问题1

在该问题中，我们找了几乎所有的关于延迟的计算方法、相关公式

In [1]:

```
import pylatex
import latexify
import numpy as np
import pandas as pd
import plotly.graph_objects as go
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = 'all'
InteractiveShell.ast_node_interactivity = 'last'

from sko import GA
from colorama import Fore
```

In [6]:

```
data = pd.read_csv('环形振荡器输出频率计算表.csv', index_col=0, encoding='gbk')
data
```

Out[6]:

	反相器个数/个	PMOS宽/m	NMOS宽/m	MOS长/m
0	11	4.000000e-07	2.000000e-07	1.000000e-07
1	11	8.000000e-07	4.000000e-07	2.000000e-07
2	11	1.600000e-06	8.000000e-07	4.000000e-07
3	31	2.000000e-07	4.000000e-07	1.000000e-07
4	31	4.000000e-07	8.000000e-07	2.000000e-07
5	31	8.000000e-07	1.600000e-06	4.000000e-07
6	51	5.000000e-07	5.000000e-07	1.000000e-07
7	51	1.000000e-06	1.000000e-06	2.000000e-07
8	51	1.800000e-06	1.800000e-06	3.000000e-07
9	99	2.000000e-06	1.000000e-06	5.000000e-07

In [7]:

```
# TODO 参数

VDD = 1.2  # V
```

```python
# Vt = 0.2 * VDD  # V
# Vtn = 0.2 * VDD  # V
# Vtp = Vtp_abs = 0.2 * VDD  # V
Vtp = 0.398  # V
Vtn = 0.42  # V

Vds = VDD  # V
Vds_Vdsat = 0.2  # V
Vdsat = Vds - Vds_Vdsat  # V
Vdsatn = Vds - Vtn  # V
Vdsatp = Vds - Vtp  # V

K_n = 111.6634  # μA/V^2
K_p = 68.7134  # μA/V^2
K_n = 111.6634e-6  # A/V^2
K_p = 68.7134e-6  # A/V^2
# K_n = 111.6634e-3  # A/V^2
# K_p = 68.7134e-3  # A/V^2

gate_len_min = 60  # nm
gate_len_min = 60e-9  # m

gate_wide_min = 120  # nm
gate_wide_min = 120e-9  # m

gate_len_max = 100000  # nm
gate_len_max = 100000e-9  # m

gate_wide_max = 100000  # nm
gate_wide_max = 100000e-9  # m

beta_n = lambda W, L, K_n=K_n: W / L * K_n  # A/V  # 223.2
beta_p = lambda W, L, K_p=K_p: W / L * K_p  # A/V  # 274.8

CL_ratio = 3.137  # pF/μm^2
CL_ratio = 3.137  # F/m^2
gate_covered_area = lambda W, L: W * L  # m^2 栅极覆盖的沟道面积
CL = lambda W_n, W_p, L: CL_ratio * gate_covered_area(W_n + W_p, L)  #  F 负载电容(正比于下一级反相器的栅极面积)

sd_area  =lambda W: 190e-9 * W  # m^2 源漏面积 / 栅极两侧的红色矩形面积
mos_area = lambda W, L: gate_covered_area(W, L) + sd_area(W)  # m^2 NMOS 或 PMOS 的面积
single_inverter_area = lambda W_n, W_p, L: mos_area(W_n, L) + mos_area(W_p, L) + 70e-9 * (L + 2 * 190e-9)  # m^2 单个反相器的面积
ring_oscillator = lambda W_p, W_n, L, n=51: N * single_inverter_area(W_n, W_p, L)  # 环形振荡器的面积

N, W_p, W_n, L = 51, 120e-9, 120e-9, 60e-9
```

t_r 和 t_f

公式1：

$$t_r = \frac{C_L}{k_p V_{DD}} \left[\frac{\alpha_p - 0.1}{(1-\alpha_p)^2} + \frac{arcth(1 - \frac{0.1}{1-\alpha_p})}{1-\alpha_p} \right]$$

$$t_f = \frac{C_L}{k_n V_{DD}} \left[\frac{\alpha_n - 0.1}{(1-\alpha_n)^2} + \frac{arcth(1 - \frac{0.1}{1-\alpha_n})}{1-\alpha_n} \right]$$

