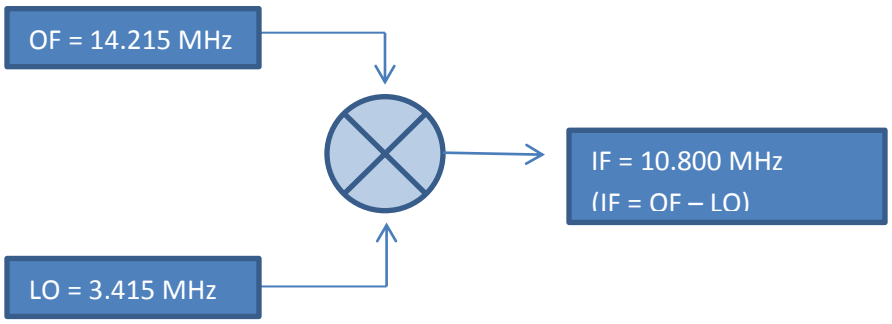
How to add IF offsets to your DDS Development Kit

By Bruce Hall, W8BH

I like direct conversion receivers. They are simple to understand, simple to build, and sound really nice. But sometimes we need the selectivity of a superhet. The mixer in a superhet requires us to consider not one, but at least three different radio frequencies: the operating frequency (OF), the local oscillator frequency (LO), and their product, the intermediate frequency (IF). Here is an example mixer for 20M:



In the above example, the operating frequency is 14.215 MHz and the fixed IF frequency is 10.8 MHz. To tune a station at 14.215 MHz we'll set our VFO at 3.415 MHz. We want to use our DDS kit for the VFO, of course, but we have to enter a number that doesn't resemble the operating frequency at all. Ugh.

We can simplify the process by letting our smart DDS handle all of the calculations. It will show us the operating frequency, but output the required LO frequency. In this example, we offset the operating frequency by -10.8 MHz to get the VFO frequency. The term "IF offset" is often used, but not quite correct: in some cases, there is no fixed "offset" number that will transform the operating frequency into the correct LO frequency.

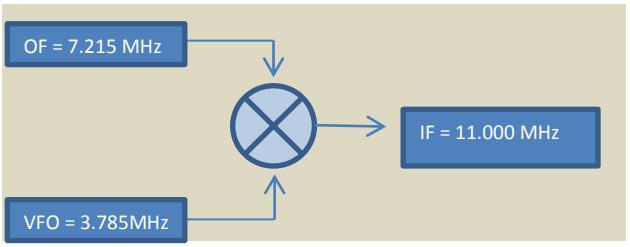
In general, there are two mixer products: the sum and difference of the inputs. We only want one signal, so we filter out the other. The remaining mixer output is the IF. There are three ways of combining the inputs to get the IF:

a) Additive. The IF = OF + LO.
b) Subtractive, Low-side injection. The IF = OF – LO (like our example).
c) Subtractive, High-side injection. The IF = LO – OF.

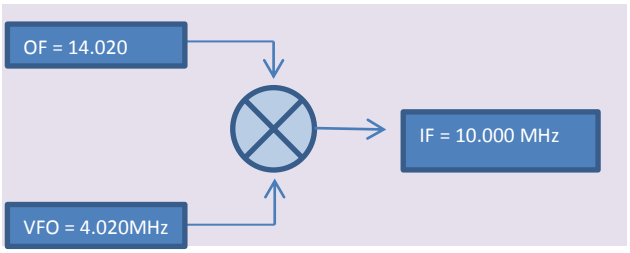
Here is an example of each method.

A) Additive:
   a. OF = 7.215 MHz
   b. VFO = 3.785 MHz
   c. IF = OF + VFO = 11 MHz

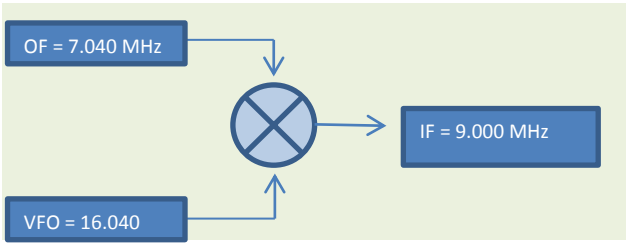
   As OF increases, VFO decreases.



