Using EEPROM
in your DDS Development Kit

By Bruce Hall, W8BH

This article will describe how to store and retrieve data using the EEPROM in your DDS Development kit. I will upgrade the VFO memory project with EEPROM, enabling frequency memories to be created and stored without programming.

EEPROM stands for Electrically Erasable and Programmable Read Only Memory. It is a section of the ATmega88 microcontroller which is used for storing small amounts of data. Up to 512 bytes of data can be stored, which remain on the chip until erased. In today's gigabyte word, half a kilobyte does not sound like much, but it is still very useful in our DDS kit.

In my VFO memory project, described at http://w8bh.net/avr/AddMemories.pdf, I created a list of frequency presets. You can scroll through them with the encoder, allowing quick band changes. Unfortunately, once you compile your code, you are stuck with whatever frequencies you programmed. There is no way to change presets 'in the field'.

EEPROM gives us a great way to save VFO memories and other program settings. EEPROM allow us to change these settings without recompiling, and untethers our DDS kit from the programming cable.

I got excited about using EEPROM and quickly searched the internet for good programming examples. I found plenty of C code examples, mostly written for the Arduino, that show how to use EEPROM. But there were very few assembly language examples. I hope that some of you find my example helpful in your own designs.

Reading and Writing EEPROM

Reading and writing the EEPROM is more complicated that reading and writing data from other memory locations. It is almost like the EEPROM is on another chip. Each data operation involves setting 4 different registers: the address registers (high and low), the data register, and the control register. These registers are in the chip's I/O space, meaning that they are accessed with the IN and OUT instructions. The actual data transfer is accomplished by setting individual bits in the control register.

Here is the code for reading a byte from EEPROM:

```
ReadEE:
       sbic   EECR,EEPE                    ;busy writing EEPROM?
       rjmp   ReadEE                       ;yes, so wait
       out    EEARH,YH                     ;set up address reg.
       out    EEARL,YL
       sbi    EECR,EERE                    ;strobe the read bit
       in     temp1,EEDR                   ;get the data
       ret
```

The first thing to do is make sure the EEPROM is not busy writing data. Reading EEPROM is fast, and takes only a few microcontroller cycles, but writing EEPROM is much slower. On average, it takes 3.4 milliseconds to write a byte to EEPROM. The first two lines check the EEPE (EE programming enable) bit in the control register to see if a write operation is in progress. It loops here until the write operation, if any, is completed.

Next, we set up the EEPROM address registers, high and low, with the address that we want to write to. In my code I am using the Y register combination, which is registers R28 and R29. You may use any two registers that you like.

Once we know it is OK to read and the address is loaded, it is time to request the byte. The EE read is triggered by setting the read enable bit (EERE) in the control register with the SBI instruction. The byte is transferred from EEPROM into the EEDR data register, which is then transferred to temp1 (or any other register) by the IN instruction.

It took me a while to get used to seeing all of those EE's. Mentally ignore them, and the code looks a lot simpler. The write operation is very similar:

```
WriteEE:
       sbic   EECR,EEPE                    ;busy writing EEPROM?
       rjmp   WriteEE                      ;yes, so wait
       out    EEARH,YH                     ;set up address reg.
       out    EEARL,YL
       out    EEDR,temp1                   ;put data in data reg.
       cli                                 ;dont interrupt the write
       sbi    EECR,EEMPE                   ;master write enable
       sbi    EECR,EEPE                    ;strobe the write bit
       sei                                 ;interrupts OK now
       ret
```

The first four lines are the same. We wait until any pending write operation is completed, and then set up our address. Next we write our data, in register temp1, to the EEPROM data register. Our EEPROM data and address registers are now ready, but we have a couple more tasks before we can write. Writing is a two-step process: first, we prepare for the write by setting the Master Programming Enable (EEMPE) bit; then, we request the write by setting the Programming Enable (EEPE) bit. These two operations must happen sequentially and without interruption, so interrupts are first disabled by CLI and then re-enabled by SEI.

After finishing the VFO memory project, I thought about what would happen after I installed my VFO. I could use the memories that I had originally programmed, but there was no easy way to add or change them. I would have to open the radio, reattach the programming header, remember how to use AVR studio (or whatever I program I used), find my source code, edit it, reassemble, and then burn the new code into my ATmega88 chip. This wouldn't be much of a hassle if it happened next week, or maybe even next month. But what if I wanted to do it next year? Ugh. I'd probably just make do with the original presets. With EEPROM we have a way of saving NEW information on our chip, without go through all of these steps. I decided it was time to upgrade the project, adding EEPROM routines as a way of saving and retrieving frequency settings.

First, a little planning is in order. The EEPROM is 512 bytes in size. Is this enough space for what we want? Each frequency preset, without any space-saving measures, takes 8 bytes. If we use EEPROM for presets only, we can store up to 512/8 = 64 memory presets. For me this is more than enough. I decided that I would use only the first half of the available memory, which is 256 bytes. Of this half, I would save 32 bytes for non-preset data, and have room for 28 presets. You can divide your EEPROM space however you want. My plan gives me the space that I need now and leaves plenty for future expansion.

After writing routines for loading and saving individual bytes, it seems natural to create routines for loading and saving memory presets. We need to move 8 bytes at a time, so here is a routine for moving 8 bytes:

```
Read8E:                                 ;read 8 bytes from EE
       ldi    temp2,8                   ;counter=8
r81:   rcall  ReadEE                    ;get byte from EE
       st     Z+,temp1                  ;move byte to destination
       adiw   Y,1                       ;go to next EE addr
       dec    temp2
       brne   r81
       ret
```

The ReadEE routine used Y as the EEPROM address pointer, so I used Z as the destination (SRAM) pointer. It is a simple loop, counting out 8 bytes and incrementing the pointers as we go. Incrementing Z is done using the ST Z+ instruction. We are not loading memory from Y, however, so we cannot use a Y auto-increment instruction. Instead, I used one of the few 16-bit instructions, ADIW, to add one to the current Y value. The Write8E routine is almost exactly the same, calling WriteEE instead of ReadEE.

We need to move the presets between EEPROM and our frequency buffer, which is located at LDCrcve0. According my EEPROM memory plan above, I saved the first 32 bytes for future use, and then stored each 8 byte preset sequentially, #00 to #27. The EEPROM address for a given preset should therefore be 8*(preset #) + 32. Here is the code for reading a frequency preset from EEPROM:

```
LoadEEMem:
;      specify the preset# in temp1
;      will return the EE memory into LCDrcve0
       clr    YH
       ldi    YL,32                     ;start of presets
       ldi    ZH,high(LCDrcve0)
```

```
        ldi     ZL,low(LCDrcve0)
ge0:    cpi     temp1,0
        breq    ge1
        adiw    Y,8
        dec     temp1
        brne    ge0
ge1:    rcall   Read8E
        ret
```

Starting at ge0, a simple loop counts the number of presets, adding 8 to the EEPROM memory pointer for each preset. By the time this loop is done, we've added 8*preset# to the address. We started counting at 32, so the pointer is at 8*preset+32. You can use the hardware multiply instruction instead of looping, if you prefer. With Y and Z pointing to the correct locations, all that remains is to read the 8 bytes with Read8E. The code to write is almost exactly the same, calling Write8E instead of Read8E.

