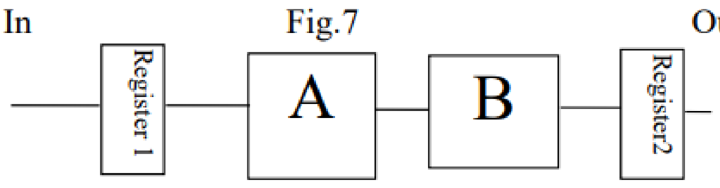
Consider the circuit in Fig.7. Modules A and B have a delay of 19 ns and 30.5 ns at 0.9V (Vdd) with switching of 20 pF and 30 pF respectively. All the buses in Fig.7 are 16 bits. One register has a 0.5 ns clock-to-Q delay and switches 0.05 pF. The clock rate of Fig.7 is thus 1/(50 ns) and the power dissipation is P0. The power dissipation can be estimated by P= C•Vdd^2•F and the delay with respect to Vdd can be approximated by k/(Vdd-Vt) with Vt equals 0.3 V. k is a constant and is different for A and B. You can use the information of delay time, Vdd and Vt to calculate k.

Fig.7



第一步先計算整理所需要的參數，並且根據題目所給的擬合曲線計算 AB 模組個別的 K 值(正比常數)。

$$\begin{cases} T_A = 19ns @ Vdd = 0.9V \\ T_B = 30.5ns @ Vdd = 0.9V \\ C_A = 20pF \\ C_B = 30pF \\ T_{c2q} = T_{reg} = 0.5ns \\ C_{reg} = 0.05pF \\ T_{clk} = 50ns,\ f = \dfrac{1}{50ns} = 20MHz \\ V_t = 0.3V \end{cases}$$

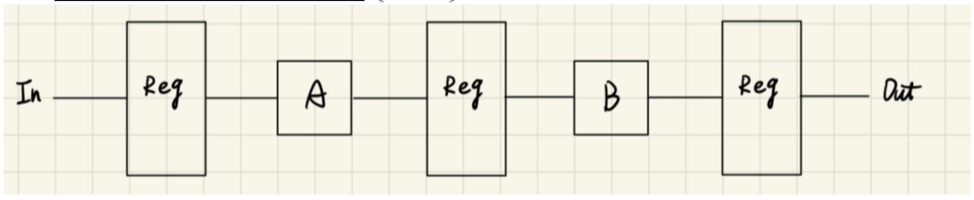
$$\begin{cases} T_A = \dfrac{K_A}{(Vdd - V_t)} \Rightarrow K_A = T_A \cdot (Vdd - V_t) = 19n \cdot (0.9 - 0.3) = 11.4n(s \cdot V) \\ T_B = \dfrac{K_B}{(Vdd - V_t)} \Rightarrow K_B = T_B \cdot (Vdd - V_t) = 30.5n \cdot (0.9 - 0.3) = 18.3n(s \cdot V) \end{cases}$$

接著，計算在目前的這個狀況下還沒進行 low power 優化時 design 所耗的 power:

$$P_0 = P_A\big|_{Vdd=0.9V} + P_B\big|_{Vdd=0.9V} + P_{reg}\big|_{Vdd=0.9V} = \left(C_A + C_B + 2 \cdot 16 \cdot C_{reg}\right) \cdot Vdd^2 \cdot f = 835.92\mu W$$

而後，來試著將 design 切 pipeline 使得在單個 clk period 所需要計算的數量減少，使得 Vdd 可以下降以降低 low power。抑或是使用 parallel design 並且使用 MUX 來決定要輸出哪個結果也可以在滿足 timing spec 的情況下，去進一步的降低 power 的損耗。

(a) Adding a pipeline register between A and B allows for reduction of the supply voltage (A and B can use different Vdd) while maintaining throughput with power dissipation of P1. Show the block diagram. Explain the operation and calculate the power reduction ratio, P1/P0 (30%)



