able to adjust the constants which control the instructions in the timing loops. Then, in the end, I checked it with my oscilloscope and it was right on.

I did some math, calculating exactly what the theoretical timing should be for ideal Morse code. The three-to-one DAH-to-DIT ratio was implemented and the time between elements was also set to be one DIT-time. (For fun, try figuring out the number of DIT-times there are in the word PARIS. It actually contains exactly 50 of the basic "DIT elements".) I built up constants for loop routines that take exactly one DIT-time to execute, and constructed loops in such a way that they would work when the sidetone was on or off. The 600-Hz tone cycle is a basic building block of these timing loops. Then I made a routine for a "DAH element". It was constructed by simply calling the DIT-time loop three times. You can see that it gets a bit complicated and confusing so I kept careful notes (as comments in the code symbolics) to keep my bearings. I set up the basic timing cycles to have perfect timing with the keyer running at 25 words per minute. Then I simply used proportions to make the constants for the other speeds. I checked the timing with the simulator and verified it on the oscilloscope. All written in one sentence, but it took a long time to actually do.

I need to explain the RX Mute function a bit. This was a request on the "wish list" from the folks at Four States QRP Group and I was initially quite skeptical of the usage. I know that homebrew rigs have widely disparate requirements for switching time so I knew that implementing one with a fixed delay would be somewhat limited. Usually this type of function is implemented in the hardware. After extended discussions and analysis I agreed to implement a simple version - with a fixed delay of 7 mS on both the "make" and "break" sides. I am still not 100% comfortable with it, since I know it changes the basic "weighting" of the DITs and DAHs relative to the spacing and thus does not produce "mathematically correct" CW. It becomes more objectionable at higher CW speeds. Nevertheless, it is only an option and users are free to use it or to leave it off. It's a rather unusual function and is there for experimentation.

There you have it. A neat, fully functional keyer, all developed on the PIC-EL. It was a real pleasure to implement it this way.

In case you are wondering, I am not going to be publishing the source code for this keyer for a while. This is by agreement with the Four States QRP Group - to protect their fund-raising investment. I expect to release it eventually. Code protection is also turned on in the production PICs - so it can't be downloaded and copied. This is different than anything I have done in the past but I hope you understand the reasoning.

UPDATE 2013: Now, it's October, 2013 and EZKeyer II is becoming available. There are a couple of new features added to the code (message repeat and bug-emulation) but the main difference is the new, custom enclosure. Some builders had difficulty in the assembly of the EZKeyer in an Altoids tin. Some had difficulty in punching the holes and some had problems in applying the proper amount of super glue to hold the pushbuttons in place.

At OzarkCon 2013, Dave Cripe, NM0S, showed me his new Cyclone-40 transceiver in a custom PC board enclosure. I realized that a custom enclosure like this could be a great way to solve the EZKeyer build problems and immediately started working on a similar case. Now, I have it. The enclosure provides a method of mounting the PC board with its 3 pushbuttons and 3 jacks in the enclosure without any wires except for the battery. The silkscreening of the labels of the buttons and jacks as well as the command list on the bottom makes it a very attractive enclosure.

I would be glad to answer any questions you might have.

Best Regards,

-Craig, AA0ZZ

10/7/2013