Saving 'Factory Defaults' to EEPROM

The VFO memory code is modified to call LoadEEMem instead of the original LoadPreset routine. But initially there is nothing in the EEPROM to read. We could start with a blank EEPROM and store each frequency manually, but it would be more convenient to have the EEPROM load itself with some 'factory' default frequencies the first time it is used.

How can we determine if the EEPROM has ever been programmed? One method is to check a specific spot on the EEPROM for a signature word. It is digital graffiti, a "Kilroy was here" signature. If the word is present, then "we've been here before" and the EEPROM is assumed to be programmed. If we read something other than the expected signature, then we write our signature and initialize the EEPROM with the default frequency presets. I created two bytes, SigByte1 and SigByte2, for my signature. You can use a single byte instead, with the assumption that random data in the EEPROM won't match your byte. Non-initialized EEPROM has a good chance (but not guaranteed) of being $00 or $FF, so a single byte value will probably work. I put my signature at the beginning of the EEPROM address space, using 2 of the 32 non-preset bytes.

If the signature word is incorrect, the EEPROM is programmed with the source code frequencies. I created a table of 20 frequencies, located at the end of the program. You can make less or more (up to 28) by changing the MaxPreset equate. The data is written to the EEPROM using the following somewhat lengthy code:

```
ProgramEE:
;       copy default memories from program FLASH to EE
        ldi     temp1,8                 ;'EEPROM STARTUP'
        rcall   DisplayLine1            ;display it
        clr     YH
        clr     YL                      ;start with EEPROM byte 00
        ldi     temp1,SigByte1          ;load first signature byte
        rcall   WriteEE                 ;write it
        inc     YL                      ;go to byte 1
```

```
        ldi     temp1,SigByte2              ;load second signature byte
        rcall   WriteEE                     ;write it
        ldi     temp2,30                    ;create space for 30 values
pe1:    inc     YL                          ;go to next EERPOM byte
        clr     temp1
        rcall   WriteEE                     ;write a 0 to EEPROM
        dec     temp2                       ;all 30 byte written?
        brne    pe1                         ;loop until all written
        ldi     temp2,MaxPreset*8+8         ;load # of preset bytes
        ldi     ZH,high(presets*2)          ;point to preset bytes
        ldi     ZL,low(presets*2)
pe2:    lpm     temp1,Z+                    ;get byte from program memory
        adiw    Y,1                         ;go to next EE addr
        rcall   WriteEE                     ;store byte in EE
        dec     temp2                       ;all preset bytes written?
        brne    pe2                         ;loop until all written
        ret
```

You might want to break up this routine into smaller pieces, for easier readability. I kept it together since it accomplishes a single uninterrupted task, programming the EEPROM from Flash memory. The first two lines let the user know we are busy with EEPROM programming. Each write takes more than 3 ms, and we lots of bytes to write – about 0.7 seconds total in the current configuration.

Next, we write the signature word followed by 30 zeroed-out bytes for future use. Starting at label pe2, each frequency preset is loaded from program memory and copied to EEPROM. These presets are read one byte at a time by the LPM instruction, and stored in EEPROM by the WriteEE routine. Notice that program memory is indexed differently than SRAM memory, and so we need to multiply our ‘presets’ label by 2 to get to the correct program memory location.

Each time the program starts, we need to check the signature word and initialize EEPROM if necessary. The appropriate spot for this check is in the initialization section, before the start of the main program loop. If the retrieved data matches our signature, then EEPROM is ready for use. Otherwise, program it with our default presets:

```
CheckEE:
;       looks to see if EE has been loaded with default presets
;       if not, defaults are programmed into the EE
        clr     YH
        clr     YL                          ;go to byte 00
        rcall   ReadEE                      ;look at first signature byte
        cpi     temp1,SigByte1              ;is it correct?
        brne    ee1                         ;no, so store defaults
        inc     YL                          ;go to byte 01
        rcall   ReadEE                      ;look at second signature byte
        cpi     temp1,SigByte2              ;is it correct?
        brne    ee1                         ;no, so store defaults
        rjmp    ee2                         ;signature byte OK, so done
ee1:    rcall   ProgramEE                   ;write defaults to EE
ee2:    ret
```

Loading factory defaults is useful enough to consider adding it to the user-interface.

Upgrading the VFO Memory project

Now that the EEPROM has been loaded with preset frequencies, we can upgrade the VFO memory project to get its presets from EEPROM. The mode1 encoder routine now looks like this:

```
ENCODERMODE1:
       lds    temp1,preset              ;get current preset#
       tst    encoder                   ;which way did encoder turn?
       brmi   e11
       cpi    temp1,MaxPreset           ;CW rotation
       brge   e12                       ;hard stop at Max Preset
       inc    temp1                     ;go to next higher preset
       rjmp   e12
e11:   cpi    temp1,0                   ;CCW rotation
       breq   e12                       ;hard stop at 0
       dec    temp1                     ;go to next lower preset
e12:   clr    encoder                   ;ignore any more requests
       sts    preset,temp1              ;save current preset#
       rcall  LoadEEMem                 ;get the preset in LCD buffer
       rcall  ShowPreset                ;and display it
```

This code increments/decrements the preset variable, according to encoder direction, limiting the value to between 0 and MaxPreset. Thanks to a suggestion by Tom, AK2B, I use the LDS/STS instructions to access the preset variable. The last two lines are the business end of the routine, loading the preset from EEPROM and then displaying it on the LCD. In my first version of the project, I used the original output routine, which displays the frequency as “10,000,000 Hz”. I found myself mistakenly trying to ‘tune’ this value, forgetting I was in preset mode. So I modified the display a little bit, showing the preset number as #00, followed by a more compact frequency displayed as “10.000000”.