B) Subtractive, Low-side injection:
   a. OF = 14.020 MHz
   b. VFO = 4.020 MHz
   c. IF = OF – VFO = 10 MHz



C) Subtractive, High-side injection
   a. OF = 7.040 MHz
   b. VFO = 16.040 MHz
   c. IF = VFO – OF = 9 MHz



This mixer stuff is basic ham knowledge, I know. But I was amazed at how quickly I got stumped when trying to figure out what the VFO output frequencies should be. I needed to write out some examples. Once I did, the answer was simple algebra. For each method above:

a) Additive: VFO = IF – OF
b) Subtractive, Low-side injection: VFO = OF – IF
c) Subtractive, High-side injection: VFO = OF + IF

Notice that the idea of a fixed offset does not work for the additive “A” mixer. There is no number, added or subtracted from the operating frequency that will give you the VFO frequency. For me, the term “IF offset” can be misleading.

Great, now how do we code it?? I am sure there are many ways. Diz gives us a very clever way to do offsets with his original code. Unfortunately it can only add or subtract from the operating frequency, so were out of luck if we have an additive (above, Type A) mixer. I decided to focus on using manipulating frequency ‘magic numbers’ instead. I chose them because we’ll ultimately need to do some addition and subtraction of frequencies, which are bigger numbers than our 8-bit microcontroller can handle. At least the magic numbers, which correspond to frequency, are already in a compact (32bit) binary form. If we try to do math on the frequencies digits themselves, we’ll have to come up with some math routines for large decimal numbers. Too much work. It’s left to the reader as an exercise!

For a discussion about magic numbers, please see my tutorial on AD9834 programming at http://w8bh.net/avr/AD9834.pdf. When dealing with mixers we need to keep track of 3 different frequencies (operating frequency, VFO frequency, and the IF). So our code will need to keep track of three different sets of magic numbers. Here is a table of names for my frequency/magic number pairs:

Frequency	Magic Number
Operating Frequency “LCDrcve0”	“rcve0”
VFO output frequency – not named	“VFOmn”
Intermediate Frequency “IFfreq”	“IFmn”

The operating frequency and its magic number were named by Diz in his original code, and have not been changed. Funny names, eh? I added the other two. Using a top-down approach, let’s create routines for a) additive, b) subtractive, high-injection and c) subtractive, low-injection mixers using the VFO frequency equations above. We’ll just use the X, Y, and Z registers to point to the magic numbers, and hope that we can program the adding and subtracting later. Here is the code for the three IF routines:

```
IFModeA:
;     additive mixer
;     mixer equation: operating freq OF + VFO = IF
;     In this mode, VFO goes down when OF goes up.

      ldi   XH, high(IFmn)          ;point to IF magic#
      ldi   XL, low(IFmn)
      ldi   YH, high(rcve0)         ;point to OF magic#
      ldi   YL, low(rcve0)
      ldi   ZH, high(VFOmn)         ;point to VFO magic#
      ldi   ZL, low(VFOmn)
      rcall Comp32                  ;is OF>IF?
      brlo  if1                     ;yes, so turn off
      rcall Sub32                   ;no, set VFO = IF - OF
      rjmp  if2
if1:  rcall Clear32
if2:  ret
```

```
IFModeB:
;      subtractive mixer, with low-side injection
;      mixer equation: OF - VFO = IF

       ldi   XH, high(rcve0)          ;point to OF magic#
       ldi   XL, low(rcve0)
       ldi   YH, high(IFmn)           ;point to IF magic#
       ldi   YL, low(IFmn)
       ldi   ZH, high(VFOmn)          ;point to VFO magic#
       ldi   ZL, low(VFOmn)
       rcall Comp32                   ;is OF<IF?
       brlo  if3                      ;yes, so turn off
       rcall Sub32                    ;no, set VFO = OF - IF
       rjmp  if4
if3:   rcall Clear32
if4:   ret

IFModeC:
;      subtractive mixer, with high-side injection
;      mixer equation: VFO - OF = IF

       ldi   XH, high(IFmn)           ;point to IF magic#
       ldi   XL, low(IFmn)
       ldi   YH, high(rcve0)          ;point to OF magic#
       ldi   YL, low(rcve0)
       ldi   ZH, high(VFOmn)          ;point to VFO magic#
       ldi   ZL, low(VFOmn)
       rcall Add32                    ;set VFO = OF + IF
       ret
```