In [8]:

```
InteractiveShell.ast_node_interactivity = 'all'
# InteractiveShell.ast_node_interactivity = 'last'

alpha_p = Vtp / VDD
_alpha_p = 1 - alpha_p
alpha_n = Vtn / VDD
_alpha_n = 1 - alpha_n

t_r = lambda W_n, W_p, L: 2 * CL(W_n, W_p, L) / beta_p(W_p, L) / VDD * \
                            ((alpha_p - 0.1) / _alpha_p ** 2 + np.arctanh(1 - 0.1 / _alpha_p) / _alpha_p)
t_f = lambda W_n, W_p, L: 2 * CL(W_n, W_p, L) / beta_n(W_n, L) / VDD * \
                            ((alpha_n - 0.1) / _alpha_n ** 2 + np.arctanh(1 - 0.1 / _alpha_n) / _alpha_n)
# t_r = lambda W_n, W_p, L: CL(W_n, W_p, L) / K_p / VDD * \
#                             ((alpha_p - 0.1) / _alpha_p ** 2 + np.arctanh(1 - 0.1 / _alpha_p) / _alpha_p)
# t_f = lambda W_n, W_p, L: CL(W_n, W_p, L) / K_n / VDD * \
#                             ((alpha_n - 0.1) / _alpha_n ** 2 + np.arctanh(1 - 0.1 / _alpha_n) / _alpha_n)
T = lambda W_n, W_p, L, N: (t_r(W_n, W_p, L) + t_f(W_n, W_p, L)) * N / 2
# T = lambda W_n, W_p, L, N: N * VDD * CL(W_n, W_p, L) / (VDD - Vt) ** 2 * (1 / beta_n(W_n, L) + 1 / beta_p(W_p, L))
f = lambda N, W_p, W_n, L: 1 / T(W_n, W_p, L, N)
f = lambda data: 1 / T(data[2], data[1], data[3], data[0])

np.array(data.apply(f, axis=1)) / 1e6
f((N, W_p, W_n, L)) / 1e6
# f((11, 400e-9, 200e-9, 100e-9 / 11)) / 1e6
```

Out[8]:

```
array([29.06497859,  7.26624465,  1.81656116,  8.91483381,  2.22870845,
        0.55717711,  6.53957244,  1.63489311,  0.72661916,  0.12917768])
```

Out[8]:

```
18.165478994867176
```

公式2：

$$t_r = \frac{2V_{DD}C_L}{\beta_p(V_{DD} - |V_{tp}|)^2}$$

$$t_f = \frac{2V_{DD}C_L}{\beta_n(V_{DD} - |V_{tn}|)^2}$$

In [16]:

```python
InteractiveShell.ast_node_interactivity = 'all'
# InteractiveShell.ast_node_interactivity = 'last'

t_r = lambda W_n, W_p, L: 2 * VDD * CL(W_n, W_p, L) / beta_p(W_p, L) / (VDD - Vtp) ** 2
t_f = lambda W_n, W_p, L: 2 * VDD * CL(W_n, W_p, L) / beta_n(W_n, L) / (VDD - Vtn) ** 2
T = lambda W_n, W_p, L, N: (t_r(W_n, W_p, L) + t_f(W_n, W_p, L)) * N / 2
# T = lambda W_n, W_p, L, N: N * VDD * CL(W_n, W_p, L) / (VDD - Vt) ** 2 * (1 / beta_n(W_n, L) + 1 / beta_p(W_p, L))
f = lambda N, W_p, W_n, L: 1 / T(W_n, W_p, L, N)
f = lambda data: 1 / T(data[2], data[1], data[3], data[0])

np.array(data.apply(f, axis=1)) / 1e6
f((N, W_p, W_n, L)) / 1e6
```

Out[16]:

```
array([30.92214664,  7.73053666,  1.93263417,  9.52583685,  2.38145921,
        0.5953648 ,  6.97367707,  1.74341927,  0.77485301,  0.13743176])
```

Out[16]:

```
19.371325184125965
```

公式3：

$$t_r = \frac{2(V_{TN} - 0.1V_{DD})C_L}{\beta_p(V_{DD} - |V_{TP}|)^2} + \frac{(0.9V_{DD} - V_{TN})C_L}{k_p\left(\frac{W}{L}\right)_p\left[(V_{DD} - V_{TP})V_{DS} - \frac{1}{2}V_{DD}^2\right]}$$

$$t_f = \frac{2(V_{TP} - 0.1V_{DD})C_L}{\beta_n(V_{DD} - |V_{TN}|)^2} + \frac{(0.9V_{DD} - V_{TP})C_L}{k_n\left(\frac{W}{L}\right)_n\left[(V_{DD} - V_{TN})V_{DS} - \frac{1}{2}V_{DD}^2\right]}$$