Explain the operation:
在 A 與 B 模塊中間多切一級 pipeline 可以使得原本 AB 皆需要在同個 clock cycle 內做完變為 A 與 B 可以各自擁有一個 clock cycle time 來完成。使得 timing slack 變大。因此在同一個 clock 內，因為需要計算的事件減少了，我們可以將 AB 模塊的電壓降低，使得整體 design 的 power 可以更進一步的減少。但值得注意的是，如同教授上課所提，在單一晶片中使用多個 Vdd 不見得會是一個好的 low power design。因為在這個 chip 外面會需要有各式各樣的 power management system 或是 power deliver 的 framework。若是要真的確實的分析這樣的改動是否真的有將 design 變得 low power，是不能夠忽略這部分外界供電給這個 chip 的 cost。但這只是一個簡單的小練習，我們就先忽略這個部分，專注在單一 chip 上的 low power design。

Low Vdd with delay calculation:
由上面的分析可以得知，A 與 B 可以使用較低的電壓來供電，在這邊我將會計算若是在相同的 clock period 要完成 A 運算，及在相同的 clock period 要完成 B 的計算，A 與 B 的電壓可以降到多少。

而在這之中需要滿足的不等式為:

$$T_A', T_B' \le T_{clk} - T_{c2q} = 50ns - 0.5ns = 49.5ns$$

因此可以推得:

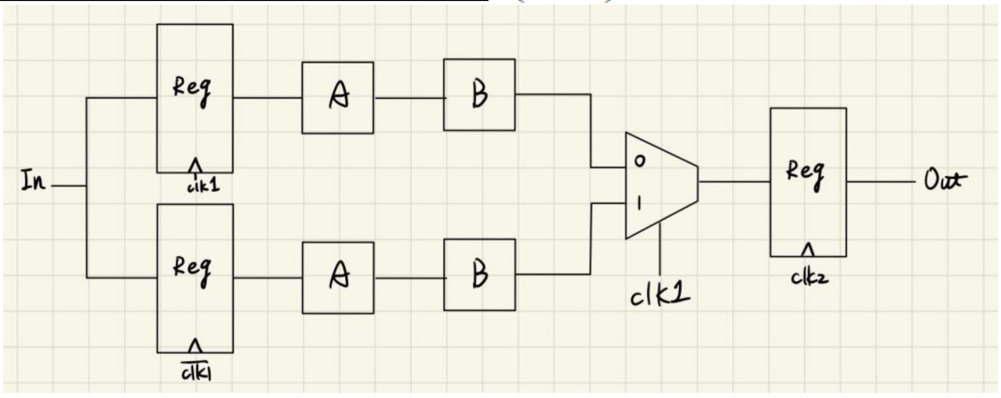
$$\begin{cases} T_A' = \dfrac{K_A}{\left(Vdd_A' - V_t\right)} \le 49.5ns \Rightarrow Vdd_A' \ge 0.530V \\ T_B' = \dfrac{K_B}{\left(Vdd_B' - V_t\right)} \le 49.5ns \Rightarrow Vdd_B' \ge 0.669V \end{cases}$$

$$P_1 = P_A\big|_{Vdd=0.53V} + P_B\big|_{Vdd=0.669V} + 3P_{reg}\big|_{Vdd=0.9V} \approx 419.77\mu W$$

因此節省了:

$$\frac{P_1}{P_0} = \frac{419.77}{835.92} \approx 50.22\%$$

(b) Assume that a 2-to-1 multiplexer has a delay of 0.4 ns at 0.9 V and switches 0.05 pF. Try parallel version with two copies (using two A and B modules) while maintaining date rate (P2). Show the block diagram, explain the operation and calculate power reduction ratio, P2/P0. (30%)



$$\begin{cases} clk1 = 10MHz \\ clk2 = 20MHz \end{cases}$$

Explain the operation:
透過兩套 AB 的模組架構，我們可以在相同的 clk rate 中達到兩倍的 data rate。但是這邊只需要在維持原本設計的 data rate 中來做 low power 的改良，因此可以將上下兩組硬體的 clk rate 放鬆為一半的速度，使得 AB 模組運算的時間可以較長，使得 timing slack 較大，讓 AB 模組的供給電壓下降以達到 low power 的設計。

因此定義 clk1 為 10MHz 的操作頻率。在上圖，上面那組的 AB 模塊僅有當 clk1 posedge 時會 fetch 值進來運算，算到 clk1 下次拉起來前要結束即可。而下面那組 AB 模塊為了要與上面那組模塊計算不同的值，我們可以將其改為 negative trigger，或是吃 clk1_bar 的 positive trigger 的 register。在最後要輸出前，因為有兩份結果，將會透過 MUX 來決定最終要輸出的結果來自哪個硬體，最後送進輸出 data rate 為 20MHz 的 register 來輸出。

