Verilog Implementation of Oscillators

Manikanta Krishnamurthy

June 22, 2024

Abstract

In this report, I will provide a comprehensive explanation of jitter and phase noise in clock generation, detailing its significance and impact on system performance. I will begin by implementing the initial code provided, ensuring a clear understanding of its functionality and purpose. Following this, I will conduct a thorough analysis of phase noise, identifying its sources and quantifying its effects. I will then explore various techniques to mitigate phase noise, discussing their implementation and efficacy. The report will end with the implementation, and phase noise analysis of an 8-phase oscillator using Verilog and Python. This in-depth examination will demonstrate practical methods for reducing phase noise and improving the overall performance of clock generation systems.

1 Jitter and Phase Noise

1.1 Jitter

The need for high-speed communications infrastructure is driving strong demand for high-frequency reference signal sources that provide stable output signals. One of the indicators used to evaluate the stability of these output signal waveforms is called "jitter."
Waveforms of digital signals appear as a bright line on an oscilloscope. This bright line, which should oscillate at regular intervals, sometimes appears thick. This widening of the line is evidence of jitter. Figure 1 shows a single period, or cycle, of a signal in which multiple different periods are evident. An

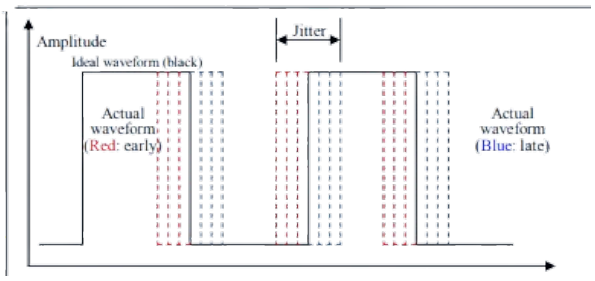


Figure 1: The concept of jitter as indicator for evaluating the output waveform of a reference signal source .

ideal waveform repeats an invariable cycle. Actual waveforms, however, vary in the time domain, with signal edges rising or falling earlier (red) or later (blue) than they are supposed to.
Jitter is the result of time-domain fluctuations in digital signals, but jitter comes in a lot of different types.
There are many types of jitter, including the following:

- Period jitter (peak-to-peak)
- RMS jitter
- Random jitter
- Deterministic jitter
- Accumulated jitter (long-term jitter)

1.2 Phase Noise

Crystal oscillators output frequency components outside the frequencies they are designed to output. As shown in Figure 2, the frequency characteristics of signals output by a crystal oscillator contain other frequencies in the vicinity of the fundamental frequency. This is produced by phase modulation due to random signals; that is, the noise source modulates the oscillator. Commonly called phase noise, these frequencies are almost always higher than the noise floor, appearing near the carrier frequency.

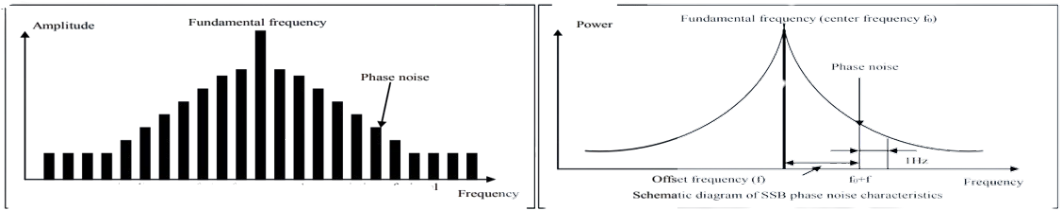


Figure 2: .

1.3 Theoretical calculations

1.3.1 Phase Noise

$$v(t) = A(t)\sin(\omega_0 t + \varphi(t)) \tag{1}$$
$$v(t) = A\left[\sin(\omega_0 t)\cos\varphi(t) + \cos(\omega_0 t)\sin\varphi(t)\right] \tag{2}$$
$$v(t) \approx A\sin(\omega_0 t) + A\varphi(t)\cos(\omega_0 t) \tag{3}$$

Autocorrelation function of the signal

$$R_v(t,\tau) = \frac{A^2}{2}\left\{\cos(\omega_0\tau) - \cos(2\omega_0 t + \omega_0\tau) + \left[\cos(2\omega_0 t + \omega_0\tau) + \cos(\omega_0\tau)\right] R_\varphi(\tau)\right\} \tag{4}$$

$$\bar{R}_v(\tau) = \frac{\omega_0}{\pi}\int_0^{\pi/\omega_0} R_v(t,\tau)dt = \frac{A^2}{2}\cos(\omega_0\tau)\left[1 + R_\varphi(\tau)\right] \tag{5}$$

Power spectral density (PSD) of the signal

$$S_v(f) = \frac{A^2}{4}\left[\delta(f - f_0) + \delta(f + f_0) + S_\varphi(f - f_0) + S_\varphi(f + f_0)\right] \tag{6}$$

Simplified PSD considering only one sideband

$$S'_v(f) = \frac{A^2}{2}\left[\delta(f - f_0) + S_\varphi(f - f_0)\right] \tag{7}$$

Alternatively, if we define f as offset from f_0 :

$$S_\varphi(f) = \frac{S_v(f_0 + f)}{A^2/4} = \frac{S'_v(f_0 + f)}{A^2/2} \tag{8}$$

