ECE 216B

Problem 1. Delay Model

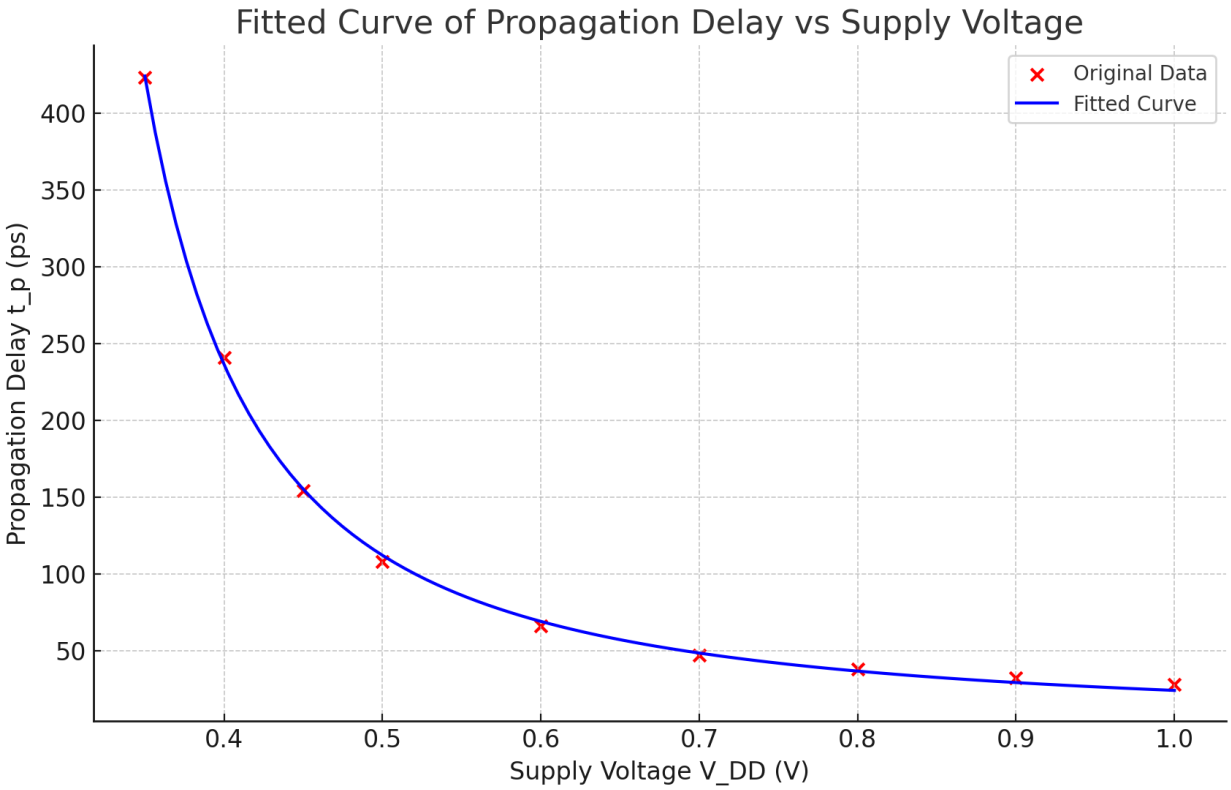
A

```
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

# Given data from the problem
V_DD = np.array([1.00, 0.90, 0.80, 0.70, 0.60, 0.50, 0.45, 0.
40, 0.35]) # Supply Voltage in Volts
t_p_typical = np.array([28, 32, 38, 47, 66, 108, 154, 241, 42
3]) # Propagation delay in picoseconds for typical process

# Function to model t_p as a function of V_DD
def model_t_p(V_DD, K_d, V_on, alpha_d):
    return (K_d * V_DD) / ((V_DD - V_on)**alpha_d)

# Initial guesses for the parameters
initial_guess = [1, 0.2, 1]

# Perform the curve fit
popt, pcov = curve_fit(model_t_p, V_DD, t_p_typical, p0=initi
al_guess)

# Extracting the optimal values of K_d, V_on, and alpha_d
K_d_optimal, V_on_optimal, alpha_d_optimal = popt

# Using the obtained parameters to plot the fitted curve
```

```
V_DD_fit = np.linspace(min(V_DD), max(V_DD), 100) # Generatin
g V_DD values for plotting
t_p_fit = model_t_p(V_DD_fit, *popt) # Calculated t_p values
using the fitted model

# Plotting the original data and the fitted curve
plt.figure(figsize=(10, 6))
plt.scatter(V_DD, t_p_typical, label='Original Data', color
='red')
plt.plot(V_DD_fit, t_p_fit, label='Fitted Curve', color='blu
e')
plt.title('Fitted Curve of Propagation Delay vs Supply Voltag
e')
plt.xlabel('Supply Voltage V_DD (V)')
plt.ylabel('Propagation Delay t_p (ps)')
plt.legend()
plt.grid(True)
plt.show()

# Show the optimal parameters
K_d_optimal, V_on_optimal, alpha_d_optimal
```

Result

```
(13.918365303752909, 0.22301753019097617, 2.1643271130258395)
```



I employed a nonlinear fitting method to a mathematical model of propagation delay, which is expressed as follows:

$t_p = \frac{K_d \cdot V_{DD}}{(V_{DD} - V_{on})^{\alpha_d}}$

Where:

- t_p represents the propagation delay of a FO4 gate,
- V_{DD} is the supply voltage,
- K_d, V_{on}, and α_d are the parameters to be determined.