Low Vdd with delay calculation:
而在這之中需要滿足的不等式為:

$$T_A'' + T_B'' \le T_{clk1} - T_{c2q} - T_{MUX} = 2 \cdot 50ns - 0.5ns - 0.4ns = 99.1ns$$

因此可以推得可以將其 model 為一個最佳化的算式:

$$\begin{cases} min: P = P_A\big|_{Vdd=Vdd_A''} + P_B\big|_{Vdd=Vdd_B''} + P_{reg}\big|_{Vdd=0.9V} + P_{mux}\big|_{Vdd=0.9V} \\ subject\ to: \dfrac{K_A}{\left(Vdd_A'' - V_t\right)} + \dfrac{K_B}{\left(Vdd_B'' - V_t\right)} \le T_{clk1} - T_{c2q} - T_{MUX} \end{cases}$$

其中 Preg,Pmux 為定值，因此可以改寫為:

$$\begin{cases} min: P' = \left(C_A Vdd_A''^2 + C_B Vdd_B''\right)\cdot f_{clk1} \\ subject\ to: \dfrac{K_A}{\left(Vdd_A'' - V_t\right)} + \dfrac{K_B}{\left(Vdd_B'' - V_t\right)} \le T_{clk1} - T_{c2q} - T_{MUX} \end{cases}$$

帶入相關數據可以得到:

$$\begin{cases} min: P' = \left(20p\cdot Vdd_A''^2 + 30p\cdot Vdd_B''\right)\cdot 10M \\ subject\ to: \dfrac{11.4n\left(s\cdot V\right)}{\left(Vdd_A'' - 0.3\right)} + \dfrac{18.3n\left(s\cdot V\right)}{\left(Vdd_B'' - 0.3\right)} \le 99.1n \end{cases}$$

可以使用 Lagrange Multiplier 來解:

$$L\left(Vdd_A'', Vdd_B'', \lambda\right) = \left(C_A Vdd_A''^2 + C_B Vdd_B''\right)\cdot f_{clk1} + \lambda\left(\frac{K_A}{\left(Vdd_A'' - V_t\right)} + \frac{K_B}{\left(Vdd_B'' - V_t\right)} - \left(T_{clk1} - T_{c2q} - T_{MUX}\right)\right)$$

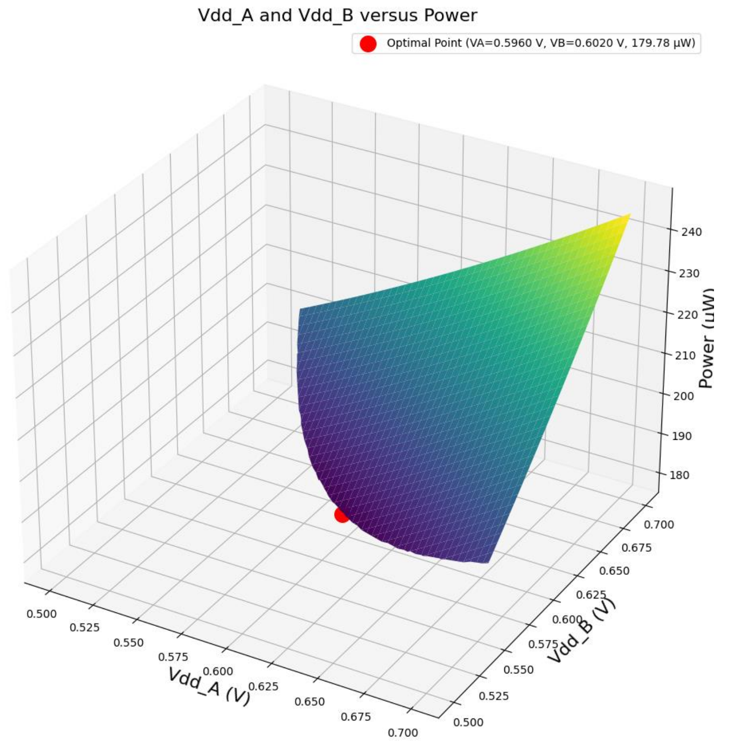
而後解偏微分的聯立方程便可以找到極值:

$$\begin{cases} \dfrac{\partial L}{\partial Vdd_A''} = 2C_A Vdd_A'' f_{clk1} - \lambda \dfrac{K_A}{\left(Vdd_A'' - V_t\right)^2} = 0 \\ \dfrac{\partial L}{\partial Vdd_B''} = 2C_B Vdd_B'' f_{clk1} - \lambda \dfrac{K_A}{\left(Vdd_B'' - V_t\right)^2} = 0 \\ \dfrac{\partial L}{\partial \lambda} = \dfrac{K_A}{\left(Vdd_A'' - V_t\right)} + \dfrac{K_B}{\left(Vdd_B'' - V_t\right)} - \left(T_{clk1} - T_{c2q} - T_{MUX}\right) = 0 \end{cases}$$

其實理論上三個變數三條方程式，是可以使用 analytical 的方法找到最佳解，但是我資質駑鈍，數學解不出來，因此我將借助程式之力，幫我將功耗最小值時的 AB 模塊供電電壓算出來。先透過上面聯立中第三個式子算出當 Vdd_A'' = Vdd_B'' 時 $Vdd = 0.6V$ ，而後將 $Vdd_A'' = Vdd_B'' = Vdd = 0.6V$ 視為初始答案，給 python 現成的 scipy 套件來解出極值為多少。而我也注意到初始解的位置會很大機會的影響最終最佳化的結果，因此我在 $0.6V \pm 0.1V$ 附近先找 10 個 cost 最小的點當成初始的解，每個解再經由 scipy 中 optimization 的套件來解，以逼近真正的最小值。最後，為了更加的了解 Vdd_A'' 及 Vdd_B'' 對 power 的影響，我也將其視覺化為 3d 的圖片，使其一目瞭然。以下是結果:

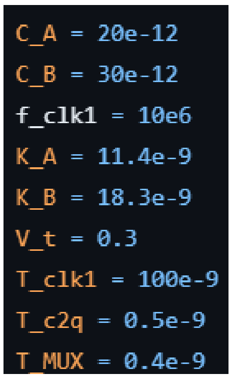
$$min\left(P'\right) = 179.78\mu W\ @Vdd_A'' = 0.5960V, Vdd_B'' 0.602V$$

AB 模塊供電電壓與功耗對應圖:



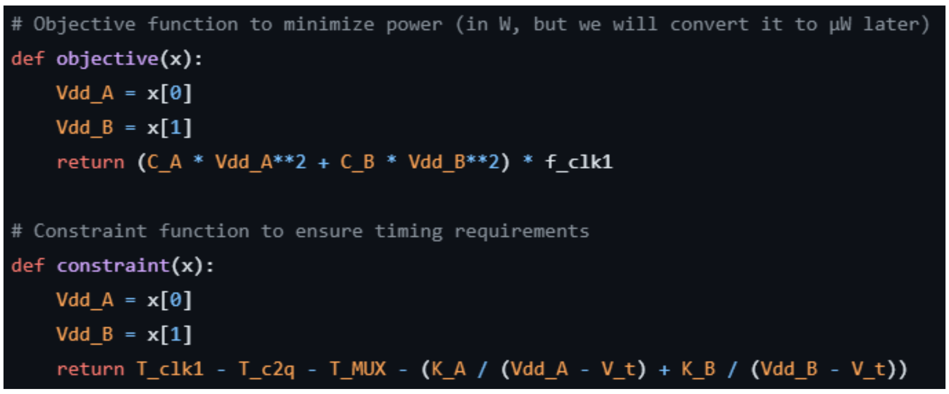
程式運作分析:

● 常數定義



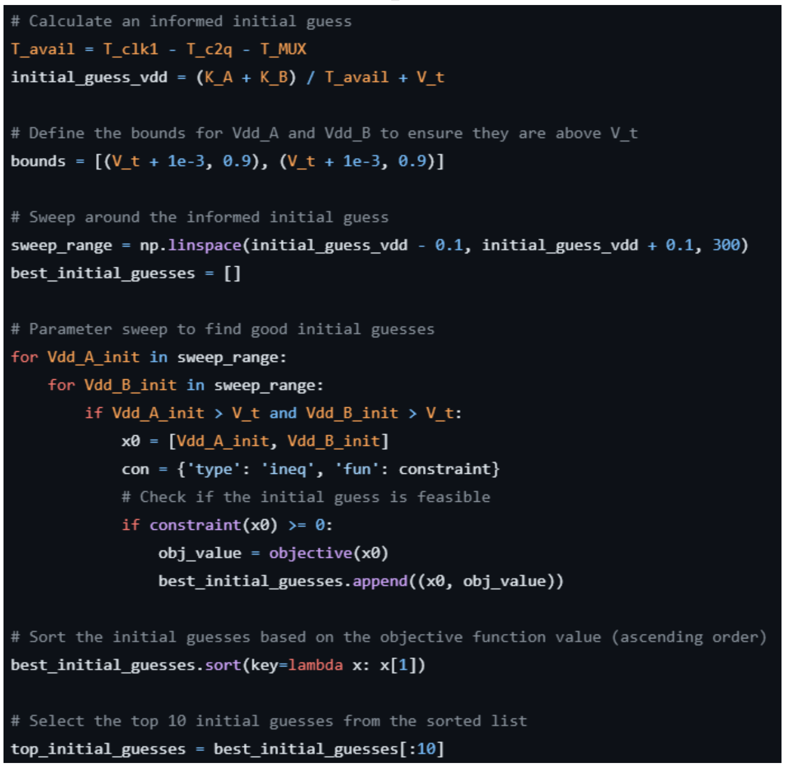
```
C_A = 20e-12
C_B = 30e-12
f_clk1 = 10e6
K_A = 11.4e-9
K_B = 18.3e-9
V_t = 0.3
T_clk1 = 100e-9
T_c2q = 0.5e-9
T_MUX = 0.4e-9
```

● 目標函數及限制函數的定義



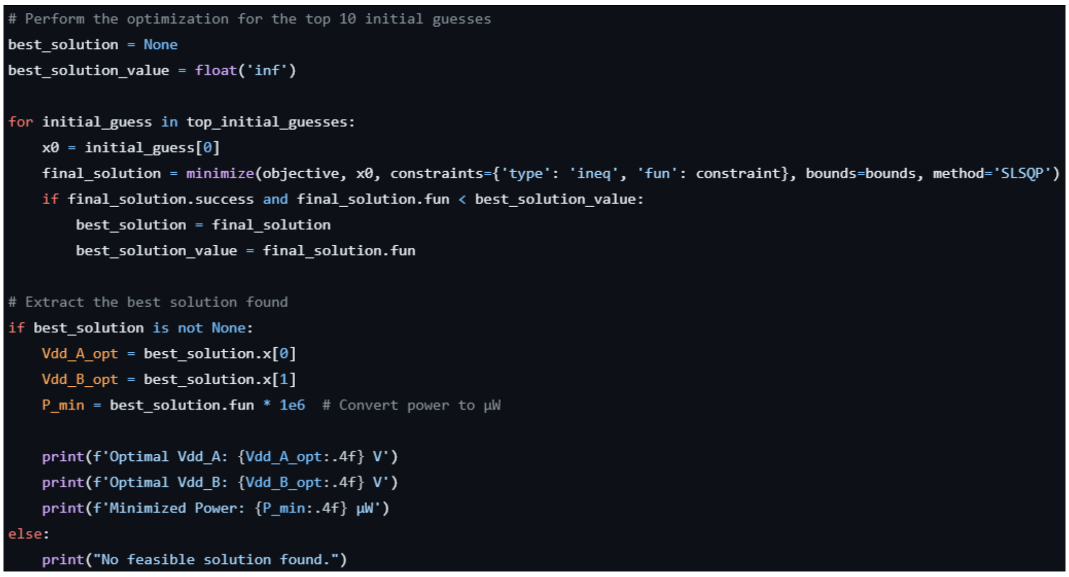
```
# Objective function to minimize power (in W, but we will convert it to μW later)
def objective(x):
    Vdd_A = x[0]
    Vdd_B = x[1]
    return (C_A * Vdd_A**2 + C_B * Vdd_B**2) * f_clk1

# Constraint function to ensure timing requirements
def constraint(x):
    Vdd_A = x[0]
    Vdd_B = x[1]
    return T_clk1 - T_c2q - T_MUX - (K_A / (Vdd_A - V_t) + K_B / (Vdd_B - V_t))
```