To display the preset number, I first used the ShowHex routine from the original code. But when it displays ‘13’, it is hard to remember that this is actually preset decimal number 19 and not 13. So I wrote a small routine to display the decimal number instead. I thought I would be able to find a quickie routine from the internet, but everything I found was too complicated. All we need is to decode one byte and display a decimal number from 0 to 99. So I got out my notepad and wrote something that counted the number of tens (10’s), displayed it, and then displayed the remainder, which is the number of ones. For example, 23 is 2 tens and 3 ones. Here it is:

```
SHOWDECIMAL:                ;displays a number 00-99 on the LCD
       clr    temp2                     ;10's counter
sd1:   cpi    temp1,10                  ;at least 10 remaining?
       brlo   sd2                       ;no, done counting 10's
       inc    temp2                     ;count the next 10
       subi   temp1,10                  ;remove the next 10
       brpl   sd1                       ;loop until all 10's gone
sd2:   mov    temp3,temp1               ;save 10's counter
       mov    temp1,temp2
       rcall  ShowDec                   ;display 10's digit
       mov    temp1,temp3               ;get 1's digit
```

```
        rcall  ShowDec                     ;and display it
        ret
```

The VFO project changed frequencies while scrolling through the presets. I thought about this and decided that I only wanted to change the DDS output when I selected a preset, not when scrolling through them. So I moved the DDS update to the button's Tap Up event.

In the Button Event project [http://w8bh.net/avr/ButtonEvents.pdf], I derived four different events for a single button press: tap-down, tap-up, hold-down, and hold-up. When using a hold to change modes, the hold-up event is useful as a way to initialize the new mode, and prepare the display for user-input. The tap-down event can be used for the button press, but if it will also be called when the user is pressing the button at the start of a hold. If you need a strictly non-hold event, using tap-up might be a better choice than tap-down.

Loading presets from EEPROM is finished, but only half of the project. We still need a way of saving new presets. Where should we attach our new EEPROM writing routines? I created another mode for this, since it seemed intuitive. Could you put it somewhere else? I suppose you could put it at after scrolling the last preset, or before the first preset. You could also dedicate an input pin to enable EEPROM programming, and check this pin when jumping to the preset routines. Maybe you can find another spot that works better for you.

The code for the mode2 encoder routine looks very similar to mode1:

```
ENCODERMODE2:
        lds    temp1,preset                ;get current preset#
        tst    encoder                     ;which way did encoder turn?
        brmi   e21
        cpi    temp1,MaxPreset             ;CW rotation
        brge   e22                         ;hard stop at Max Preset
        inc    temp1                       ;go to next higher preset
        rjmp   e22
e21:    cpi    temp1,0                     ;CCW rotation
        breq   e22                         ;hard stop at 0
        dec    temp1                       ;go to next lower preset
e22:    clr    encoder                     ;ignore any more requests
        sts    preset,temp1                ;save current preset#
        rcall  ShowPresetNum               ;display preset number
        ret
```

It is almost exactly the same, except that the step that loads the preset from memory is missing. We only want to the user to decide the *location* of our new frequency preset. So the action is to change the preset number while the encoder shaft is rotated. The preset is not written to EEPROM until the user confirms the save with a tap. Again, tap-up is used, so that he user can hold the button down to cancel:

```
TAPUP2:
        lds    temp1,preset
        rcall  SaveEEmem                   ;save preset to EEPROM
        ldi    temp1,9
        rcall  DisplayLine2                ;display 'SAVED'
        ldi    temp1,2
```

```
rcall  Blink_LED                  ;blink for user feedback
ldi    temp1,0
rcall  ChangeMode                 ;return to tuning mode
ret
```

The preset is written by the call to SaveEEMem. This is all you really need. I added a LCD message and a LED blink to visually indicate that the preset was saved. After saving I return control back to tuning mode.

That's about it. What else could we do with the EEPROM? It would be a great place to save some VFO preferences, such as:

- Preferred tuning rate
- Tuning direction
- Frequency display format
- RIT/Split settings
- IF offset(s)/IF mode

All of these preferences can fit in the reserved space at the beginning of the EEPROM address space. If you are using the DDS kit for more than just a VFO (for example, rig control or a keyer) you still have plenty of space in the upper 256 bytes for other settings. If this still isn't enough space for your application, upgrading to the ATmega328 will double the EEPROM to 1024 bytes.

Below is the full source code for my upgraded VFO memory project, which now uses EEPROM to save frequencies. The interface works, but there is always room for improvement. For example, a 'Restore Factory Defaults' option would be useful. Don't be afraid to modify it to suit your needs. Have fun!

Instructions

To go to a preset frequency: hold the button down for 1 second until you see 'scroll presets' on the LCD. Turn the encoder knob until you see the frequency you want. Press the button to select the frequency.

To save a new preset frequency: First, go to the frequency you want to display. Then hold the button down for 2 seconds, until you see 'save a preset'. Turn the encoder knob until it displays the slot in which you want to save the preset. Press the button to save the frequency. If you want to cancel without saving, hold the button down for 1 second until you see 'VFO tuning mode'.