The phase noise L_f at an offset frequency f from the carrier frequency f_0 is given by:

$$\mathcal{L}(f) = 10\log_{10}\left(\frac{S'_v(f_0 + f)}{A^2}\right) \text{ dBc/Hz} \tag{9}$$

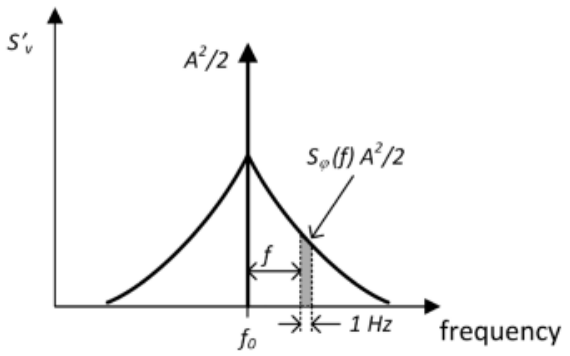


Figure 3: .

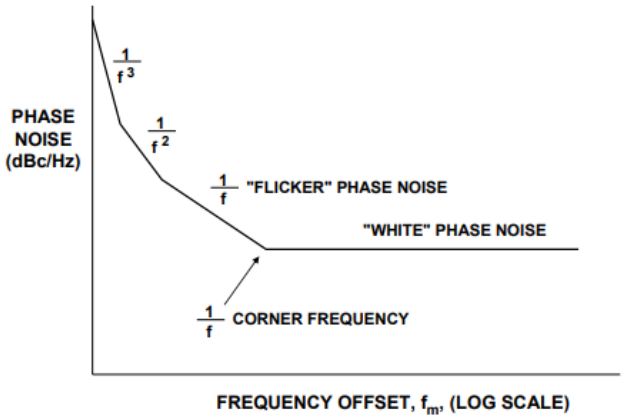


Figure 4: .

1.3.2 Jitter

The first step in calculating the equivalent rms jitter is to obtain the integrated phase noise power over the frequency range of interest, i.e., the area of the curve, A. The curve is broken into a number of individual areas (A1, A2, A3, A4), each defined by two data points.
Generally speaking, the **upper frequency range** for the integration should be **twice the sampling frequency.**

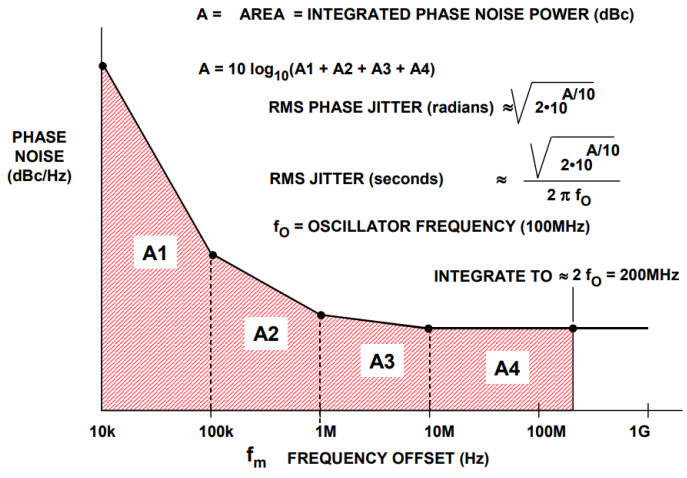


Figure 5: .

The integration of each individual area yields individual power ratios. The individual power ratios are then summed and converted back into dBc. Once the integrated phase noise power is known, the RMS phase jitter in radians is given by the equation :

$$\text{RMS Phase Jitter (radians)} \approx \sqrt{2 \cdot 10^{A/10}} \tag{10}$$

$$\tag{11}$$

and dividing by $2\pi f_O$ converts the jitter in radians to jitter in seconds:

$$\text{RMS Jitter (seconds)} \approx \frac{\sqrt{2 \cdot 10^{A/10}}}{2\pi f_O} \tag{12}$$

$$\tag{13}$$

1.4 Impact of Phase Noise

The effect of $\phi(t)$,on the output, in the frequency domain is shown in the image below(sinusoidal input):

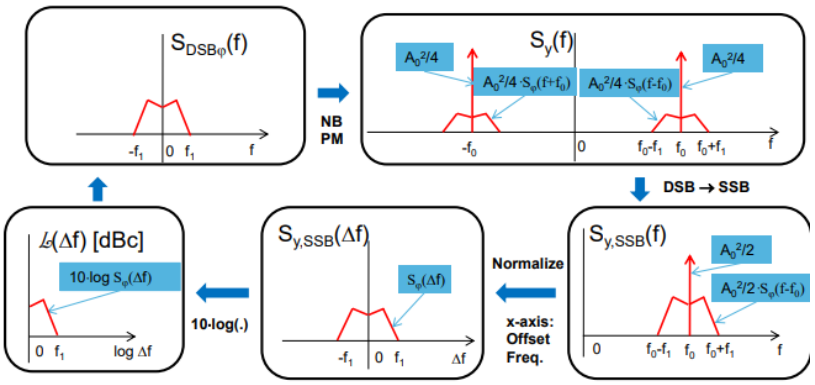


Figure 6: .