- 找到前 10 好的初始解，並且定義 optimization 邊界



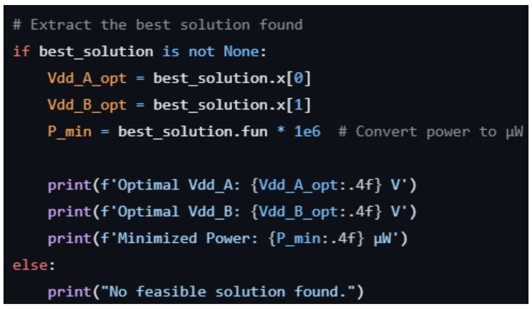
```python
# Calculate an informed initial guess
T_avail = T_clk1 - T_c2q - T_MUX
initial_guess_vdd = (K_A + K_B) / T_avail + V_t

# Define the bounds for Vdd_A and Vdd_B to ensure they are above V_t
bounds = [(V_t + 1e-3, 0.9), (V_t + 1e-3, 0.9)]

# Sweep around the informed initial guess
sweep_range = np.linspace(initial_guess_vdd - 0.1, initial_guess_vdd + 0.1, 300)
best_initial_guesses = []

# Parameter sweep to find good initial guesses
for Vdd_A_init in sweep_range:
    for Vdd_B_init in sweep_range:
        if Vdd_A_init > V_t and Vdd_B_init > V_t:
            x0 = [Vdd_A_init, Vdd_B_init]
            con = {'type': 'ineq', 'fun': constraint}
            # Check if the initial guess is feasible
            if constraint(x0) >= 0:
                obj_value = objective(x0)
                best_initial_guesses.append((x0, obj_value))

# Sort the initial guesses based on the objective function value (ascending order)
best_initial_guesses.sort(key=lambda x: x[1])

# Select the top 10 initial guesses from the sorted list
top_initial_guesses = best_initial_guesses[:10]
```

- 針對這 10 個點更進一步去優化，尋找更小的功耗可能性



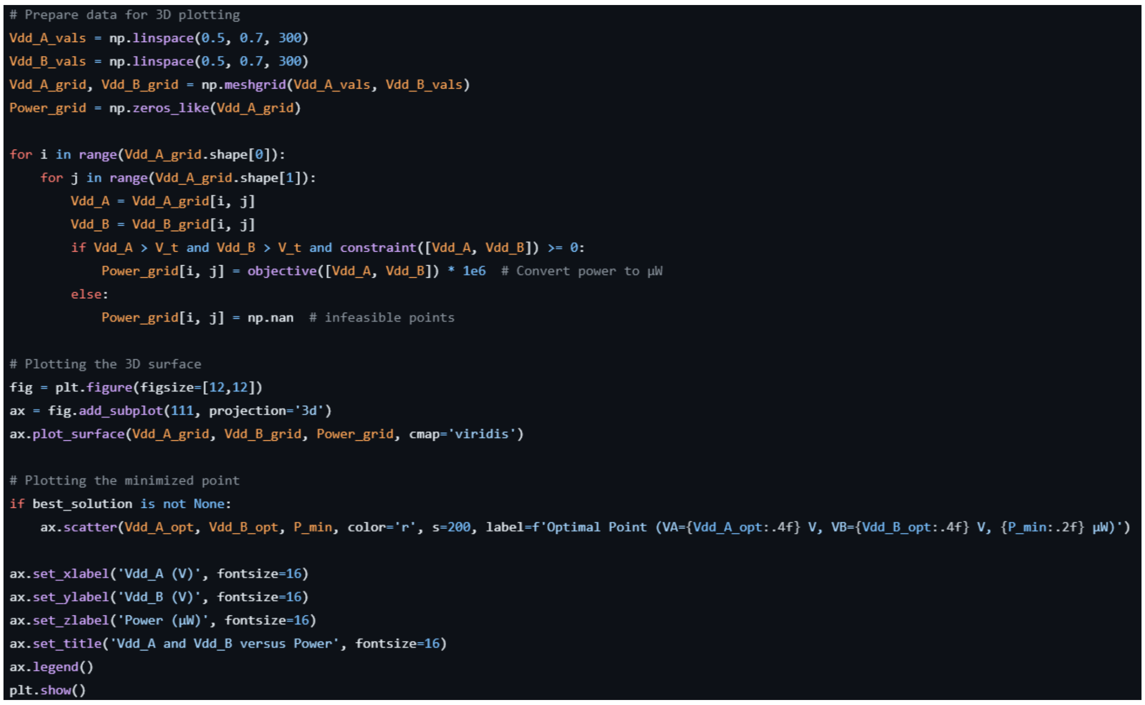
```python
# Perform the optimization for the top 10 initial guesses
best_solution = None
best_solution_value = float('inf')

for initial_guess in top_initial_guesses:
    x0 = initial_guess[0]
    final_solution = minimize(objective, x0, constraints={'type': 'ineq', 'fun': constraint}, bounds=bounds, method='SLSQP')
    if final_solution.success and final_solution.fun < best_solution_value:
        best_solution = final_solution
        best_solution_value = final_solution.fun

# Extract the best solution found
if best_solution is not None:
    Vdd_A_opt = best_solution.x[0]
    Vdd_B_opt = best_solution.x[1]
    P_min = best_solution.fun * 1e6  # Convert power to µW

    print(f'Optimal Vdd_A: {Vdd_A_opt:.4f} V')
    print(f'Optimal Vdd_B: {Vdd_B_opt:.4f} V')
    print(f'Minimized Power: {P_min:.4f} µW')
else:
    print("No feasible solution found.")
```