Source Code

```
; 1.  In the block of defines, Add/Change the following register definitions:
;     .def release = R21
;     .def hold    = R15

; 2.  In .dseg, add the following lines
;     mode:   .byte 1   ; 0=tuning mode; 1=load preset; 2=save preset
;     preset: .byte 1   ; buffer for current frequency preset number
;     flags:  .byte 1   ; bit0 = hold in progress

; 3. Insert the following instruction below the 'menu' label
;     rjmp  W8BH                    ;!! go to new main program



;*********************************************************
;*     W8BH - INTERRUPT VECTOR TABLE
;*********************************************************
;  use RJMP instructions with ATmega88 chips
;  use JMP instructions with ATmega328 chips

.cseg
.org $000
       jmp    RESET
.org INT0addr
       jmp    EINT0                 ; New External Interrupt Request 0
.org INT1addr
       jmp    EINT1                 ; New External Interrupt Request 1
.org OVF0addr
       jmp    OVF0                  ; Timer/Counter0 Overflow
.org OVF2addr
       jmp    OVF2                  ; Timer/Counter2 overflow
.org INT_VECTORS_SIZE



;*********************************************************
;*     W8BH - INITIALIZATION CODE
;*********************************************************

W8BH:
       ldi    temp1,$03             ;binary 0000.0011
       out    DDRB,temp1            ;set PB0,1 as output

       ldi    temp1,$3C             ;binary 0011.1100
       out    PORTB,temp1           ;set pullups on PB2-5

       ldi    temp1,$A3             ;b1010.0011 (add bit PD7)
       out    DDRD,temp1            ;set PD0,1,5,7 outputs

       clr    release               ;no button events on startup
       clr    hold                  ;no hold events on startup
       clr    temp1
       sts    mode,temp1            ;start mode0 = normal operation
       sts    flags,temp1           ;nothing to flag yet
       sts    preset,temp1          ;start with first preset

       ldi    temp1, $07            ;set timer2 prescale divider to 1024
       sts    TCCR2B,temp1
       ldi    temp1, $01            ;enable TIMER2 overflow interrupt
       sts    TIMSK2,temp1
```

```
        rcall  CheckEE                    ;make sure EEPROM is initialized
        ldi    temp1,1
        rcall  DisplayLine1               ;startup message


;**********************************************************
;*  W8BH - REVISED MAIN PROGRAM LOOP
;**********************************************************

MAIN:
        rcall  CheckEncoder               ;check for encoder action
        rcall  CheckButton                ;check for button events
        rcall  CheckHold                  ;check for button holds
        rcall  Keypad                     ;check for keypad action
        rjmp   Main                       ;loop forever


CHECKENCODER:
        tst    encoder                    ;any encoder requests?
        breq   ce9                        ;no, so quit
        lds    temp1,mode
        cpi    temp1,0                    ;are we in normal mode (0)?
        brne   ce1                        ;no, skip
        rcall  EncoderMode0               ;yes, handle it
        rjmp   ce9
ce1:    cpi    temp1,1                    ;are we in mode 1?
        brne   ce2                        ;no, skip
        rcall  EncoderMode1               ;yes, handle it
        rjmp   ce9
ce2:    cpi    temp1,2                    ;are we in mode 2?
        brne   ce3                        ;no, skip
        rcall  EncoderMode2               ;yes, handle it
        rjmp   ce9
ce3:
ce9:    ret


CHECKHOLD:
        tst    hold                       ;any new hold event?
        brpl   ck1                        ;no, so quit
        lds    temp1,flags
        sbr    temp1,$01                  ;flag the hold
        sts    flags,temp1                ;save it
        rcall  ButtonHoldDown             ;do the hold event
        clr    hold                       ;reset = allow future holds
ck1:    ret


CHECKBUTTON:
        tst    encoder                    ;any encoder requests?
        brne   cb4                        ;wait until encoder is done
        tst    press                      ;any button down events?
        breq   cb1                        ;no, check for button up events?
        rcall  ButtonTapDown              ;do the button down
        dec    press                      ;one less button tap to do
cb1:    tst    release                    ;any button up events?
        breq   cb4                        ;no, so quit
        lds    temp1,flags
        sbrs   temp1,0                    ;is there a hold in progress?
        rjmp   cb2                        ;no
        cbr    temp1,$01                  ;yes, remove hold flag
        sts    flags,temp1                ;save un-held state
```

```
        rcall  ButtonHoldUp                 ;do hold release
        rjmp   cb3
cb2:    rcall  ButtonTapUp                  ;do the Tap Release
cb3:    dec    release                      ;one less release to do
cb4:    ret


BUTTONTAPUP:
        lds    temp1,mode                   ;get mode
        cpi    temp1,0                      ;are we in mode0?
        brne   tu1                          ;no, skip
;       rcall  TapUp0                       ;yes, handle it
        rjmp   tu9
tu1:    cpi    temp1,1                      ;are we in mode1?
        brne   tu2                          ;no, skip
        rcall  TapUp1                       ;yes, handle it
        rjmp   tu9
tu2:    cpi    temp1,2                      ;are we in mode2?
        brne   tu3                          ;no, skip
        rcall  TapUp2                       ;yes, handle it
        rjmp   tu9
tu3:                                        ;placeholder for higher modes
tu9:    ret

BUTTONTAPDOWN:
        lds    temp1,mode                   ;get mode
        cpi    temp1,0                      ;are we in mode0?
        brne   td1                          ;no, skip
        rcall  TapDown0                     ;yes, handle it
        rjmp   td9
td1:    cpi    temp1,1                      ;are we in mode1?
        brne   td2                          ;no, skip
;       rcall  TapDown1                     ;yes, handle it
        rjmp   td9
td2:    cpi    temp1,2                      ;are we in mode2?
        brne   td3                          ;no, skip
;       rcall  TapDown2                     ;yes, handle it
        rjmp   td9
td3:                                        ;placeholder for higher modes
td9:    ret

BUTTONHOLDUP:
        lds    temp1,mode                   ;get mode
        cpi    temp1,0                      ;are we in mode0?
        brne   hu1                          ;no, skip
        rcall  HoldUp0                      ;yes, handle it
        rjmp   hu9
hu1:    cpi    temp1,1                      ;are we in mode1?
        brne   hu2                          ;no, skip
        rcall  HoldUp1                      ;yes, handle it
        rjmp   hu9
hu2:    cpi    temp1,2                      ;are we in mode2?
        brne   hu3                          ;no, skip
        rcall  HoldUp2                      ;yes, handle it
        rjmp   hu9
hu3:                                        ;placeholder for higher modes
hu9:    ret

BUTTONHOLDDOWN:
        lds    temp1,mode                   ;get mode
        cpi    temp1,0                      ;are we in mode0?
        brne   hd1                          ;no, skip
        rcall  HoldDown0                    ;yes, handle it
```

```
        rjmp    td9
hd1:    cpi     temp1,1                 ;are we in mode1?
        brne    hd2                     ;no, skip
        rcall   HoldDown1               ;yes, handle it
        rjmp    hd9
hd2:    cpi     temp1,2                 ;are we in mode2?
        brne    hd3                     ;no, skip