For example if $\phi(t)$ was sinusoidal of a particular frequency f_1,

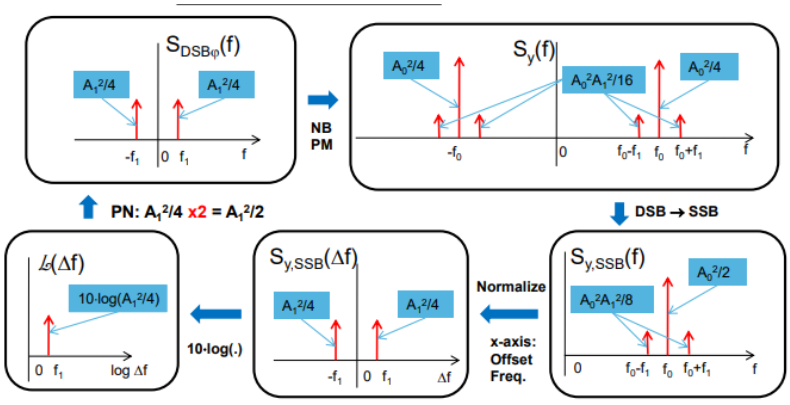


Figure 7: .

Now since noise is a composition of all kinds of frequency , the net effect of noise tends to change the frequency spectrum of output in the following manner:

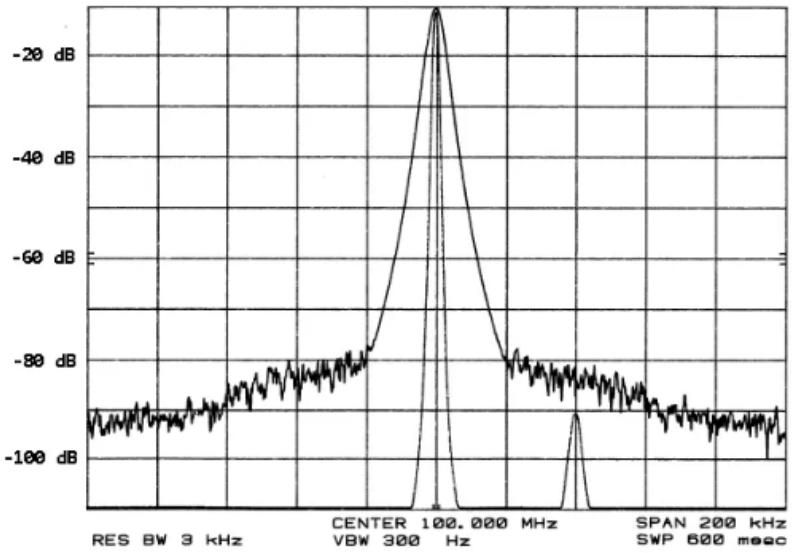


Figure 8: .

2 4-Phase Oscillator

2.1 Code

2.1.1 Module

```
`timescale 1ns/1fs
module Oscillator(clk, powerdown);
        //parameter
        parameter real d_buff = 0.05;
        parameter real f_ref = 2;
        parameter real p_ref = 1/(f_ref);

       //parameter for jitter
        parameter real j_pm_db = -160 ;
        parameter real j_pm_mag =
            (10**(j_pm_db/10)*(f_ref*1e9))**0.5/2/3.141592/f_ref;
        parameter real j_fm_frq = 1e-3;

        parameter real j_fm_db = -90 ;
        parameter real j_fm_mag =
            (10**(j_fm_db/10)/(f_ref*1e9))**0.5*j_fm_frq/f_ref*1e9/2**0.5;

       //register for jitter(frequency, phase)
        real j_fm = 0;
        real j_pm = 0;
        integer j_pm_seed = 222;
        integer j_fm_seed = 131;

        //variable for flicker
        real j_fl = 0;
```

```
        //input
        input wire powerdown;

        //output
        output reg[3:0] clk;

       //variables for ideal clock
        integer j_fl_seed = 121;
        reg [3:0] clk_i;

        always @(clk_i) begin
        j_pm <= 1e-6*($dist_normal(j_pm_seed,0,1000000))*j_pm_mag;
        j_fm <= 1e-6*($dist_normal(j_fm_seed,0,1000000))*j_fm_mag;
        j_fl <= 0.99996*j_fl +
            3e-10*($dist_normal(j_fl_seed,0,1000000))*j_fm_mag/2**0.5;
        end

        always @(clk_i[1]) begin
                clk[1]<= #(d_buff+j_pm) clk_i[1];
                clk[3]<= #(d_buff+j_pm) clk_i[3];
        end

        always @(clk_i[0]) begin
                clk[0]<= #(d_buff+j_pm) clk_i[0];
                clk[2]<= #(d_buff+j_pm) clk_i[2];
        end

        always @(clk_i[0] or powerdown) begin
                if(powerdown)
                begin
                 clk_i[1] <=1'B1;
                 clk_i[3] <=1'B0;
                end
        else
                begin
                 clk_i[1]<= #(p_ref/4+ j_fm/2**0.5+ j_fl) clk_i[0];
                 clk_i[3]<= #(p_ref/4+ j_fm/2**0.5+ j_fl) clk_i[2];
                end

        end

        always @(clk_i[1] or powerdown) begin
                if(powerdown)
                begin
                 clk_i[0] <=1'B1;
                 clk_i[2] <=1'B0;
                end
                else
                begin
                 clk_i[0]<= #(p_ref/4+ j_fm/2**0.5+ j_fl) clk_i[3];
                 clk_i[2]<= #(p_ref/4+ j_fm/2**0.5+ j_fl) clk_i[1];
                end
        end

endmodule
```

