made from aluminum provides ventilation, which is a good practice. When wiring the AC components I used heat shrink tubing to keep those connections as well insulated as possible. Safety first. Every panel in the cabinet is removable so that access is easily available to all internally mounted parts.

I found that this supply has given me a great deal of utility when building QRP projects. The size is perfect for my bench and the weight of the cabinet gives it some heft so it isn't pulled around easily. I have been able to easily determine lower limit voltages for circuits to see at what point a given project will fail to work. That is great for field radios. The wide variability of voltages permits the emulation of any battery voltage as I prefer not using them on the bench. The build was easy, so give it a try!

●●

Open Source Development with the TEN-TEC Rebel

James Lynes—KE4MIQ jmlynesjr@gmail.com

In my last article [1], I introduced you to the microcontroller used as the basis for the Ten-Tec Rebel 506 Open Source 20/40 meter transceiver. The Rebel is now shipping and the user community [2] is already actively designing software and hardware additions. I thought it might be fun to follow the progress of this new style of product development.

As shipped from TEN-TEC, the Rebel comes loaded with a software package that implements the basic functions of a 20/40 meter CW Ham transceiver. The Open Source design philosophy of the Rebel encourages user experimentation and modifications to the hardware and software. Several individuals and groups are putting this philosophy into practice. One ad-hoc group with a great name, "The Rebel Alliance" (Paul NA8E, James K4JK, Johan PA3ANG, et. al.), has established a github repository [3] for the collection of additions and changes being made by several contributors. Some of these additions and those made by others may find their way back into the official TEN-TEC release code in the future. But, until then, each Rebel owner that has a basic software understanding or the willingness to learn, has the capability to develop their own features or pick and choose what The Rebel Alliance features to include in their own customized version of the Rebel code today.

On the hardware front, (Paul NA8E) has contributed a design for a band switching relay board (Figures 1 & 2 [4]). As delivered from TEN-TEC, band switching on the Rebel is accomplished by moving several jumpers on the mother-board. This board plugs into the jumper pins that would normally be moved manually and accomplishes band switching without having to go inside the Rebel. Jack, W9NMT/8 and Dennis, W6DQ are developing a CW decoder board based on the LM567 tone decoder chip (currently working up to 18 wpm). I am working on a shield (daughter board, discussed below) which will use a Fubarino Mini as an I/O Expansion Processor connecting back to the main chipKIT UNO32 board via an I^2C interface. Finally, the Rebel can now support several types of LCD displays: 20x4 and 16x2 parallel displays, 20x4 and 16x2 I^2C displays and the Nokia 5110 graphic display. Figure 3 [4] shows an example 20x4 display. The Rebel as currently shipped does not support a display mounted internally. However, Ten-Tec is currently designing several optional top covers that will support adding LCD displays and stacking multiple shields. Other developers are also looking to expand their hardware by the use of cables to external cabinets.

The Rebel Alliance has also developed/integrated a number of very useful software features. As of the time of this writing (November 2013), they are at revision 1.1 and have included: Support for LCD displays; an Iambic A/B keyer; a Beacon mode; basic Kenwood format CAT control; code to support a prototype band switching PCB, frequency announcement via side tone, and an expanded USB serial debug packet.

I can't wait to see what else is in store. For example, a discussion of support for digital modes is underway.

My own contribution, the I/O Expansion Processor, came about as I started thinking about adding some additional peripherals. When I saw the Rebel

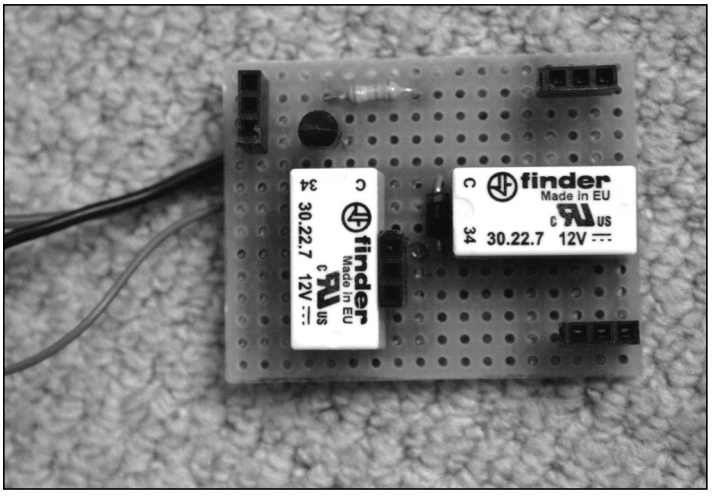

Figure 1—Band switching relay board.