        rcall   HoldDown2               ;yes, handle it
        rjmp    hd9
hd3:
hd9:    ret


CHANGEMODE:
;       call this routine with new mode in temp1
;       only action is to change the message on Line 1
        sts     mode,temp1              ;save the new mode
        cpi     temp1,0                 ;mode 0?
        brne    cm1                     ;no, skip
        inc     temp1
        rcall   DisplayLine1            ;yes, show normal title
        rjmp    cm9
cm1:    cpi     temp1,1                 ;mode 1?
        brne    cm2                     ;no, skip
        inc     temp1
        rcall   DisplayLine1            ;yes, show mode 1 title
        rjmp    cm9
cm2:    cpi     temp1,2                 ;mode 2?
        brne    cm3                     ;no, skip
        inc     temp1
        rcall   DisplayLine1            ;yes, show mode 2 title
        rjmp    cm9
cm3:                                    ;placeholder for higher modes
cm9:    ret


QUICKBLINK:
        cbi     PORTC,LED               ;turn  LED on
        ldi     delay,15                ;keep on 20 ms
        rcall   wait
        sbi     PORTC,LED               ;turn LED off
        ret


;*******************************************************
;*  W8BH - MODE 0 (VFO TUNING) ROUTINES
;*******************************************************

ENCODERMODE0:
;       This code taken from original program loop.
;       Called when there is a non-zero value for encoder variable.
;       Negative encoder values = encoder has turned CCW
;       Positive encoder values = encoder has turned CW
;       In mode 0, encoder should increase/decrease the DDS freq

        tst     encoder
        brpl    e02                     ;which way did encoder rotate?
        inc     encoder                 ;remove 1 negative rotation
        rcall   DecFreq0                ;reduce displayed frequency
        cpi     temp1,55                ;55 = all OK
        brne    e01
        rcall   IncFreq0                ;correct freq. underflow
        rjmp    e05
```

```
e01:   rcall  DecFreq9                    ;reduce magic number
       rjmp   e04

e02:   dec    encoder                     ;remove 1 positive rotation
       rcall  IncFreq0                    ;increase displayed frequency
       cpi    temp1,55                    ;55 = all OK
       brne   e03
       rcall  DecFreq0                    ;correct freq. overflow
       rjmp   e05
e03:   rcall  IncFreq9                    ;increase magic number

e04:   rcall  FREQ_OUT                    ;update the DDS
       rcall  ShowFreq                    ;display new frequency
e05:   rcall  QuickBlink
       ret

TAPDOWN0:
;      This code taken from original program loop.
;      Called when there is a non-zero value for press variable.
;      Non-zero value = number of times button has been pressed
;      In mode 0, button should advance cursor to the right

       tst    encoder                     ;check for pending encoder requests
       brne   b01                         ;dont advance cursor until encoder done
       dec    StepRate                    ;advance cursor position variable
       brpl   b01                         ;position >= 0 (Hz position)
       ldi    StepRate,7                  ;no, so go back to 10MHz position
b01:   rcall  ShowCursor
       rcall  QuickBlink                  ;flash the LED
       ret


HOLDDOWN0:
;      Called when button has been held down for about 1 second.
;      In mode 0, action should be to invoke mode1 = scrolling freq. presets

       ldi    temp1,1
       rcall  ChangeMode                  ;go to next mode
       ret

HOLDUP0:
;      Called when entering this mode from another mode
       rcall  ShowTuning
       ret


;**********************************************************
;*  W8BH - MODE 1 (LOAD FREQUENCY PRESET) ROUTINES
;**********************************************************

INITMODE1:
       lds    temp1,preset
       rcall  LoadEEMem
       rcall  ClearLine2
       rcall  ShowPreset
       ret


ENCODERMODE1:
       lds    temp1,preset                ;get current preset#
       tst    encoder                     ;which way did encoder turn?
       brmi   e11
       cpi    temp1,MaxPreset             ;CW rotation
```

```
        brge   e12                       ;hard stop at Max Preset
        inc    temp1                     ;go to next higher preset
        rjmp   e12
e11:    cpi    temp1,0                   ;CCW rotation
        breq   e12                       ;hard stop at 0
        dec    temp1                     ;go to next lower preset
e12:    clr    encoder                   ;ignore any more requests
        sts    preset,temp1              ;save current preset#
        rcall  LoadEEMem                 ;get the preset in LCD buffer
        rcall  ShowPreset                ;and display it
        ret


TAPUP1:
        rcall  LoadNewFreq               ;DDS output new frequency
        rcall  ClearLine2
        ldi    temp1,0
        rcall  ChangeMode                ;go to mode 0 = normal op.
        ret

HOLDDOWN1:
        ldi    temp1,2                   ;go to next mode
        rcall  ChangeMode
        ret

HOLDUP1:
        rcall  InitMode1
        ret

;***********************************************************
;*  W8BH - MODE 2 (SAVE NEW PRESET) ROUTINES
;***********************************************************

ENCODERMODE2:
        lds    temp1,preset              ;get current preset#
        tst    encoder                   ;which way did encoder turn?
        brmi   e21
        cpi    temp1,MaxPreset           ;CW rotation
        brge   e22                       ;hard stop at Max Preset
        inc    temp1                     ;go to next higher preset
        rjmp   e22
e21:    cpi    temp1,0                   ;CCW rotation
        breq   e22                       ;hard stop at 0
        dec    temp1                     ;go to next lower preset
e22:    clr    encoder                   ;ignore any more requests
        sts    preset,temp1              ;save current preset#
        rcall  ShowPresetNum             ;display preset number
        ret

TAPUP2:
        lds    temp1,preset
        rcall  SaveEEmem                 ;save preset to EEPROM
        ldi    temp1,9
        rcall  DisplayLine2              ;display 'SAVED'
        ldi    temp1,2
        rcall  Blink_LED                 ;blink for user feedback
        ldi    temp1,0
        rcall  ChangeMode                ;return to tuning mode
        ret

HOLDDOWN2:
;       called when leaving this mode
        ldi    temp1,0                   ;escape to tuning mode
```

```
        rcall  ChangeMode
        ret

HOLDUP2:
;       called when this entering this mode
        rcall  ClearLine2                ;erase line 2
        rcall  ShowMemFreq               ;show freq left side of line2
        ret

;*******************************************************
;*  W8BH - MODE 3 (TESTING) ROUTINES
;*******************************************************

TapDown3:
        ldi    temp1,4
        rjmp   dd1
TapUp3:
        ldi    temp1,5
        rjmp   dd1
HoldDown3:
        ldi    temp1,6
        rjmp   dd1
HoldUp3:
        rcall  ClearLine2
        ldi    temp1,7
dd1:    rcall  QuickBlink
        rcall  DisplayLine2
        ret

;*******************************************************
;*  W8BH - KEYPAD ROUTINES
;*******************************************************
;
;  KEYPAD CONNECTIONS (7 wires)
;  Row1 to PB5, Row2 to BP4,
;  Row3 to PB3, Row4 to PB2,
;  Col0 to PD7, Col1 to PB1, Col2 to PB0
;
;  FUNCTIONS
*  # = cursor right
;  * = frequency presets.


KEYPAD:
        tst    encoder                   ;is encoder busy?
        brne   kp0                       ;wait for encoder to finish
        cbi    PORTD,PD7                 ;take column1 low