Listing 1: Verilog code for 4-Phase Oscillator module

2.1.2 Testbench

```
`timescale 1ns/1fs

module tb_Oscillator;

    //Inputs
    reg powerdown;

    //Outputs
    wire [3:0] clk;

    //variables
    integer f_clk_osc;

    //Unit Under Test
    Oscillator osc (
        .clk(clk),
        .powerdown(powerdown)
    );

    initial begin
        powerdown = 1;
        #100;
        powerdown = 0;
        #500;
        f_clk_osc = $fopen("./Measure/clk_osc_46.txt","w");
        #1800000;
        $finish;
        $fclose(f_clk_osc);
    end

    always @(posedge clk[3]) begin
        $fwrite(f_clk_osc, "%f\n", $realtime);
    end

endmodule
```

Listing 2: Verilog code for tb_Oscillator module

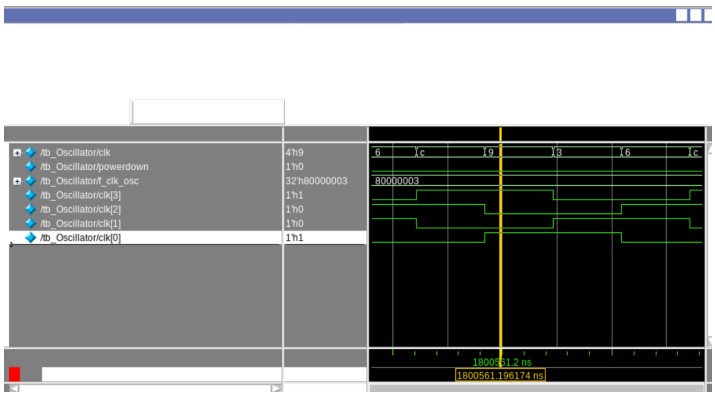


Figure 9: Output of 4-Phase Oscillator .

As we can see the consecutive clocks differ in phase by 90° .

2.1.3 Explanation

- **Initialization of Parameters**:
  - Parameters necessary for the calculation of jitter are initialized.
  - The desired frequency (`f_ref`) is set to 2GHz.
  - `j_pm_db` is declared as the area under the phase noise graph (in logarithmic scale).
- **Jitter Calculation**:
  - Jitter is calculated in seconds using a specific formula.
  - Jitter due to frequency modulation is defined (`j_fm_mag`).
- **Jitter Types**:
  - Three jitter values are defined:
    - Jitter due to phase modulation (`j_pm`).
    - Jitter due to frequency modulation (`j_fm`).
    - Jitter due to flicker noise (`j_fl`).
- **Seed Values**:
  - Seed values are provided to generate a specific sequence of random values.
- **Internal Clock (`clk_i`)**:
  - The four clocks are contained in the variable `clk_i`.
  - Whenever the internal clock (`clk_i`) changes, a new jitter is generated because, in real life, jitter varies for different cycles.
  - Jitter is generated using the `$dist_normal` function, which models the Gaussian behavior of real-life jitter. The jitter values are scaled accordingly.
- **Output Clock (`clk`)**:
  - The output clock (`clk`) is updated with the values of `clk_i`.
- **Powerdown Behavior**:
  - When `powerdown` changes, `clk_i` is initialized to `0011`.
- **Clock Updates**:
  - When `clk_i[0]` is updated, `clk_i[1]` and `clk_i[3]` are updated with a jitter delay based on the previous values of `clk_i[0]` and `clk_i[2]`, respectively.
  - When `clk_i[1]` is updated, `clk_i[0]` and `clk_i[2]` are updated with a jitter delay based on the previous values of `clk_i[3]` and `clk_i[1]`, respectively.
- **Phase Difference**:
  - Each part of the clock is updated after `p_ref/4` to introduce the phase difference between the clocks.
- **Initial State and Powerdown**:
  - The `powerdown` signal initializes the internal clock `clk_i` to `0011`.
  - The initial values for `clk_i` are `clk_i[0] = 1`, `clk_i[1] = 1`, `clk_i[2] = 0`, `clk_i[3] = 0`.
- **Clock Update Mechanism**:
  - The internal clock `clk_i` is updated in pairs:

    - * clk_i[1] and clk_i[3] are updated based on the values of clk_i[0] and clk_i[2].
    - * clk_i[0] and clk_i[2] are updated based on the values of clk_i[3] and clk_i[1].
  - This ensures a 90-degree phase shift between consecutive clock signals.

- **First Update Cycle**:
  - When clk_i[0] is updated, clk_i[1] and clk_i[3] are set to the values of clk_i[0] and clk_i[2] with a jitter delay.
  - Given clk_i = 0011, after the update:
    - * clk_i[1] = clk_i[0] = 1
    - * clk_i[3] = clk_i[2] = 0
  - The output clk also updates to match clk_i, resulting in clk = 0110.

- **Second Update Cycle**:
  - When clk_i[1] is updated, clk_i[0] and clk_i[2] are set to the values of clk_i[3] and clk_i[1] with a jitter delay.
  - Given clk_i = 0110, after the update:
    - * clk_i[0] = clk_i[3] = 0
    - * clk_i[2] = clk_i[1] = 1
  - The output clk updates to 1100.