In [17]:

```python
InteractiveShell.ast_node_interactivity = 'all'
# InteractiveShell.ast_node_interactivity = 'last'

t_r = lambda W_n, W_p, L: 2 * (Vtn - 0.1 * VDD) * CL(W_n, W_p, L) / beta_p(W_p, L) / (VDD - Vtp) ** 2 + \
                           (0.9 * VDD - Vtn) * CL(W_n, W_p, L) / (beta_p(W_p, L) * ((VDD - Vtp) * Vds - 0.5 * VDD ** 2))
t_f = lambda W_n, W_p, L: 2 * (Vtp - 0.1 * VDD) * CL(W_n, W_p, L) / beta_n(W_n, L) / (VDD - Vtn) ** 2 + \
                           (0.9 * VDD - Vtp) * CL(W_n, W_p, L) / (beta_n(W_n, L) * ((VDD - Vtn) * Vds - 0.5 * VDD ** 2))
T = lambda N, W_p, W_n, L: (t_r(W_n, W_p, L) + t_f(W_n, W_p, L)) * N / 2
# T = lambda W_n, W_p, L, N: N * VDD * CL(W_n, W_p, L) / (VDD - Vt) ** 2 * (1 / beta_n(W_n, L) + 1 / beta_p(W_p, L))
# f = lambda N, W_p, W_n, L: 1 / T(W_n, W_p, L, N)
```

```
f = lambda data: 1 / T(data[0], data[1], data[2], data[3])

np.array(data.apply(f, axis=1)) / 1e6
f((N, W_p, W_n, L)) / 1e6
```

Out[17]:
```
array([30.6367789 ,  7.65919472,  1.91479868,  9.59724599,  2.3993115 ,
        0.59982787,  6.97126651,  1.74281663,  0.77458517,  0.13616346])
```

Out[17]:
```
19.364629202901142
```

公式4：

$$t_r = t_{r1} + t_{r2} = \left[\frac{C_L(V_{TP} - 0.1V_{DD})}{K_P(V_{DD} - V_{TP})^2}\right] + \left[\frac{C_L}{2K_p(V_{DD} - V_{TP})} + ln\left(\frac{19V_{DD} - 20V_{TP}}{V_{DD}}\right)\right] = \frac{C_L}{K_P(V_{DD} - V_{TP})}\left[\frac{(V_{TP} - 0.1V_{DD})}{(V_{DD} - V_{TP})} + \frac{1}{2}ln\left(\frac{19V_{DD} - 20V}{V_{DD}}\right.\right.$$

$$t_f = t_{f1} + t_{f2} = \left[\frac{C_L(V_{TN} - 0.1V_{DD})}{K_N(V_{DD} - V_{TN})^2}\right] + \left[\frac{C_L}{2K_n(V_{DD} - V_{TN})} + ln\left(\frac{19V_{DD} - 20V_{TN}}{V_{DD}}\right)\right] = \frac{C_L}{K_N(V_{DD} - V_{TN})}\left[\frac{(V_{TN} - 0.1V_{DD})}{(V_{DD} - V_{TN})} + \frac{1}{2}ln\left(\frac{19V_{DD} - 2}{V_{DD}}\right.\right.$$

In [18]:
```
InteractiveShell.ast_node_interactivity = 'all'
# InteractiveShell.ast_node_interactivity = 'last'

t_r = lambda W_n, W_p, L: CL(W_n, W_p, L) / (beta_n(W_n, L) / 2) / (VDD - Vtp) * \
                          ((Vtp - 0.1 * VDD) / (VDD - Vtp) + 0.5 * np.log((19 * VDD - 20 * Vtp) / VDD))
t_f = lambda W_n, W_p, L: 2 * CL(W_n, W_p, L) / (beta_p(W_p, L) / 2) / (VDD - Vtn) * \
                          ((Vtn - 0.1 * VDD) / (VDD - Vtn) + 0.5 * np.log((19 * VDD - 20 * Vtn) / VDD))
T = lambda W_n, W_p, L, N: (t_r(W_n, W_p, L) + t_f(W_n, W_p, L)) * N / 2
# T = lambda W_n, W_p, L, N: N * VDD * CL(W_n, W_p, L) / (VDD - Vt) ** 2 * (1 / beta_n(W_n, L) + 1 / beta_p(W_p, L))
# f = lambda N, W_p, W_n, L: 1 / T(W_n, W_p, L, N)
f = lambda data: 1 / T(data[2], data[1], data[3], data[0])

np.array(data.apply(f, axis=1)) / 1e6
f((N, W_p, W_n, L)) / 1e6
```

Out[18]:
```
array([20.01206632,  5.00301658,  1.25075414,  4.91977923,  1.22994481,
        0.3074862 ,  3.9747328 ,  0.9936832 ,  0.44163698,  0.08894252])
```