        ldi    temp1,2                   ;col1 value is 2
        rcall  ScanRows                  ;see if a row went low
        sbi    PORTD,PD7                 ;restore column1 high

        cbi    PORTB,PB0                 ;take column2 low
        ldi    temp1,1                   ;col2 value is 1
        rcall  ScanRows                  ;see if a row went low
        sbi    PORTB,PB0                 ;restore col2 high

        cbi    PORTB,PB1                 ;take column3 low
        ldi    temp1,0                   ;col3 value is 0
        rcall  ScanRows                  ;see if a row went low
        sbi    PORTB,PB1                 ;restore column3 high
kp0:    ret
```

```
SCANROWS:
       clc                             ;clear carry
       sbis   pinB,PB5                 ;is row1 low?
       subi   temp1,3                  ;yes, subtract 3
       sbis   pinB,PB4                 ;is row2 low?
       subi   temp1,6                  ;yes, subtract 6
       sbis   pinB,PB3                 ;is row3 low?
       subi   temp1,9                  ;yes, subtract 9
       sbis   pinB,PB2                 ;is row4 low?
       subi   temp1,12                 ;yes, subtract 12
       brcc   kp1                      ;no carry = no keypress
       neg    temp1                    ;negate answer
       rcall  PadCommand               ;do something
kp1:ret


PADCOMMAND:
       cpi    temp1,11                 ;special case: is it 0?
       brne   kp2                      ;no, continue
       ldi    temp1,0                  ;yes, replace with real zero

kp2:   cpi    temp1,12                 ;special case: "#" command?
       brne   kp3                      ;no, try next command
       inc    press                    ;emulate button press = cursor right
       ldi    temp1,1                  ;1 blink for switch debouncing
       rjmp   kp6                      ;done with '#'

kp3:   cpi    temp1,10                 ;special case:"*" command
       brne   kp4                      ;no, try next command
       rcall  LoadNextPreset           ;yes, get next preset
       rjmp    kp6                     ;done with '*'

kp4:   mov    temp2,StepRate           ;no, get current cursor position
       ldi    ZH,high(rcve0)           ;point to frequency value in memory
       ldi    ZL,low(rcve0)            ;16 bits, so need two instructions
kp5:   dec    ZL                       ;advance through frequency digits
       dec    temp2                    ;and advance through cursor positions
       brpl   kp5                      ;until we get to current digit
       ld     temp3,Z                  ;load value at cursor
       sub    temp1,temp3              ;subtract from keypad digit
       mov    encoder,temp1            ;set up difference for encoder routines.
       inc    press                    ;advance cursor position
kp6:   ldi    delay,150                ;simple key debouncer
       rcall  wait                     ;give the LED a rest!
       ret


;**********************************************************
;*  W8BH - FREQUENCY PRESET ROUTINES
;**********************************************************

ZeroMagic:
       ldi    ZH,high(rcve0)           ;point to magic#
       ldi    ZL,low(rcve0)
       ldi    temp1,0
       st     Z+,temp1                 ;zero first byte (MSB)
       st     Z+,temp1                 ;zero second byte
       st     Z+,temp1                 ;zero third byte
       st     Z+,temp1                 ;zero fourth byte (LSB)
       ret

ShowMagic:
       ldi    ZH,high(rcve0)           ;point to magic number
```

```
        ldi    ZL,low(rcve0)              ;2 byte pointer
        ldi    temp3,4                    ;counter for 4 byte display
        ldi    temp1,$80                  ;display on line1
        rcall  LCDCMD
sh1:    ld     temp1,Z+                   ;point to byte to display
        rcall  SHOWHEX                    ;display first nibble
        ldi    temp1,' '                  ;add a space
        rcall  LCDCHR                     ;display the space
        dec    temp3                      ;all 4 bytes displayed yet?
        brne   sh1                        ;no, so do the rest
        ret

AddMagic:
;       adds one component to magic according to StepRate
;       0 = Hz rate, 3=Khz rate, 6=MHz rate, 7=10MHz rate
        rcall IncFreq9
        ret

BuildMagic:
        push   StepRate                   ;save StepRate
        ldi    XH,high(LCDrcve0)          ;point to LCD digits
        ldi    XL,low(LCDrcve0)           ;16bit pointer
        ldi    StepRate,7                 ;Start with 10MHz position
bm1:    ld     temp3,X+                   ;get next LCD digit
        tst    temp3                      ;is it zero?
        breq   bm3                        ;yes, so go to next digit
bm2:    rcall  AddMagic                   ;no, so add magic component
        dec    temp3                      ;all done with this component
        brne   bm2                        ;no, add some more
bm3     :dec   StepRate                   ;all done with freq. positions?
        brne   bm1                        ;no, go to next (lowest) position
        pop    StepRate                   ;restore StepRate
        ret

LoadPMmem:
        ldi    ZH,high(freqLCD*2)         ;point to the preset table (-8 bytes)
        ldi    ZL,low(freqLCD*2)          ;16bit pointer
lp1:    adiw   ZL,8                       ;point to next frequency preset
        dec    temp1                      ;get to the right preset yet?
        brne   lp1                        ;no, keep looking
        ldi    YH,high(LCDrcve0)          ;yes, point to LCD digits
        ldi    YL,low(LCDrcve0)           ;16bit pointer
        ldi    temp2,8                    ;there are 8 frequency digits
lp2:    lpm    temp1,Z+                   ;get an LCD digit from FLASH mem
        st     Y+,temp1                   ;and put into LCD display buffer
        dec    temp2                      ;all digits done?
        brne   lp2                        ;not yet
        ret

LoadNewFreq:
        rcall  ZeroMagic                  ;clear out old magic number
        rcall  BuildMagic                 ;build new one based on current freq
        rcall  Freq_Out                   ;send new magic to DDS
;       rcall  ShowMagic                  ;show magic# on line 1 (debugging)
;nf1    :tst   encoder                    ;wait for encoder
;       breq   nf1
        ret

LoadNextPreset:
        lds    temp1,preset
        cpi    temp1,MaxPreset
        brne   ln1
        clr    temp1
```

```
        rjmp   ln2
ln1:    inc    temp1
ln2:    sts    preset,temp1
        rcall  LoadEEMem                  ;get preset from EE
        rcall  LoadNewFreq                ;update DDS with new freq
        rcall  ShowTuning                 ;display it
        ret


;**********************************************************
;*  W8BH - Timer 2 Overflow Interrupt Handler
;**********************************************************
;       This handler is called every 8 ms @ 20.48MHz clock
;       Increments HOLD counter (max 128) when button held
;       Resets HOLD counter if button released before hold met
;       Sets hold & down flags in button state register.

OVF2:
        push   temp1
        in     temp1,SREG                 ;save status register
        push   temp1
        ldi    temp1,90                   ;256-90=160; 160*50us = 8ms
        sts    TCNT2,temp1                ;reduce cycle time to 8 ms
        tst    hold                       ;counter at max yet?
        brmi   ov1                        ;not yet
        sbic   pinD,PD3
        clr    hold                       ;if button is up, then clear
        sbis   pinD,PD3
        inc    hold                       ;if button is down, then count
ov1:    pop    temp1