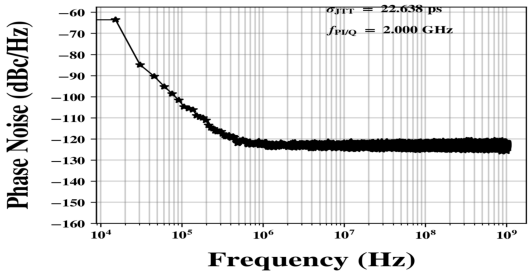
- **Third Update Cycle**:
  - When clk_i[0] is updated again, clk_i[1] and clk_i[3] are set to the values of clk_i[0] and clk_i[2] with a jitter delay.
  - Given clk_i = 1100, after the update:
    - * clk_i[1] = clk_i[0] = 0
    - * clk_i[3] = clk_i[2] = 1
  - The output clk updates to 1001.

- **Fourth Update Cycle**:
  - When clk_i[1] is updated again, clk_i[0] and clk_i[2] are set to the values of clk_i[3] and clk_i[1] with a jitter delay.
  - Given clk_i = 1001, after the update:
    - * clk_i[0] = clk_i[3] = 1
    - * clk_i[2] = clk_i[1] = 0
  - The output clk updates to 0011, completing the cycle.

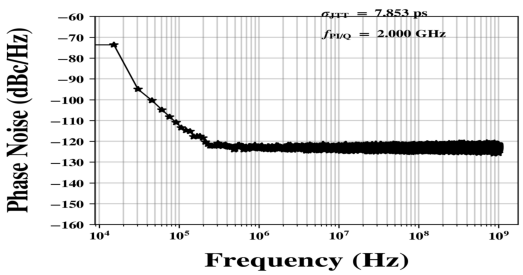
- **Repeat**:
  - The process repeats, maintaining the sequence: 0011 → 0110 → 1100 → 1001 → 0011.

2.2 Phase Noise Analysis

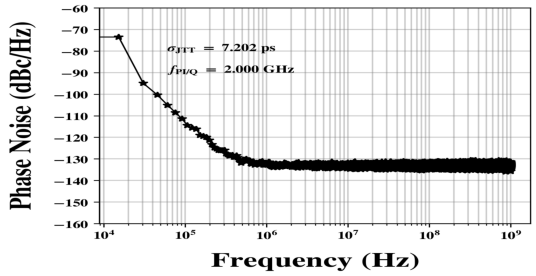
I have calculated the phase noise graph for different values of j pm db and j fm db respectively.Here are some of the examples:



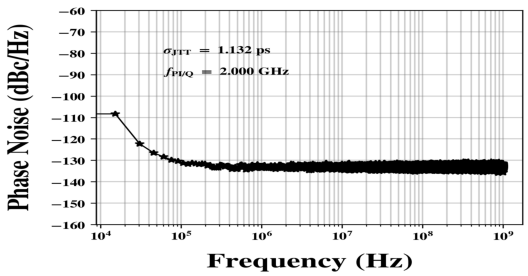
(a) -120,-130



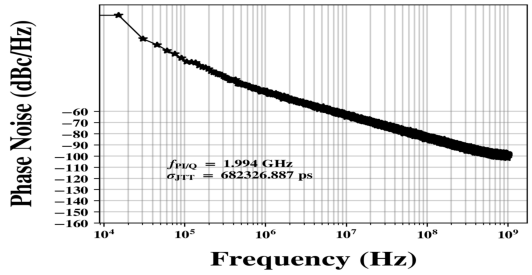
(b) -120,-140



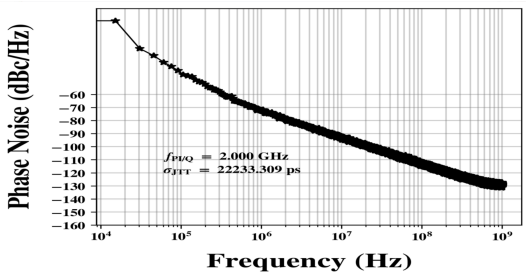
(a) -130,-140



(b) -130,-160



(a) -160,-40



(b) -160,-70

Figure 12: Phase noise graphs for different values of j_pm_db and j_fm_db

From these plots it is visible that the phase noise becomes constant after a certain frequency. This noise is called "White Phase Noise".
Changes in the modulation values affects the output frequency as well as the rms jitter value .

2.2.1 Data

First I am keeping the j_fm_db constant.

j_pm_db	**j_fm_db**	**rms_jitter (in ps)**
-85	-90	117722.435
-90	-90	42995.773
-95	-90	6998.606
-100	-90	2229.6
-120	-90	2224.209
-160	-90	2223.699
-170	-90	2223.695
-180	-90	2223.694
-190	-90	2223.694
-210	-90	2223.694
-310	-90	2223.694

Table 1: RMS Jitter Data for j_fm_db = -90

j_pm_db	**j_fm_db**	**rms_jitter (in ps)**
-100	-140	33.776
-110	-140	13.098
-120	-140	7.853
-130	-140	7.202
-150	-140	7.164
-160	-140	7.167
-170	-140	7.168

Table 2: RMS Jitter Data for j_fm_db = -140

j_pm_db	**j_fm_db**	**rms_jitter (in ps)**
-110	-130	24.729
-120	-130	22.638
-150	-130	22.517
-160	-130	22.520
-170	-130	22.521

Table 3: RMS Jitter Data for j_fm_db = -130

j_pm_db	**j_fm_db**	**rms_jitter (in ps)**
-110	-160	11.322
-120	-160	3.580
-130	-160	1.132
-160	-160	0.036
-170	-160	0.011

Table 4: RMS Jitter Data for j_fm_db = -160

Now keeping the value of j_pm_db constant.

j_pm_db	j_fm_db	rms_jitter (in ps)
-120	-80	7031.575
-120	-90	2224.206
-120	-100	703.606
-120	-110	222.611
-120	-120	70.134
-120	-130	22.638
-120	-140	7.853
-120	-160	3.580
-120	-180	3.580

Table 5: RMS Jitter Data for j_pm_db = -120

j_pm_db	j_fm_db	rms_jitter (in ps)
-160	-40	682326.887
-160	-70	22233.309
-160	-80	7031.122
-160	-85	3953.775
-160	-90	2223.699
-160	-100	702.949
-160	-110	222.258
-160	-120	70.280
-160	-130	22.520
-160	-140	7.167
-160	-160	0.036
-160	-180	0.036

Table 6: RMS Jitter Data for j_pm_db = -160

After looking through the data , we can notice the $\sqrt{10^{A/10}}$ dependence of the jitter time. We notice that when a reduction of 20 occurs in either j_pm_db or j_fm_db ,we notice the jitter reduces by ten times.
From the above data we notice there are many values for which we can achieve a RMS-jitter less than 30 picoseconds as required by the assignment.

Therefore selecting $j\_pm\_db = -120$ and $j\_fm\_db = -140$.

3 8-Phase Oscillator

Implementing an 8-Phase Oscillator is very similar to the 4-Phase Oscillator .

3.1 Code

3.1.1 Module

```
`timescale 1ns/1fs
module Oscillator(clk, powerdown);
        //parameter
        parameter real d_buff = 0.05;
        parameter real f_ref = 2;
        parameter real p_ref = 1/(f_ref);

       //parameter for jitter
        parameter real j_pm_db = -160 ;
```

```
 parameter real j_pm_mag =
     (10**(j_pm_db/10)*(f_ref*1e9))**0.5/2/3.141592/f_ref;
 parameter real j_fm_frq = 1e-3;

 parameter real j_fm_db = -90 ;
 parameter real j_fm_mag =
     (10**(j_fm_db/10)/(f_ref*1e9))**0.5*j_fm_frq/f_ref*1e9/2**0.5;

//register for jitter(frequency, phase)
 real j_fm = 0;
 real j_pm = 0;
 integer j_pm_seed = 222;
 integer j_fm_seed = 131;

 //variable for flicker
 real j_fl = 0;

 //input
 input wire powerdown;

 //output
 output reg[7:0] clk;

//variables for ideal clock
 integer j_fl_seed = 121;
 reg [7:0] clk_i;

 always @(clk_i) begin
 j_pm <= 1e-6*($dist_normal(j_pm_seed,0,1000000))*j_pm_mag;
 j_fm <= 1e-6*($dist_normal(j_fm_seed,0,1000000))*j_fm_mag;
 j_fl <= 0.99996*j_fl +
     3e-10*($dist_normal(j_fl_seed,0,1000000))*j_fm_mag/2**0.5;
 end

 always @(clk_i[1] or clk_i[3]) begin
         clk[1]<= #(d_buff+j_pm) clk_i[1];
         clk[3]<= #(d_buff+j_pm) clk_i[3];
         clk[5]<= #(d_buff+j_pm) clk_i[5];
         clk[7]<= #(d_buff+j_pm) clk_i[7];
 end

 always @(clk_i[0] or clk_i[2]) begin
         clk[0]<= #(d_buff+j_pm) clk_i[0];
         clk[2]<= #(d_buff+j_pm) clk_i[2];
         clk[4]<= #(d_buff+j_pm) clk_i[4];
         clk[6]<= #(d_buff+j_pm) clk_i[6];
 end

 always @(clk_i[0] or clk_i[2] or powerdown) begin
         if(powerdown)
         begin
          clk_i[1] <=1'B1;
          clk_i[3] <=1'B1;
          clk_i[5] <=1'B0;
          clk_i[7] <=1'B0;
         end
 else
         begin
          clk_i[1]<= #(p_ref/4+ j_fm/2**0.5+ j_fl) clk_i[0];
          clk_i[3]<= #(p_ref/4+ j_fm/2**0.5+ j_fl) clk_i[2];
          clk_i[5]<= #(p_ref/4+ j_fm/2**0.5+ j_fl) clk_i[4];
          clk_i[7]<= #(p_ref/4+ j_fm/2**0.5+ j_fl) clk_i[6];
```

```
                end

        end

        always @(clk_i[1] or clk_i[3] or powerdown) begin
                if(powerdown)
                begin
                 clk_i[0] <=1'B1;
                 clk_i[2] <=1'B1;
                 clk_i[4] <=1'B0;
                 clk_i[6] <=1'B0;
                end
                else
                begin
                 clk_i[0]<= #(p_ref/4+ j_fm/2**0.5+ j_fl) clk_i[7];
                 clk_i[2]<= #(p_ref/4+ j_fm/2**0.5+ j_fl) clk_i[1];
                 clk_i[4]<= #(p_ref/4+ j_fm/2**0.5+ j_fl) clk_i[3];
                 clk_i[6]<= #(p_ref/4+ j_fm/2**0.5+ j_fl) clk_i[5];
                end
        end

endmodule
```