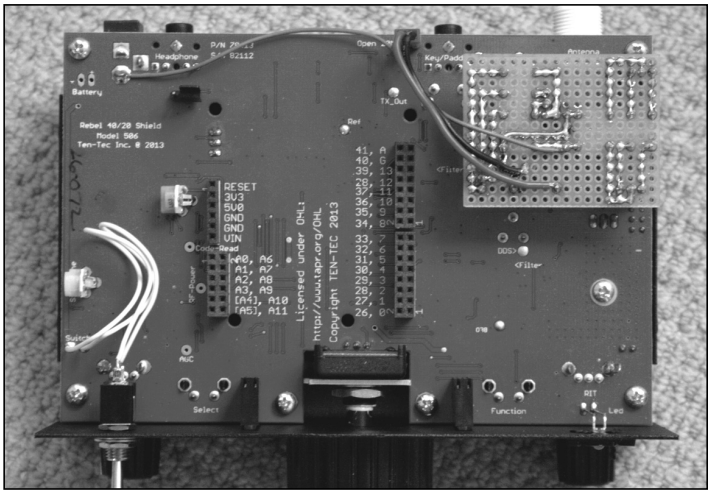

Figure 2—Band switching board installed in the Rebel.

Figure 3—Example outboard frequency display.

announcement, I joined the Yahoo group and downloaded the schematic and code. The 42 I/O pins of the chipKIT UNO32 seem like a lot, but when you dig into the gory details you see that most of these pins are used for the basic operation of the transceiver. Therefore, if you choose to add a parallel LCD display, there won't be many pins left. The other issue that you have to consider is that while the UNO32 supports many internal peripherals, the low package pin count dictates that pins are typically shared by several functions. As a pin is assigned to a specific function, other functions supported by that pin are no longer available. To me, the pin limits on the base UNO32 will very soon limit the enhancements that can be deployed with just the single UNO32. So, as in real estate, if you can't build out, build up.

As luck would have it, the pins for one I^2C peripheral are available. I^2C is a fast (100-400 kbps) serial interface that only requires two pins—serial clock (pin19 SCL1) and serial data (pin18 SDA1)—and two pull-up resistors. Multiple I^2C devices can reside on this bus, each answering to their own address. MPIDE [*Multi-Platform Interactive Development Environment*, an open source software development environment specifically made for Arduino —*Ed.*] ships with an I^2C library that makes using these devices easy.

This has led me to the development of an I/O expansion architecture based on the available I^2C bus. I began by breadboarding my Fubarino Mini and setting up a simple pot voltage divider circuit. Once I got the analogRead code working, which turned out to be non-trivial due to an error in the Fubarino Mini documentation, I could turn the pot and watch the value change in the MPIDE terminal window on my laptop. I had previously developed a driver for a Sparkfun I^2C 2x16 LCD display, so it was easy at this point to have my UNO32 ask the Fubarino Mini to read the pot and then have the UNO32 send the value to the MPIDE terminal window. Interesting, but something a little more general is needed.

```
// Define the Process Table Structure
 struct Process {
  byte Process_Id;                    // Remote Process Id
  byte I2C_Address;                   // Remote I2C Address
 byte Tx_Length;                      // Transmit Length
  byte * Tx_Buffer;                   // Transmit Buffer Pointer
  byte Rx_Length;                     // Receive Length
  byte * Rx_Buffer;                   // Receive Buffer Pointer
 };
```