        out    SREG,temp1                 ;restore status register
        pop    temp1
        reti

;**********************************************************
;*  W8BH - External Interrupt 1 Handler
;**********************************************************
;       This handler replaces the original EXT_INT1 code
;       It is called when a logic-level change on the
;       external interrupt 1 (pushbutton) pin occurs.
;       Press is incremented on button-down events.
;       Release is incremented on button-up events.

EINT1:
        push   temp1                      ;save temp1 register
        in     temp1,SREG
        push   temp1                      ;save status register
        lds    temp1,EICRa                ;get interrupt control register
        sbrs   temp1,2                    ;bit2: rising edge =0, falling edge =1
        rjmp   ei1
                                          ;here is the falling-edge code
        cbr    temp1,$04                  ;falling edge '11' -> rising edge '10'
        inc    release                    ;count the button-up
        rjmp   ei2
                                          ;here is the rising-edge code
ei1:    sbr    temp1,$04                  ;rising edge '10' -> falling edge '11'
        inc    press                      ;count the button-down

ei2:    sts    EICRa,temp1                ;save interrupt control register
        pop    temp1
        out    SREG,temp1                 ;restore status register
        pop    temp1                      ;restore temp1 register
        reti
```

```
;*******************************************************
;*  W8BH - External Interrupt 0 Handler
;*******************************************************
;      This handler replaces the original EXT_INT0 code
;      It is called when a logic-level change on the
;      external interrupt 0 (encoder state) pin occurs.
;      Press is incremented on button-down events.
;      Release is incremented on button-up events.

EINT0:
       push   temp1                     ;save temp1 register
       in     temp1,SREG                ;save the status register
       push   temp1
       lds    temp1,EICRA               ;get current interrupt mode
       sbrs   temp1,0                   ;is mode rising-edge?
       rjmp   i02                       ;no, so go to falling edge (bit0=0)
       cbr    temp1,$01                 ;yes, clear bit 0
       sts    EICRA,temp1               ;change mode to falling-edge
       sbis   PIND,PHASE                ;is PHASE=1?
       rjmp   i01                       ;no, increase encoder (CW rotation)
       dec    encoder                   ;yes, decrease encoder (CCW rotation)
       rjmp   i04
i01:   inc    encoder
       rjmp   i04
i02:                                    ;current mode = falling-edge
       sbr    temp1,$01                 ;set bit 0
       sts    EICRA,temp1               ;change mode to rising-edge
       sbis   PIND,PHASE                ;is PHASE=1?
       rjmp   i03                       ;no, decrease encoder (CCW rotation)
       inc    encoder                   ;yes, increase encoder (CW rotation)
       rjmp   i04
i03:   dec    encoder
i04:   pop    temp1
       out    SREG,temp1                ;restore the status register
       pop    temp1                     ;restore temp1 register
       reti


;*******************************************************
;*  W8BH - Message Display routines
;*******************************************************


;DISPLAYMSG:
;      displays a null-terminated message on line 1
;      call with pointer to message in Z

;      ldi    temp1,$80                 ;use line 1
;      rcall  LCDCMD
;      rcall  DISPLAY_LINE              ;display the message
;      ldi    StepRate,3                ;put cursor at KHz posn
;      rcall  ShowCursor
;      ret

DISPLAYLINE1:
;      displays a 16-character msg on line 1
;      call with msg# in temp1

       mov    temp2,temp1
       ldi    temp1,$80                 ;use line 1
       rcall  LCDCMD
       rcall  DISPLAY16                 ;send 16 characters
```

```
        ret

DISPLAYLINE2:
;       displays a 16-character msg on line 2
;       call with msg# in temp1

        mov     temp2,temp1
        ldi     temp1,$C0                   ;use line 2
        rcall   LCDCMD
        rcall   DISPLAY16                   ;send 16 characters
        ret

DISPLAY16:
;       displays a 16-character msg
;       call with msg# in temp2

        ldi     ZH,high(messages*2-16)
        ldi     ZL,low(messages*2-16)
di1:    adiw    Z,16                        ;add 16 for each message
        dec     temp2                       ;add enough?
        brne    di1                         ;no, add some more
        ldi     temp3,16                    ;16 characters
di2:    lpm     temp1,Z+                    ;get the next character
        rcall   LCDCHR                      ;put character on LCD
        dec     temp3                       ;all 16 chars sent?
        brne    di2                         ;no, so repeat
        ret

CLEARLINE2:
        ldi     temp1,$C0                   ;point to second display line
        rcall   LCDCMD
        ldi     temp3,16                    ;16 characters to write
cl1:    ldi     temp1,' '
        rcall   LCDCHR                      ;write a blank space
        dec     temp3                       ;all 16 written?
        brne    cl1                         ;not yet
        ret

SHOWDECIMAL:
;displays a number 00-99 on the LCD
        clr     temp2                       ;10's counter
sd1:    cpi     temp1,10                    ;at least 10 remaining?
        brlo    sd2                         ;no, done counting 10's
        inc     temp2                       ;count the next 10
        subi    temp1,10                    ;remove the next 10
        brpl    sd1                         ;loop until all 10's gone
sd2:    mov     temp3,temp1                 ;save 10's counter
        mov     temp1,temp2
        rcall   ShowDec                     ;display 10's digit
        mov     temp1,temp3                 ;get 1's digit
        rcall   ShowDec                     ;and display it
        ret

SHOWMEMFREQ:
;       Displays the frequency in a more compact form: 'XX.XXXXXX'
        ldi     temp1,$C5                   ;Line 2, after preset number
        rcall   LCDCMD                      ;move cursor
        ldi     ZH,high(LCDrcve0)           ;point to frequency buffer
        ldi     ZL,low(LCDrcve0)            ;16 bit address
        ld      temp1,Z+                    ;get firstdigit from the buffer
        rcall   ShowDec                     ;and display it
        ld      temp1,Z+                    ;get second digit (MHz)
        rcall   ShowDec                     ;and display it.
```

```
        ldi    temp1,'.'                 ;decimal point
        rcall  LCDCHR
        ldi    temp2,6                   ;6 digits after decimal
cf1:    ld     temp1,Z+                  ;get next frequency digit
        rcall  SHOWDEC                   ;and display it
        dec    temp2                     ;all 6 done?
        brne   cf1                       ;not yet
        ret

SHOWPRESETNUM:
;       displays current preset '#xx' on line2
        ldi    temp1,$C0                 ;start of line2
        rcall  LCDCMD                    ;move cursor
        ldi    temp1,'#'                 ;display '#'
        rcall  LCDCHR
        lds    temp1,preset              ;get preset number
        rcall  ShowDecimal               ;and display it
        ret

SHOWPRESET:
        rcall  ShowPresetNum             ;first show preset number
        rcall  ShowMemFreq               ;then show preset frequency
        ret

SHOWTUNING:
        rcall  ClearLine2                ;erase second line
        rcall  ShowFreq                  ;format = "xx,xxx,xxx Hz"