- 印出最佳值



```python
# Extract the best solution found
if best_solution is not None:
    Vdd_A_opt = best_solution.x[0]
    Vdd_B_opt = best_solution.x[1]
    P_min = best_solution.fun * 1e6  # Convert power to µW

    print(f'Optimal Vdd_A: {Vdd_A_opt:.4f} V')
    print(f'Optimal Vdd_B: {Vdd_B_opt:.4f} V')
    print(f'Minimized Power: {P_min:.4f} µW')
else:
    print("No feasible solution found.")
```

- 資料視覺化



```python
# Prepare data for 3D plotting
Vdd_A_vals = np.linspace(0.5, 0.7, 300)
Vdd_B_vals = np.linspace(0.5, 0.7, 300)
Vdd_A_grid, Vdd_B_grid = np.meshgrid(Vdd_A_vals, Vdd_B_vals)
Power_grid = np.zeros_like(Vdd_A_grid)

for i in range(Vdd_A_grid.shape[0]):
    for j in range(Vdd_A_grid.shape[1]):
        Vdd_A = Vdd_A_grid[i, j]
        Vdd_B = Vdd_B_grid[i, j]
        if Vdd_A > V_t and Vdd_B > V_t and constraint([Vdd_A, Vdd_B]) >= 0:
            Power_grid[i, j] = objective([Vdd_A, Vdd_B]) * 1e6  # Convert power to µW
        else:
            Power_grid[i, j] = np.nan  # infeasible points

# Plotting the 3D surface
fig = plt.figure(figsize=[12,12])
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(Vdd_A_grid, Vdd_B_grid, Power_grid, cmap='viridis')

# Plotting the minimized point
if best_solution is not None:
    ax.scatter(Vdd_A_opt, Vdd_B_opt, P_min, color='r', s=200, label=f'Optimal Point (VA={Vdd_A_opt:.4f} V, VB={Vdd_B_opt:.4f} V, {P_min:.2f} µW)')

ax.set_xlabel('Vdd_A (V)', fontsize=16)
ax.set_ylabel('Vdd_B (V)', fontsize=16)
ax.set_zlabel('Power (µW)', fontsize=16)
ax.set_title('Vdd_A and Vdd_B versus Power', fontsize=16)
ax.legend()
plt.show()
```