Table 1—Process table C structure.

```
// Define the Application Specific Process Table
 struct Process Processes[MaxProcesses] = {
  ReadAN12, I2CAddr1, 1, Tx_Buffer_Ptr, 3, Rx_Buffer_Ptr,
  Process2, I2CAddr1, Transmit_Length, Tx_Buffer_Ptr, Receive_Length, Rx_Buffer_Ptr,
  Process3, I2CAddr1, Transmit_Length, Tx_Buffer_Ptr, Receive_Length, Rx_Buffer_Ptr,
  Process4, I2CAddr2, Transmit_Length, Tx_Buffer_Ptr, Receive_Length, Rx_Buffer_Ptr,
  Process5, I2CAddr2, Transmit_Length, Tx_Buffer_Ptr, Receive_Length, Rx_Buffer_Ptr,
 };
```

Table 2—Application specific process table (an array of structures).

```
// Process State Machine - this code would be blended into the Rebel main loop
// Five processes setup for demonstration purposes only. YMMV.

switch(Current_State) {
        case ReadAN12:
                ReadAN12_Code();            // Read an analog from a remote processor
                Current_State++;            // Cycle to the next process
                break;
        case Process2:
                Process2_Code();
                Current_State++;
                break;
        case Process3:
                Process3_Code();
                Current_State++;
                break;
        case Process4:
                Process4_Code();
                Current_State++;
                break;
        case Process5:
                Process5_Code();
                Current_State++;
                break;
        default: {Current_State = ReadAN12;}
        }
delay(40);                                  // Slow down to read the debug prints
}
```

Table 3—Process state machine.

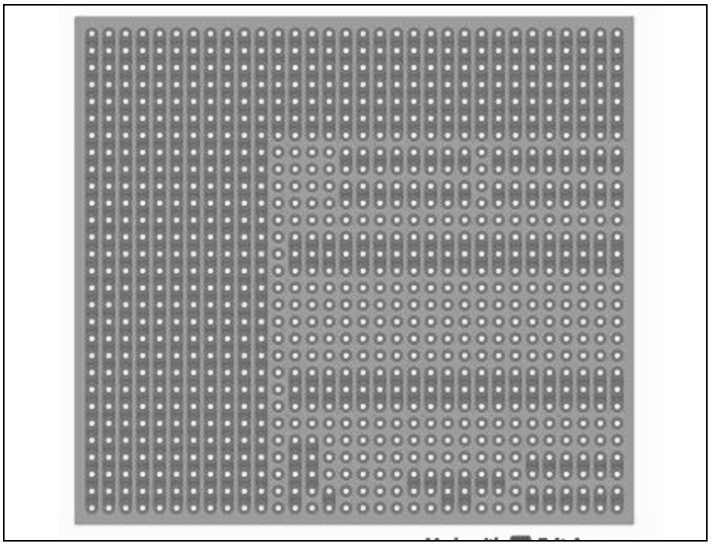

Figure 4—Strip board as laid out by Fritzing.

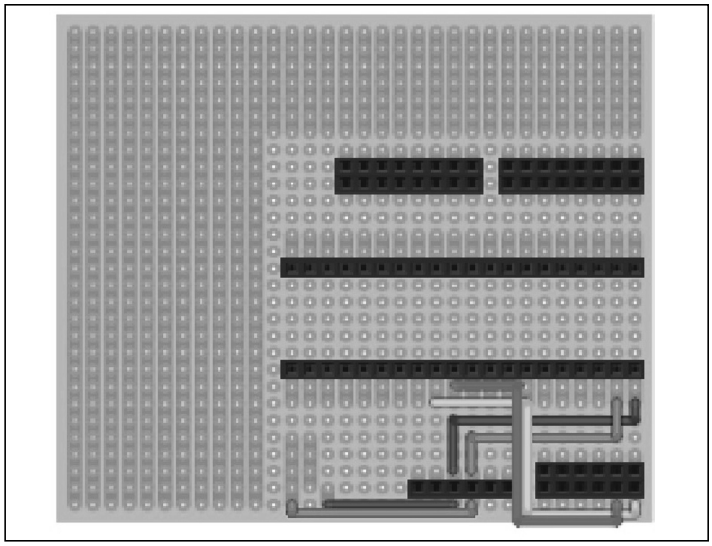

Figure 5—Wiring and parts placement.