That wasn't hard at all. Just point to what you want to add/subtract, point where you want the result, and call a function (Add32/Sub32) to do the numerical mixing. We postponed the calculation part. I searched for ways to add and subtract big numbers. It turns out that 32 bit binary numbers are a cinch. Atmel, maker of our microcontroller chip, published an application note on how to do it. Look for AVR202 which is now at http://www.atmel.com/dyn/resources/prod_documents/doc0937.pdf. All we need to do is a single-byte addition/subtraction, and then use the carry bit to continue to calculation to the next byte. We can extend this for as many bytes we need. In our case, the 32-bit magic numbers are four bytes long. Here are some math routines, using the X, Y, and Z pointers, that work on a single byte at a time:

```
SUBBYTE:
;      subtracts byte at Y from byte at X, with carry
;      result put in byte at Z
;      used for 32-bit substraction routine

       ld    temp1,X+
       ld    temp2,Y+
       sbc   temp1,temp2              ;subtract Y from X
       st    Z+,temp
       ret

ADDBYTE:
;      adds byte at Y to byte at X, with carry
;      result put in byte at Z
;      used for 32-bit addition routine
```

```
      ld    temp1,X+
      ld    temp2,Y+
      adc   temp1,temp2             ;add Y to X
      st    Z+,temp1
      ret

COMPBYTE:
;     compare bytes at X and Y
;     return flags compatible with branch instructions

      ld    temp1,X+
      ld    temp2,Y+
      cpc   temp1,temp2
      ret
```

The carry bit is preserved by ST (store), RET (return) and RCALL, so any carry will be intact between the end of one add/subtract/compare and start of the next one. With these building blocks in place, doing the four-byte math is as easy as calling each routine four times. For example,

```
ADD32:
;     adds the 4-byte value at Y to the 4-byte value at X
;     stores the four byte result at Z

      clc                           ;clear the carry bit
      rcall AddByte                 ;add 1st bytes (LSB)
      rcall AddByte                 ;2nd bytes
      rcall AddByte                 ;3rd bytes
      rcall AddByte                 ;4th bytes (MSB)
      ret
```

Routines for 4-byte subtract and compare are exactly the same. The hard part is done. Pick the IF mode that you want, plug in a value for the IF magic#, do the math, and send the result to the DDS. It works great. Then Tom, AK2B, reminded me that the mixer, while important for receiving, is often not in the transmit chain. Could we have the IF offset active during receive only? In the keyer articles, all of the dit and dah logic eventually funnels down to two routines, KeyUp and KeyDown, which control the state of the Key output I/O line. This seems like a good spot to control the IF calculation. If we are KeyDown, then it's time to transmit and the IF is turned off. If we are KeyUp, then the receiver is active and the IF should be turned back on.

I tried a number of schemes to make the IF turn on and off. Some of them worked. They all looked messy. Then I remembered that the DDS chip contain two different frequency registers. We can load one of them with the receive frequency, and the other with the transmit frequency. When keying, all we need to do is toggle between them. Here are the modified KeyUp and KeyDown routines:

```
KEYDOWN:
      rcall DDSOutputB              ;change to transmit freq
      sbi   PortD,KeyOut            ;turn on output line
      cbi   PortC,LED               ;turn on LED
      ret
```

```
KEYUP:

      cbi   PortD,KeyOut            ;turn off output line
      sbi   PortC,LED               ;turn off LED
      rcall DDSOutputA              ;change to receive freq
      ret
```

The only remaining issue is providing a user interface for selecting the mixer type and entering the IF itself. The values will be stored in EEPROM, rather than in code, so that they can be changed later without recompiling. Here is a chart of how I use the encoder and button for IF settings.