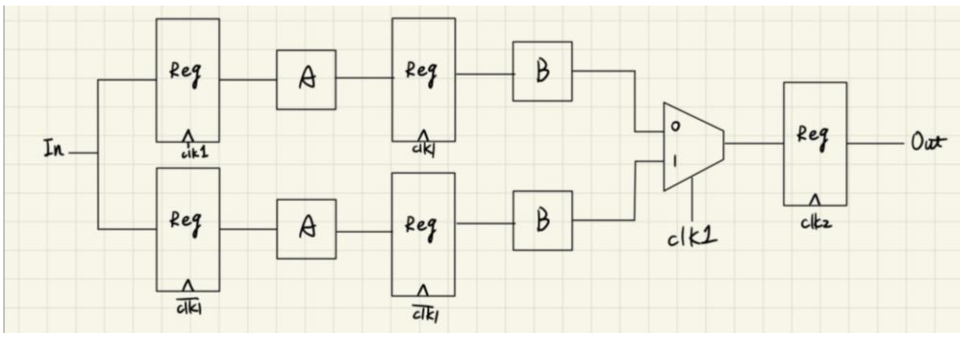
最後加上 Mux 及 Reg 的功耗，便為此 parallel design 的整體功耗:

$$P_2 = 2P_A\big|_{Vdd=0.5960V} + 2P_B\big|_{Vdd=0.602V} + 2P_{reg}\big|_{Vdd=0.9V,clk1} + P_{reg}\big|_{Vdd=0.9V,clk2} + P_{mux}\big|_{Vdd=0.9V} \approx 359.56 + 25.92 + 12.96 = 398.44\mu W$$

因此節省了:

$$\frac{P_2}{P_0} = \frac{398.44}{835.92} \approx 47.66\%$$

(c) Try to combine (a) and (b) to design a parallel-pipeline version while maintaining data rate (p3). <u>Show the block diagram, explain the operation and calculate power reduction ratio, P3/P0 (40%)</u>



Explain the operation:
基本上功能如同前一題 parallel 相同，只不過在 AB 之間又多加了一級 pipeline register，使得 AB 模塊的供電電壓可以更加下降，希望可以再次減小 power 損耗。

Low Vdd with delay calculation:

$$\begin{cases} T_A''' = \dfrac{K_A}{\left(Vdd_A''' - V_t\right)} \le T_{clk1} - T_{c2q} \Rightarrow Vdd_A''' \ge V_t + \dfrac{K_A}{T_{clk1} - T_{c2q}} \ge 0.4146V \\ T_B''' = \dfrac{K_B}{\left(Vdd_B''' - V_t\right)} \le T_{clk1} - T_{c2q} - T_{MUX} \Rightarrow Vdd_B''' \ge V_t + \dfrac{K_B}{T_{clk1} - T_{c2q} - T_{MUX}} \ge 0.4847V \end{cases}$$

而後計算總功耗:

$$P_3 = 2P_A\big|_{Vdd=0.4146V} + 2P_B\big|_{Vdd=0.4847V} + 4P_{reg}\big|_{Vdd=0.9V,clk1} + P_{reg}\big|_{Vdd=0.9V,clk2} + P_{mux}\big|_{Vdd=0.9V}$$

$$\begin{cases} P_A\big|_{Vdd=0.4146V} = 20p\cdot 0.4146^2\cdot 10M = 34.3786\mu W \\ P_B\big|_{Vdd=0.4847V} = 30p\cdot 0.4847^2\cdot 10M = 70.4802\mu W \\ P_{reg}\big|_{Vdd=0.9V,clk1} = \left(16\cdot 0.05p\right)\cdot 0.9^2\cdot 10M = 6.48\mu W \\ P_{reg}\big|_{Vdd=0.9V,clk2} = \left(16\cdot 0.05p\right)\cdot 0.9^2\cdot 20M = 12.96\mu W \\ P_{mux}\big|_{Vdd=0.9V} = \left(16\cdot 0.05p\right)\cdot 0.9^2\cdot 20M = 12.96\mu W \end{cases} \Rightarrow P_3 \approx 261.56\mu W$$

因此節省了:

$$\frac{P_3}{P_0} = \frac{261.56}{835.92} \approx 31.29\%$$

總而言之，pipeline 又 parallel 可以消耗最少的功耗。但是換來的是面積的增長。此外，在一顆 chip 上供給不同的電壓所需要的 power management 及 power deliver framework 所帶來的 cost 也沒有考慮進去。這次的作業其實蠻像是可以透過 synthesis tool 來設定優化方向，並且告知 PPA 之間的取捨，使其設計出一個會 meet timing slack 的電路，並且功耗及面積是最佳化的。整體來說，還蠻有趣的一次 lab。