Using Fractional-N Synthesizers for Generating Multi-Frequency (MFSK) Datamodes

Andy Talbot G4JNT May 2014

The Fractional-N synthesizer is a convenient solution to generating an almost arbitrary output frequency capable of being set to a fine tuning resolution, and that is locked to an input reference. Here is not the place to describe how FractN synthesis works in detail – a web search will throw up a lot of information on this; References [1] and [2] are good starting points. Suffice to say, the output frequency of a FractN synthesizer chip is given by :

$F_{OUT} = F_{REF} / R * (N + F/D) * M$ {1}

Where F_{REF} is the input (eg. a 10MHz reference clock), R is the reference divider, N, F and D are internal registers that are user programmable over, usually, quite a wide range. N, F, D and R must all be integers. M is either an output divider, sometimes included within the chip, or subsequent RF multiplication. Therefore M can take on either an integer value (in the case of external multiplication) , or a rational (1/integer) value in the case of an output divider

FractN synthesis employs a relatively high comparison frequency, given by F_{REF} / R and a 10MHz input is often used directly by setting R = 1. Dithering the N divider , using of the values in F and D gives the fractional term. The dither happens at comparison frequency speeds, and its resulting spurii are removed in the PLL filtering process.

Examining the basic equation {1} shows that the tuning grid, or output setting resolution, is given by :

$F_{STEP} = F_{REF} * M / (R . D)$ rearranging for a specific F_{STEP} gives $D = F_{REF} * M / (R . F_{STEP})$

The D, or denominator register can usually be programmed over quite a large range of values. The LMX2xxx family, for example, has 22 bits of resolution allowing numbers up to approximately 4 million. This offers the possibility of generating arbitrarily small frequency increments by reprogramming the F (or Fractional) register with values that are changed in unit steps for each frequency increment desired.

As an example, lets look at generating the JT65B datamode at 144.657MHz with an output divider of 10 (Most FractN chips only work at UHF and upwards, so it is usual to generate at 1 - 2GHz and divide down). JT65B requires that one of 65 tones frequencies be generated with a spacing between tones of 5.3833Hz

If we want to maintain a comparison frequency of 10MHz, that means that R = 1. M = 1/10, or 0.1 and D therefore has to be 185760 when rounded to the nearest integer . To generate as near to 144.657MHz as we can get, N = 144 so F/D = 0.657 and F must therefore be 0.657 * 122044 to get an F/D of 0.657.

Plug these values into the fundamental FractN equation above to verify the results. Note that incrementing the value of F from this base value by a number in the range 1 to 65 increases the output frequency in steps of 5.3833Hz.

Which is the key to MFSK datamode generation.   D is selected to give the frequency step, then the tone number is added to the base value of F and the chip reprogrammed in real time with this modified F value to generate the tone sequence at the final output frequency.

Frequency Setting Accuracy

Where a beacon or source has to be returned to an exact frequency during the carrier or idle period, it is usually necessary to reprogramme the D register with a value that allows a "nice" rounded number. For example, staying with the values calculated for the JT65B mode above, the output frequency when 144.657MHz is requested is actually, 144.656998Hz or 1.7Hz low.   Not too critical, perhaps, but not perfect.   But by reprogramming D with a value of 100000 will give 10Hz steps and programming a value of 65700 into F then gives the exact output wanted.

Programming steps using the D register

Some of the most convenient synthesizer chips come with an internal VCO; a good example is the LMX2541. To optimise the VCO tuning and allow lower MHz/V tuning sensitivity, the internal VCO is preset close to the wanted frequency. To allow this to occur, the chip goes though an internal calibration routine every time a new frequency is programmed. This process is normally completely transparent to the user and takes just a few clock cycles of the reference input.   In normal operation this is not normally a problem, but when the chip is being repeatedly reprogrammed in real time, several times per second, wideband transients that are present for a short period during the recalibration period can be observed.

This was seen on the breadboard LMX2541 device intended for the next generation of 432MHz beacons and had to be sorted out!   The data sheet states that a recalibration process is initiated every time the '0' register is programmed. This is usually the final, or only register that is set when frequency is changed and carries part of the F and part of the N register words.   Therefore a recalibration procedure was being initiated every time a new tone frequency was requested. VCO calibration is only needed for large frequency excursions and certainly is not necessary with minor shifts such as those needed here. But the VCO calibration process can't be switched off!

The solution adopted was to generate the tone shifts by reprogramming the D register in real time instead of the F.   The D value is carried in its entirety in another register in the LMX2541 device, 'Reg 2' . When this one alone is changed, it does not trigger a recalibration process and hence no wideband transients.   However, determining starting values such that changing the value of D in unit intervals to give precise frequency steps is not so easy.   To derive starting values of D and F so that a unit change of D alone shifts the frequency by one step requires a bit of algebraic manipulation of the equation :

Start with the fundamental equation given at the beginning, then differentiate this to get an equation relating the change, or F_{STEP}, to a unit change in D.

$DF_{OUT} / dD \quad = F_{STEP} \;=\; - F_{REF} * M * F / (R * D^2)$

This is not nearly so nice !   For one thing the direction of frequency shift is negative so for a tone frequency increase, the tone number has to be subtracted from the base value in D. Also the equation still has both F and D in it!   However, we know the <u>ratio</u>   F/D must remain constant to give the required base frequency.   So by substituting this ratio in place of F and *one of the* D terms, the equation becomes:

$F_{STEP} = F_{REF} * M * \underline{F/D} \; / \; (R * D)$ Ignoring the minus sign. Rearranging gives :

$D = F_{REF} * M * \underline{F/D} \; / \; (R * F_{STEP})$

Using the same values for the example above, a step of 5.3833Hz therefore requires D = 122044

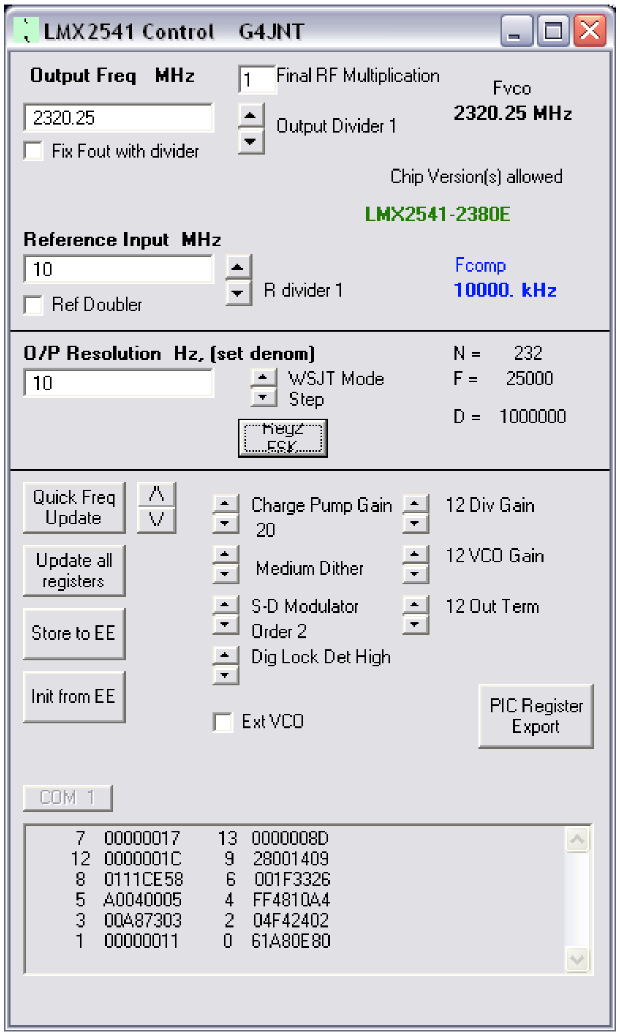
Now, $\underline{F/D}$ = 0.657 for 144.657MHz, so for this value of D, calculate F = 0.657 * 122044 = 80182

Plug F = 80182 , D = 122044, N = 144, R = 1 and M = 0.1 into {1} and verify 144.657MHz is generated. Subtract one from the value of D, and note the tone frequency increases by 5.3833Hz

(and it is no coincidence that the value of D just happens to be the same as was needed for F in the original calculation!)

LMX2541_Prog.exe

This utility for calculating the register values has now been updated to allow Reg2 FSK programming.



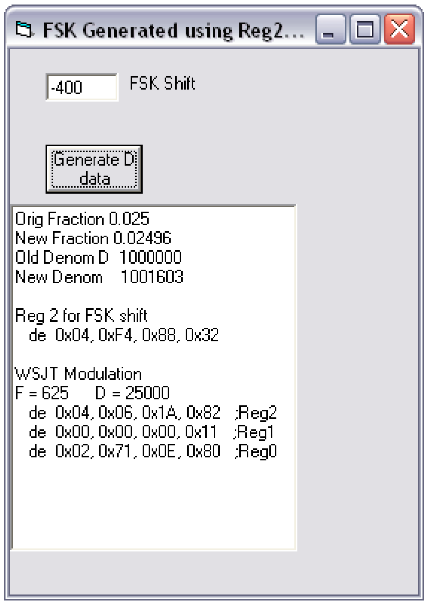
Output Freq MHz 2320.25 Final RF Multiplication 1 Fvco 2320.25 MHz
Output Divider 1
Chip Version(s) allowed LMX2541-2380E
Reference Input MHz 10 R divider 1 Fcomp 10000. kHz
O/P Resolution Hz, (set denom) 10
N = 232
F = 25000
D = 1000000
Charge Pump Gain 20 12 Div Gain
Medium Dither 12 VCO Gain
S-D Modulator Order 2 12 Out Term
Dig Lock Det High

```
 7  00000017    13  0000008D
12  0000001C     9  28001409
 8  0111CE58     6  001F3326
 5  A0040005     4  FF4810A4
 3  00A87303     2  04F42402
 1  00000011     0  61A80E80
```

Start the programme as normal, using the WSJT mode pull down menu to select the correct frequency shift, and generate the register values for conventional setting.

Once done, click the new Reg2 FSK button. A new window will appear. For FSK CW enter the frequency shift in the top box (the default is -400Hz space or key-up tone). Click the Generate D data button.

The new values for Reg2/1/0 will be shown in a format suitable for copy and pasting into the PIC assembly code.



FSK Shift -400

```
Orig Fraction 0.025
New Fraction 0.02496
Old Denom D  1000000
New Denom    1001603

Reg 2 for FSK shift
  de  0x04, 0xF4, 0x88, 0x32

WSJT Modulation
F = 625     D = 25000
  de  0x04, 0x06, 0x1A, 0x82   ;Reg2
  de  0x00, 0x00, 0x00, 0x11   ;Reg1
  de  0x02, 0x71, 0x0E, 0x80   ;Reg0
```

At the time of writing, only PIC code for JT65 modulation has been written using Reg2 programming.

[1] http://www.ti.com/lit/an/swra029/swra029.pdf

[2] http://www.ti.com/lit/an/snaa062a/snaa062a.pdf