Listing 3: Verilog code for 8-Phase Oscillator module

3.1.2 Testbench

```
`timescale 1ns/1fs
module eight_phase;

    //Inputs
    reg powerdown;

    //Outputs
    wire [7:0] clk;

    //variables
    integer f_clk_osc;

    //Unit Under Test
    Oscillator osc (
        .clk(clk),
        .powerdown(powerdown)
    );
    initial begin
        powerdown = 1;
        #100;
        powerdown = 0;
        #500;
        f_clk_osc = $fopen("./Measure/clk_osc_46.txt","w");
        #1800000;
        $finish;
        $fclose(f_clk_osc);
    end
    always @(posedge clk[3]) begin
        $fwrite(f_clk_osc, "%f\n", $realtime);
    end

endmodule
```

Listing 4: Verilog code for eight_phae module

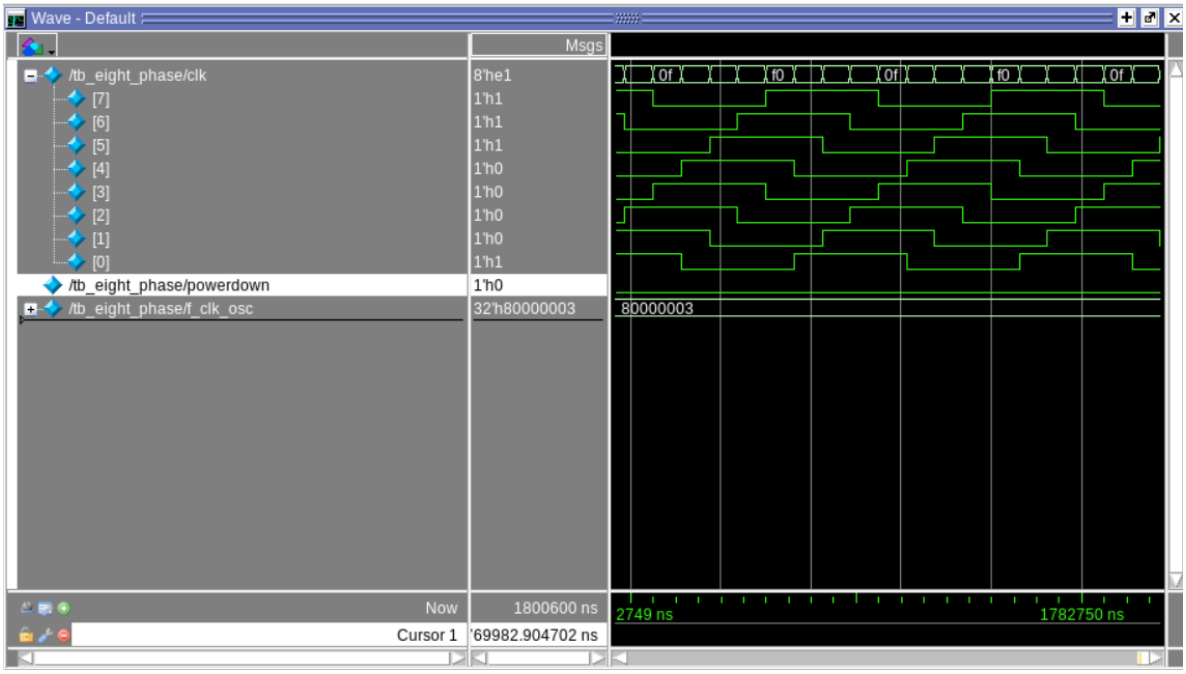
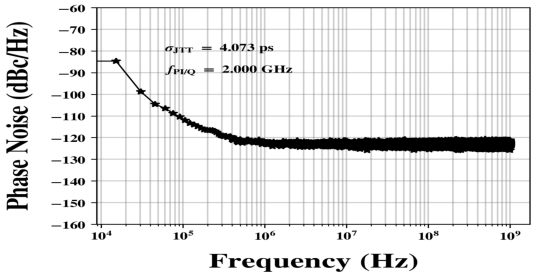


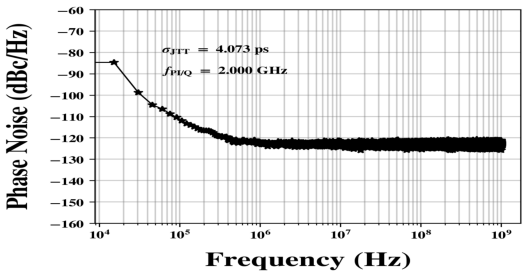
Figure 13: Output of 8-Phase Oscillator .