MODE 5 (IF SETTINGS)	**Select mixer submode (A)**	**Edit IF submode (B)**
Encoder	Allow the user to scroll through the mixer types	Allow the user to edit the digit at the current cursor position
Button Tap	Select the displayed mixer type, and go to edit submode to edit the IF.	Advance cursor to next IF digit
Button Hold	Go to next mode (6)	Save the IF and go back to VFO tuning mode

The first four routines, which handle events from the encoder and the pushbutton, branch to submode A (select mixer type) or submode B (enter the IF) depending on the current state of the ‘flags’ variable:

```
ENCODERMODE5:

      lds   temp1,flags
      sbrs  temp1,2                 ;check for alternate submode
      rjmp  Encoder5A
      rjmp  Encoder5B

TAPUP5:

      lds   temp1,flags
      sbrs  temp1,2                 ;check for alternate submode
      rjmp  TapUp5A
      rjmp  TapUp5B

HOLDDOWN5:

      lds   temp1,flags
      sbrs  temp1,2                 ;check for alternate submode
      rjmp  HoldDown5A
      rjmp  HoldDown5B

HOLDUP5:
```

```
      lds   temp1,flags
      sbrs  temp1,2                 ;check for alternate submode
      rjmp  HoldUp5A
      rjmp  HoldUp5B
```

The remaining routines handle the events. In submode A, rotating the encoder scrolls through the 3 mixer options plus an option for no mixer. These are numerically coded 0 through 3:

```
ENCODER5A:

      ldi   temp2,0                 ;set lower limit
      ldi   temp3,3                 ;set upper limit
      lds   temp1,IFmode            ;get current IF mode
      rcall EncoderValue            ;update speed based on encoder
      sts   IFmode,temp1            ;save new speed value
      rcall ShowIFmode              ;display it
      ret
```

A tap will select the displayed mixer option and save it to EEPROM. Unless 'No mixer' was chosen, the submode will switch to B and allow the user to enter the IF:

```
TAPUP5A:

      lds   temp1,IFmode            ;which IF mode was selected?
      tst   temp1                   ;are we IFmode0 = no mixer?
      breq  Exit5                   ;no mixer, so save & exit
      rcall SetAltMode              ;mixer, so get IF freq
      rcall Init5B                  ;set up display first
      ret
```

In Submode B, the encoder is used to increment/decrement the IF digits. The code for this is a close copy of the original source code, except that it does not alter our DDS output frequency. A button tap advances the cursor by calling original code, too

```
ENCODER5B:                          ;See EncoderMode0 for description

      tst   encoder
      brpl  e52                     ;which way did encoder rotate?
      inc   encoder                 ;remove 1 negative rotation
      rcall DecFreq0                ;reduce displayed frequency
      cpi   temp1,55                ;55 = all OK
      brne  e51
      rcall IncFreq0                ;correct freq. underflow
      rjmp  e55
e51:  rcall DecFreq9                ;reduce magic number
      rjmp  e54
e52:  dec   encoder                 ;remove 1 positive rotation
      rcall IncFreq0                ;increase displayed frequency
      cpi   temp1,55                ;55 = all OK
      brne  e53
      rcall DecFreq0                ;correct freq. overflow
      rjmp  e55
e53:  rcall IncFreq9                ;increase magic number
e54:  rcall ShowFreq                ;display new frequency
e55:  rcall QuickBlink
      ret
```

```
TAPDOWN5:

      rcall TapDown0            ;cursor advance
      ret
```

The mode is exited by holding down the button. On exit, the magic number for the entered frequency is calculated, the value is saved, and the VFO output is updated based on the mixer type and offset.

That's pretty much it. I skipped a few small parts, focusing on code that relates directly to implementing IF. Information on programming the DDS registers, storing variables in EEPROM, handling user interface events, and keyer routines are all covered in some of my previous articles. Have Fun!

Other DDS articles

- Keypad tutorial: http://w8bh.net/avr/AddKeypadFull.pdf
- VFO Memory Project: http://w8bh.net/avr/AddMemories.pdf
- Extending Encoder Button Functionality: http://w8bh.net/avr/ButtonEvents.pdf
- How to use the EEPROM: http://w8bh.net/avr/EEPROM.pdf
- A Simple Iambic Keyer: http://w8bh.net/avr/IambicKeyer.pdf
- A Memory Keyer: http://w8bh.net/avr/MemoryKeyer.pdf
- A Programmable Keyer: http://w8bh.net/avr/MemoryKeyerII.pdf

Source Code

I haven't included the full source code because it is 'in transition'. Each time I copy text between AVR studio and Word I lose some of the formatting. It takes time to adjust margins, tabs, fonts, etc. It also is difficult for others to copy code from this pdf file back into AVR studio without losing the formatting. To fix these problems I will post the assembler code as a text file, separate from the PDF.