        ldi    StepRate,3                ;put cursor at KHz position
        rcall  ShowCursor
        ret


;**********************************************************
;*  W8BH - EEPROM routines
;**********************************************************
;Data is transferred to/from temp1 (single byte) or Z (multiple bytes)
;EE address must be put into Y prior to call
;See ATMEL application note "AVR100"

.equ          SigByte1 = 'B'             ;first signature byte
.equ          SigByte2 = 'H'             ;second signature byte

ReadEE:
        sbic   EECR,EEPE                 ;busy writing EEPROM?
        rjmp   ReadEE                    ;yes, so wait
        out    EEARH,YH                  ;set up address reg.
        out    EEARL,YL
        sbi    EECR,EERE                 ;strobe the read bit
        in     temp1,EEDR                ;get the data
        ret

WriteEE:
        sbic   EECR,EEPE                 ;busy writing EEPROM?
        rjmp   WriteEE                   ;yes, so wait
        out    EEARH,YH                  ;set up address reg.
        out    EEARL,YL
        out    EEDR,temp1                ;put data in data reg.
        cli                              ;dont interrupt the write
        sbi    EECR,EEMPE                ;master write enable
        sbi    EECR,EEPE                 ;strobe the write bit
        sei                              ;interrupts OK now
        ret
```

```
Read8E:                                 ;read 8 bytes from EE
       ldi    temp2,8                   ;counter=8
r81:   rcall  ReadEE                    ;get byte from EE
       st     Z+,temp1                  ;move byte to destination
       adiw   Y,1                       ;go to next EE addr
       dec    temp2
       brne   r81
       ret

Write8E:                                ;write 8 bytes to EE
       ldi    temp2,8                   ;counter=8
r82:   ld     temp1,Z+                  ;get byte from source
       rcall  WriteEE                   ;store byte in EE
       adiw   Y,1                       ;go to next EE addr
       dec    temp2
       brne   r82
       ret

ProgramEE:
;      copy default memories from program FLASH to EE
       ldi    temp1,8                   ;'EEPROM STARTUP'
       rcall  DisplayLine1              ;display it
       clr    YH
       clr    YL                        ;start with EEPROM byte 00
       ldi    temp1,SigByte1            ;load first signature byte
       rcall  WriteEE                   ;write it
       inc    YL                        ;go to byte 1
       ldi    temp1,SigByte2            ;load second signature byte
       rcall  WriteEE                   ;write it
       ldi    temp2,30                  ;create space for 30 values
pe1:   inc    YL                        ;go to next EERPOM byte
       clr    temp1
       rcall  WriteEE                   ;write a 0 to EEPROM
       dec    temp2                     ;all 30 byte written?
       brne   pe1                       ;loop until all written
       ldi    temp2,MaxPreset*8+8       ;load # of preset bytes
       ldi    ZH,high(presets*2)        ;point to preset bytes
       ldi    ZL,low(presets*2)
pe2:   lpm    temp1,Z+                  ;get byte from program memory
       adiw   Y,1                       ;go to next EE addr
       rcall  WriteEE                   ;store byte in EE
       dec    temp2                     ;all preset bytes written?
       brne   pe2                       ;loop until all written
       ret

CheckEE:
;      looks to see if EE has been loaded with default presets
;      if not, defaults are programmed into the EE
       clr    YH
       clr    YL                        ;go to byte 00
       rcall  ReadEE                    ;look at first signature byte
       cpi    temp1,SigByte1            ;is it correct?
       brne   ee1                       ;no, so store defaults
       inc    YL                        ;go to byte 01
       rcall  ReadEE                    ;look at second signature byte
       cpi    temp1,SigByte2            ;is it correct?
       brne   ee1                       ;no, so store defaults
       rjmp   ee2                       ;signature byte OK, so done
ee1:   rcall  ProgramEE                 ;write defaults to EE
ee2:   ret


LoadEEMem:
```

```
;       specify the preset# in temp1
;       will return the EE memory into LCDrcve0
        clr     YH
        ldi     YL,32                       ;start of presets
        ldi     ZH,high(LCDrcve0)           ;point to frequency buffer
        ldi     ZL,low(LCDrcve0)
ge0:    cpi     temp1,0                     ;are we at zero yet?
        breq    ge1                         ;yes, so pointer correct
        adiw    Y,8                         ;add 8 for each preset
        dec     temp1                       ;finished counting?
        brne    ge0                         ;no, so continue counting
ge1:    rcall   Read8E                      ;read preset from EEPROM
        ret

SaveEEMem:
;       specify the preset# in temp1
;       will save frequency in LCDrcve0 to EE
        clr     YH
        ldi     YL,32                       ;start of presets
        ldi     ZH,high(LCDrcve0)           ;point to frequency buffer
        ldi     ZL,low(LCDrcve0)
se0:    cpi     temp1,0                     ;are we at zero yet?
        breq    se1                         ;yes, so pointer correct
        adiw    Y,8                         ;add 8 for each preset
        dec     temp1                       ;finished counting?
        brne    se0                         ;not yet
se1:    rcall   Write8E                     ;save preset to EEPROM
        ret


;**********************************************************
;*  W8BH - END OF INSERTED CODE
;**********************************************************
```

```
; The following goes at the end of the source code:

;*********************************************
;*
;*    USER-ADDED FREQUENCY PRESETS
;*
;*********************************************

.equ   MaxPreset = 19

;20 user-defined presets can be specified here
;Enter the values that you want to store into EEPROM

presets:                          ;One line for each preset freq
.db 0,3,5,6,0,0,0,0               ;80M qrp calling =  3.560 MHz
.db 0,7,0,3,0,0,0,0               ;40M qrp calling =  7.030 MHz
.db 1,0,0,0,0,0,0,0               ;WWV              = 10.000 MHz
.db 1,0,1,0,6,0,0,0               ;30M qrp calling = 10.106 MHz
.db 1,4,0,6,0,0,0,0               ;20M qrp calling = 14.060 MHz
.db 1,8,0,9,6,0,0,0               ;17M qrp calling = 18.096 MHz
.db 2,1,0,6,0,0,0,0               ;15M qrp calling = 21.060 MHz
.db 2,4,9,0,6,0,0,0               ;12M qrp calling = 24.906 MHz
.db 2,8,0,6,0,0,0,0               ;10M qrp calling = 28.060 MHz
.db 0,1,0,0,0,0,0,0               ;              = 01.000 MHz

.db 0,2,0,0,0,0,0,0               ;              = 02.000 MHz
.db 0,3,0,0,0,0,0,0               ;              = 03.000 MHz
.db 0,4,0,0,0,0,0,0               ;              = 04.000 MHz
.db 0,5,0,0,0,0,0,0               ;              = 05.000 MHz
.db 0,6,0,0,0,0,0,0               ;              = 06.000 MHz
.db 0,7,0,0,0,0,0,0               ;              = 07.000 MHz
.db 0,8,0,0,0,0,0,0               ;              = 08.000 MHz
.db 0,9,0,0,0,0,0,0               ;              = 09.000 MHz
.db 1,0,0,0,0,0,0,0               ;              = 10.000 MHz
.db 1,2,3,4,5,6,7,8               ;Test freq     = 12.345 MHz


messages:
.db "VFO Tuning Mode "            ;1
.db "Scroll Presets  "            ;2
.db "Save New Preset "            ;3
.db "Mode 4          "            ;4
.db "Mode 5          "            ;5
.db "Mode 6          "            ;6
.db "Mode 7          "            ;7
.db "EEPROM STARTUP  "            ;8
.db " SAVED          "            ;9
```