3.2 Phase Noise Analysis

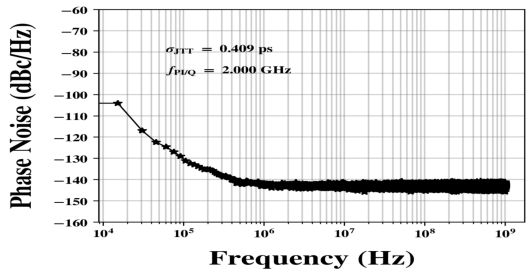
Some of the phase noise graphs for different values of j_pm_db and j_fm_db :



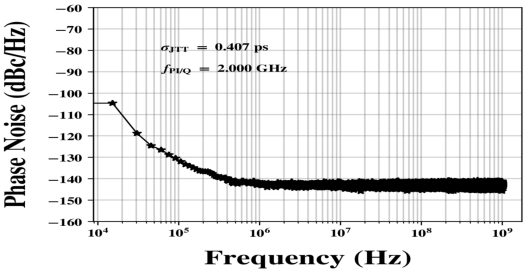
(a) -120,-160



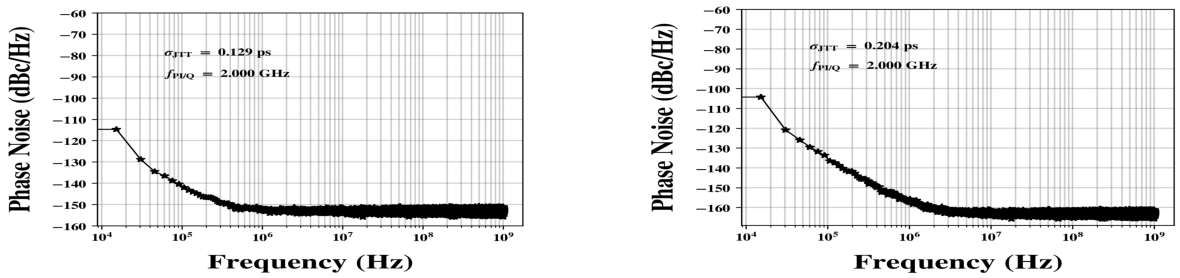
(b) -120,-170



(a) -140,-150



(b) -140,-160



(a) -150,-160 (b) -160,-150

Figure 16: Phase noise graphs for different values of j_pm_db and j_fm_db

3.2.1 Data

j_pm_db	j_fm_db	rms_jitter (in ps)
-120	-150	4.025
-120	-160	4.073
-120	-170	4.073

j_pm_db	j_fm_db	rms_jitter (in ps)
-130	-150	1.251
-130	-160	1.288

j_pm_db	j_fm_db	rms_jitter (in ps)
-140	-150	0.409
-140	-160	0.407
-140	-170	0.407

j_pm_db	j_fm_db	rms_jitter (in ps)
-150	-150	0.217
-150	-160	0.129

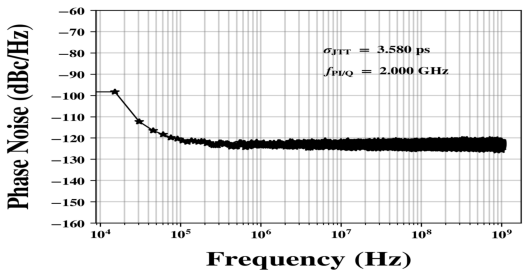
j_pm_db	j_fm_db	rms_jitter (in ps)
-160	-150	0.204
-160	-160	0.041
-160	-170	0.041

j_pm_db	j_fm_db	rms_jitter (in ps)
-170	-150	0.208
-170	-160	0.013
-170	-170	0.013

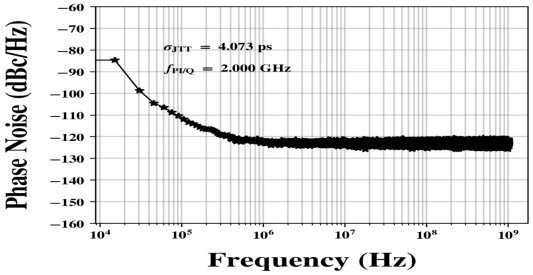
j_pm_db	j_fm_db	rms_jitter (in ps)
-180	-150	0.210
-180	-160	0.004
-180	-170	0.004

3.3 Phase Noise comparison with 4-Phase Oscillator

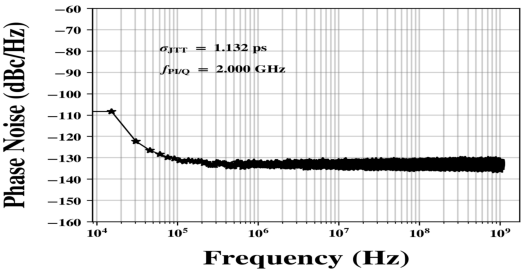
Comparing the phase noise plots of the two oscillators:



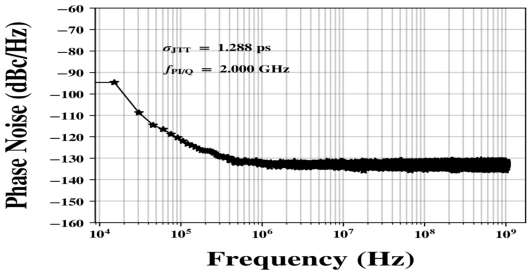
(a) -120,-160



(b) -120,-160



(a) -130,-160



(b) -130,-160

Figure 18: Phase noise graphs for 4-Phase and 8-Phase Oscillator respectively

From the above figures we see that the phase noise plot for the 8-Phase Oscillator is shifted upwards when compared to the 4-Phase Oscillator. Numerically this means that the magnitude of the phase noise has increased . This data makes more sense cause ,
In an eight-phase oscillator, there are eight distinct phases that contribute to the overall signal. Each phase can introduce its own phase noise due to imperfections in the oscillation mechanism, such as noise in the feedback network, non-linearities in the amplifiers, or thermal fluctuations.
In contrast, a four-phase oscillator has fewer phases (only four), which means there are fewer sources contributing to the overall phase noise.