Out[18]:
```
11.04092444742957
```

公式5：

$$t_r \approx 2\frac{C_L}{K_nV_{DD}} = 2\frac{C_L}{\frac{\beta_n}{2}V_{DD}}$$

$$t_f \approx 2\frac{C_L}{K_p V_{DD}} = 2\frac{C_L}{\frac{\beta_p}{2} V_{DD}}$$

In [19]:

```
InteractiveShell.ast_node_interactivity = 'all'
# InteractiveShell.ast_node_interactivity = 'last'

# t_r = lambda W_n, W_p, L: 2 * CL(W_n, W_p, L) / (K_n * VDD)
# t_f = lambda W_n, W_p, L: 2 * CL(W_n, W_p, L) / (K_p * VDD)
t_r = lambda W_n, W_p, L: 2 * CL(W_n, W_p, L) / (beta_n(W_n, L) / 2 * VDD)
t_f = lambda W_n, W_p, L: 2 * CL(W_n, W_p, L) / (beta_p(W_p, L) / 2 * VDD)
T = lambda N, W_p, W_n, L: (t_r(W_n, W_p, L) + t_f(W_n, W_p, L)) * N / 2
# T = lambda W_n, W_p, L, N: N * VDD * CL(W_n, W_p, L) / (VDD - Vt) ** 2 * (1 / beta_n(W_n, L) + 1 / beta_p(W_p, L))
# f = lambda N, W_p, W_n, L: 1 / T(W_n, W_p, L, N)
f = lambda data: 1 / T(data[0], data[1], data[2], data[3])

np.array(data.apply(f, axis=1)) / 1e6
f((N, W_p, W_n, L)) / 1e6
```

Out[19]:

```
array([35.70659453,  8.92664863,  2.23166216, 10.80670826,  2.70167707,
        0.67541927,  7.97642283,  1.99410571,  0.8862692 ,  0.15869598])
```

Out[19]:

```
22.156730074127008
```

t_{pHL} 和 t_{pLH}

公式6：

$$t_{pHL} = \frac{3ln2}{4}\frac{V_{DD}C_L}{I_{DSATn}} = \frac{3ln2}{4}\frac{V_{DD}\ C_L}{\left(\frac{W}{L}\right)_n k_n'\ V_{DSATn}\ \left(V_{DD} - V_{Tn} - \frac{V_{DSATn}}{2}\right)}$$

$$t_{pLH} = \frac{3ln2}{4}\frac{V_{DD}C_L}{I_{DSATp}} = \frac{3ln2}{4}\frac{V_{DD}\ C_L}{\left(\frac{W}{L}\right)_p k_p'\ V_{DSATp}\ \left(V_{DD} - V_{Tp} - \frac{V_{DSATp}}{2}\right)}$$

In [20]:

```
InteractiveShell.ast_node_interactivity = 'all'
# InteractiveShell.ast_node_interactivity = 'last'

# tpHL = lambda W_n, W_p, L: (3 * np.log(2) / 4) * (VDD * CL(W_n, W_p, L) / (beta_n(W_n, L) * Vdsat * (VDD - Vtn - Vdsat / 2)))
# tpLH = lambda W_n, W_p, L: (3 * np.log(2) / 4) * (VDD * CL(W_n, W_p, L) / (beta_p(W_p, L) * Vdsat * (VDD - Vtp - Vdsat / 2)))
tpHL = lambda W_n, W_p, L: (3 * np.log(2) / 4) * (VDD * CL(W_n, W_p, L) / (W_n / L * K_n * Vdsatn * (VDD - Vtn - Vdsatn / 2)))
tpLH = lambda W_n, W_p, L: (3 * np.log(2) / 4) * (VDD * CL(W_n, W_p, L) / (W_p / L * K_p * Vdsatp * (VDD - Vtp - Vdsatp / 2)))
```

```python
tpd = lambda W_p, W_n, L: (tpHL(W_n, W_p, L) + tpLH(W_n, W_p, L)) / 2
f = lambda data:1 / (2 * data[0]  * tpd(data[1], data[2], data[3]))

np.array(data.apply(f, axis=1)) / 1e6
f((N, W_p, W_n, L)) / 1e6
```

Out[20]:

```
array([29.74081841,  7.4352046 ,  1.85880115,  9.16191839,  2.2904796 ,
        0.5726199 ,  6.70725955,  1.67681489,  0.74525106,  0.13218142])
```

Out[20]:

```
18.631276519057344
```

In [ ]: