Multisynth Notes

Let the calculated Multisynth value have the format of whole numerator/denominator like 45 1/5 where 45 is whole, 1 is numerator, and 5 is denominator. If there is no fractional portion for the calculated Multisynth, set the denominator to 1 and the numerator to 0. The calculated Multisynth value cannot be less than 4.

Multisynth Conversions

These Multisynth conversions are valid for Multisynth 0 to 3 and N.

Convert from Calculated Multisynth Divider to Multisynth Registers

If the calculated Multisynth value is zero (which is 0 0/1), CLKn_INTDIV, CLKn_NUM, and CLKn_DEN = 0.

```
CLKn_INTDIV = floor( (whole * denominator + numerator) * 128 / denominator ) – 512
CLKn_NUM = floor( (whole * denominator + numerator) * 128 ) % denominator
CLKn_DEN = denominator
```

Note: % is the modulus operation.

Convert from the Multisynth Registers to the Calculated Multisynth Divider

If CLKn_INTDIV, CLKn_NUM and CLKn_DEN are zero, divider is zero (which is 0 0/1).

```
divider =  ( CLKn_INTDIV + 512 + (CLKn_NUM / CLKn_DEN) ) / 128
```

Frequency Step Size Conversions

The range of the step size is 100Hz to 10MHz.

Let stepSizeHz be the value to move the output frequency in Hertz. Let foutHz be the output frequency in Hertz.

Convert from the frequency step size to the registers

```
stepIn100Hz = stepSizeHz / 100
foutIn100Hz = foutHz / 100

Un_DENINIT = foutIn100Hz * denominator
Un_DENSTEP = stepIn100Hz * denominator
Un_NUM = floor( (whole * denominator + numerator) * foutIn100Hz )
```

Convert from the registers to step size

Un_DENINIT cannot be zero.

```
stepSizeHz = (Un_DENSTEP * foutHz * Rn) / Un_DENINIT
```

Initial Phase Offset Conversions

The range is +/- 45 nanoseconds with 20 picosecond resolution. Use only 15 bits for the two's complement operations.

Let fvcoHz be the frequency of the VCO of the device in Hertz. Let offsetNs be the offset value in nanoseconds.

Convert from the desired offset to the phase offset register

```
resolution = (1 / fvcoHz) / 128
offsetSec = offsetNs / 1e9
resolutionCount = offsetSec / resolution
CLKn_PHASEOFFSET = twosComplement(resolutionCount)
```

Convert from the register to the offset value

```
resolution = (1 / fvcoHz) / 128
resolutionCount = twosComplementInverted(CLKn_PHASEOFFSET)
offsetSec = resolution * resolutionCount
offsetNs = offsetSec * 1e9
```

Phase Step Size Conversions

The step size range is 0 to 45 nanoseconds with 20 picosecond resolution. Let stepSizeNs be the step size in nanoseconds and fvcoHz is the frequency of the VCO in Hz.

Convert from the desired offset to the phase offset register

```
resolution = (1 / fvcoHz) / 128
offsetSec = offsetNs / 1e9
CLKn_PHASESTEPSIZE = offsetSec / resolution
```

Convert from the register to the offset value

```
resolution = (1 / fvcoHz) / 128
offsetSec = resolution * CLKn_PHASESTEPSIZE
offsetNs = offsetSec * 1e9
```

Input Configurations

CLKIN

Bit Field Name	Register Number	Differential	Single-Ended	Crystal
XOCLK_SEL	28	0	0	1
XO_ENABLE	28	0	0	1
RX_POWER_SE	28	0	1	X
REFMUXIN	29	0	1	X
NOTERMR_REF	116	1	1	1

Set XO_RATE according to the desired frequency if Crystal is used.

FDBK

Bit Field Name	Register Number	Differential	Single-Ended	Off
FBRX_POWER_SE	28	0	1	X
FBMUXIN	30	0	1	X
FDBK_PDN	28	0	0	1
NOTERMR_FB	106	1	1	1

Finding Multisynth and N Values for a Frequency Plan

Use this algorithm if the PLL is needed. This finds all combinations of Multisynth and N values within the fvco range that satisfies the frequency plan.

1. Find the first P1 divider that reduces the input frequency (CLKIN) to 40MHz or less. This is the phase detector input frequency.
2. Pick the most important output frequency among the four possible output frequencies for one plan. Use this to find the first Multisynth value later. The calculated Multisynth value will be an integer and the most optimized channel usually (unless, of course, the other Multisynth values are also integers).
3. For all the output frequencies, if an output frequency is less than 5MHz, find the first R divider value that increases the output frequency of the Multisynth to greater than 5MHz. Keep these R values. The goal is to get the actual Multisynth output frequencies and ensure they are in range.
4. Calculate the output frequency of the Multisynth for the most important frequency: best Multisynth output frequency = corresponding R value * desired output frequency
5. For all the possible even-numbered divider values between 4 and 1024:
    a. possible fvco = best Multisynth output frequency * current integer divider
    b. If the fvco value is in range, keep the divider value.
6. For all the divider values collected:
    a. fvco = best Multisynth output frequency * current divider value
    b. N = fvco / phase detector frequency
    c. For all the other Multisynth output frequencies:
        i. Calculate the corresponding Multisynth value: Multisynth = fvco / (Multisynth output frequency * corresponding R divider value)
        ii. If the current Multisynth value is less than 8 and not 4 or 6, then this plan is invalid! Do not keep. Otherwise, it's a valid plan.

Variation: The above algorithm optimizes for one output frequency (see number 2). However, more solutions can be found if each output frequency is the optimized one. Duplicates of some solutions are possible.

Guidelines to Pick the Best Plan

These help with reducing jitter and power.

1. Use N divider values that are integers as much as possible.
2. Look for a Multisynth value that is an integer on important output channels or frequencies.
3. Pick plans with VCO frequencies close to 2.5GHz.
4. Enable Multisynth 0 fanout mode when possible, understanding the advantages and disadvantages.

Using Fractions to Store the Multisynth Values

One possible problem, when implementing this algorithm, is loosing precision or introducing rounding errors in the calculations. To prevent this and to better model the operation of the Multisynths, use a fraction data structure where the type will have three parts: integer or whole number, numerator and denominator. Using 64-bit unsigned numbers means that there is plenty of precision in the fraction itself. Additionally, use this data structure also to store the input and output frequencies, N, VCO frequency, and phase detector frequency. Operations like addition, subtraction, division, multiplication, simplifying the fraction, and even comparisons like equals, greater than and less than are critical to making this data structure successful in its implementation and usage in the algorithm.