What does it take to talk to an I^2C device? At the most basic level, the requirement is for an I^2C Address (7 or 10 bit) and one or more data bytes. Suppose we let the first data byte be the ID of a process to be run on a remote processor. The remaining data bytes are then determined by the needs of the specific process. These packet layouts become what is known as a protocol, a kind of mini TCP/IP. This can also be known as a Remote Procedure Call (RPC). The Rebel sends out a command packet to tell the remote to execute a process and build a response packet. The Rebel then sends out a second packet to request the remote to send back the response packet. A simple State Machine (a form of a C Switch statement) handles the dispatching of data packets to processes on the remote(s) and cycling through RPC calls on the Rebel.

For one remote process this is easy, but for many I^2C devices each with many processes, this can get very messy very quickly. So it was time to dig out the C manual and (re)learn about C Structures. Structures allow you to create custom data types which are built from the simple data types—byte, char, int, float, etc. This is similar to a database record containing fields of different types. What resulted is a table-driven I^2C driver for the Rebel and an additional driver for the I/O Expansion Processor(s). Small code snippets are given in Tables 1 to 3. The complete drivers are available on the Yahoo group file section or on my github repository [5].

Given working I^2C software drivers, how to implement the hardware side of things? I don't do Eagle(yet) and I don't etch my own boards. Well, my second favorite magazine, behind *QRP Quarterly* of course, is *Nuts & Volts* [6]. Ron Hackett in his "Picaxe Primer" column uses a lot of strip boards for his projects. Joe Pardue in "Smiley's Workshop" has done a year-long series on the Fritzing [7] prototyping tool applied to Arduino projects, including an I^2C hand-held terminal. And for good measure check out Fred Eady's "The Design Cycle" for all things PIC, Wifi, and ISM band RF data modules.

While Fritzing is best known for taking a breadboard design on to the schematic generation, PCB layout and production phases, I discovered that it also supports stripboards for first stage design. Figures 4 and 5 show my Fritzing layouts and Figure 6 shows a mostly complete stripboard designed as a carrier for my Fubarino Mini (40 pins). It appears, but I have not yet tested it, that this board could also act as a carrier for the Microchip dsPIC33F Microstrip DSP module (28 pins). Remember that this DSP module is not in the Arduino family and would have to be programmed via Microchip's MPLAB suite. And the power, ground and I^2C pins would have to be remapped on the expansion board.

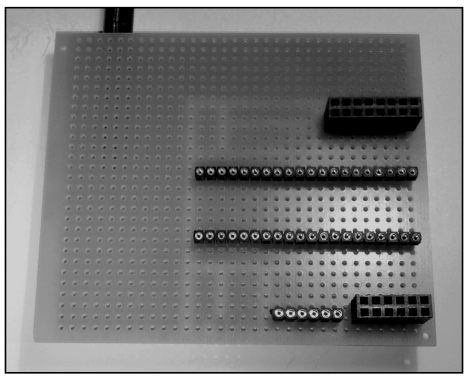

Figure 6—Finished I^2C board.

I noticed in working on this board that not all of the female headers on the chipKIT UNO32 are laid out on 0.1" centers. One header is shifted by 0.05" and thus doesn't fit into the stripboard. I later read that this was done because of a layout error in the original Arduino board design that has been "grandfathered" and carried forward into new designs. In hind sight, I would have used all male headers and forgot about the female headers and "stackability".

I plan to continue following the progress of The Rebel Open Source experiment. I'll check back as your interest and developments warrant.

References

1. James Lynes, "The chipKIT UNO32 and Friends," *QRP Quarterly*, Fall 2013.
2. www.groups.yahoo.com/group/TenTec506Rebel
3. www.github.com/pstyle/Tentec506
4. Credit for Figures 1, 2, and 3 goes to Johan van Dijk, PA3ANG (pa3ang@xs4all.nl).
5. www.github.com/jmlynesjr/chipKIT-Arduino-Examples
6. www.nutsvolts.com
7. http://fritzing.org/

●●