One of the problems with band decoders is that you have to live with the definition imposed by designer of the band decoder. Since the Arduino is flash memory based, and you have the tools to regenerate the code, what can be done is a blank canvas. For those who do not wish to get involved with programming, you might wish to use my code.

My band decoder routines are table driven, so the user only needs to change the tables to change behavior of the band decoder. Below are the three tables that define the band decoder with my software. This is an excellent starting point. I will use these tables to describe the operation of the code.

```
PROGMEM prog_uint32_t FreqTable[] = {
  0 , 1800, 2000, 3500,4000,5330,5410,7000,7300,10100,
10150,14000,14350,18068,18168,21000,21450,  24890,24990,28000,
29700,50000UL,54000UL,144000UL,148000UL,
219000UL,225000UL,420000UL,450000UL, 0xffffffff };

PROGMEM prog_int8_t AntTable[] = {
  5 ,   1,   5,  2 ,  5,  5,  5,  3,  5,    1,
  5,   4,   5,   4,   5,   4,   5,   4,   5,   4,
  5,   4,    5,   5,    5 ,
   5,    5,   5,    5,     5};

PROGMEM prog_int8_t AntPinTable[] = { 23,25,26,};
```

**Syntax aid:**

For those unfamilair with C or C++ these lines of code look like greek. So let's first investigate what is happening on a line. Each tables have the following elements:

Type Name = { table values };

A type, a variable name and equal sign and then the actual table value between the two "curly brackets". The semicolon denoted the end of the line. In C, spaces, tabs or carriage returns are ignored. Each number in between the values are separated by a comma.

Looking at the first table, the type is "PROGMEM prog_uint32_t". That is odd. PROGMEM is a key word to tell the atmel compilier that this variable is to be stored in program (flash) memory, as compared to RAM. "prog_unit32_t" is a special type that defines an unsigned 32 bit integer that is to reside in program memory. FreqTable is the variable name and the square brackets "[]" indicates that this variable is an array of numbers. The variable is thus an array of unsigned 32 bit integers.

The next two tables are similiar, however they are of a data type prog_int8_t, or 8 bit signed integers.

**FreqTable and AntTable**

The Frequency Table is a list of all the breaks between bands in kilohertz (Khz) in assending order of frequency. You must make sure that the values are in assending order. The first number of the list must be 0 (DC) and the last number of the list must be the largest possible number for a 32 bit integer, in this case expressed as the hex value of FFFFFFFF. The compilier need to be told if a

number is greater than 64000 by placing the UL after it. UL means "Unsigned Long". This is something kinda unique to this embedded compilier. Looking at the first few numbers:

0 , 1800, 2000, 3500,4000, ...

This defines the first four bands. The first band is between 0KHz and 1800KHz, which represents the spectrum that is below the amateur 160 meter band in the US. The second band is between 1800KHz and 2000KHz, which represents the US 160 meter amateur band. The third band is between 2000KHz and 3500KHz, representing the spectrum between the US 160 meter amatuer band and the 80 meter band. Look at FreqTable and you should recogized each of the US Amateur Radio bands up through the 450MHz.

It should be noted that each number represents the lowest frequency that is in the band, and the highest frequency that is not in the band. For example while tuning 80 and 75 meters with this table, tuning to 3500KHz will place you "in band" while tuning in 4000KHz will place you out of the band.

The size of the antenna table, or in other words, the number of entries is limited to 32,767 entries. Hopefully this exceed ones patients. The idea is that a band segment can be as small as one kilohertz. This permits building very complex frequency depent antenna configurations.

AntTable defines which antenna is selected for each corresponding band defined in FreqTable. In the case above the antenna table is :

5 , 1, 5, 2 ,

In my shack, antenna 5 selects dummy load, 1 selects the 160 meter inverted L and 2 selects my 80 meter dipole. So, as you can see, I have it so that the dummy load is selected when I tune outside of an amatuer band.

Between the frequency table and the antenna table complex selection criteria can be acheived. As your antenna farm grows or changes, this map can change to adapt, by changing the table and placing the new code into the processors flash memory.

**AntPinTable**

The antenna pin table defines what pins are the data placed.

The value of the antenna table can be placed to output pins in one of two ways; either with software decoding, or with hardware decoding. In my case, I run three wires to an Array Solutions Six Pak and have a have a hardware 1 of N decoder. Alternately, you can do the decoding in software and have a pin per selected antenna. In either event, the values in the antenna pin table are the Ardunio Mega pin number, starting with the least significant bit of the output, going to the most significant output.

This table only needs to be changed if the external hardware needs to be changed.