The nonlinear curve fitting yielded the following optimal values for the parameters:

- The scaling factor K_d is approximately 13.92,
- The effective turn-on voltage V_{on} is approximately $0.223V$,
- The exponent α_d is approximately 2.16.

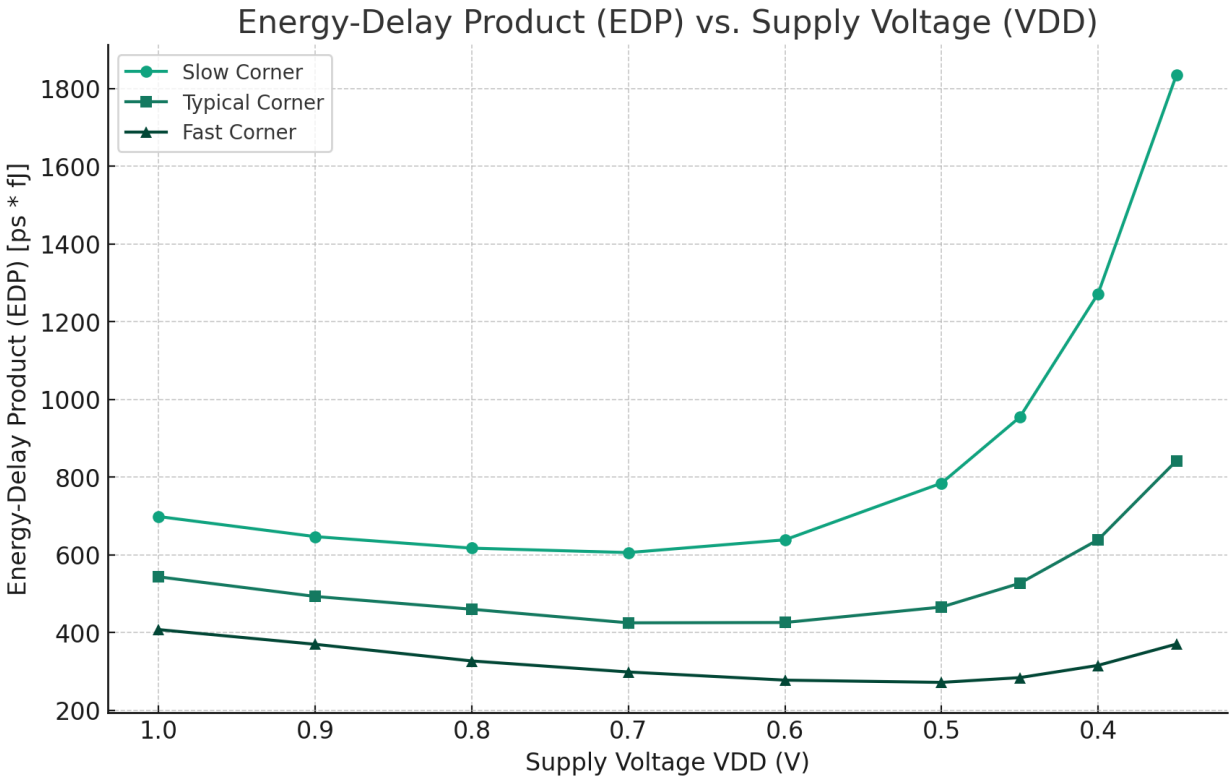
B

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Define the provided data
data = {
    'VDD': [1.00, 0.90, 0.80, 0.70, 0.60, 0.50, 0.45, 0.40,
0.35],
    'tp_slow': [36, 42, 51, 67, 99, 182, 279, 480, 922],
    'tp_typical': [28, 32, 38, 47, 66, 108, 154, 241, 423],
    'tp_fast': [21, 24, 27, 33, 43, 63, 83, 119, 186],
    'Energy': [19.4, 15.4, 12.1, 9.04, 6.45, 4.31, 3.42, 2.6
5, 1.99]
}

# Convert to a DataFrame
df = pd.DataFrame(data)

# Calculate the Energy-Delay Product (EDP) for each corner
df['EDP_slow'] = df['tp_slow'] * df['Energy']
df['EDP_typical'] = df['tp_typical'] * df['Energy']
df['EDP_fast'] = df['tp_fast'] * df['Energy']

# Find the VDD value that minimizes EDP for each corner
min_EDP_slow_VDD = df.loc[df['EDP_slow'].idxmin(), 'VDD']
min_EDP_typical_VDD = df.loc[df['EDP_typical'].idxmin(), 'VD
D']
min_EDP_fast_VDD = df.loc[df['EDP_fast'].idxmin(), 'VDD']

# Plot the Energy-Delay Product (EDP) for all process corners
plt.figure(figsize=(10, 6))
```

```
plt.plot(df['VDD'], df['EDP_slow'], marker='o', label='Slow C
orner')
plt.plot(df['VDD'], df['EDP_typical'], marker='s', label='Typ
ical Corner')
plt.plot(df['VDD'], df['EDP_fast'], marker='^', label='Fast C
orner')
plt.xlabel('Supply Voltage VDD (V)')
plt.ylabel('Energy-Delay Product (EDP) [ps * fJ]')
plt.title('Energy-Delay Product (EDP) vs. Supply Voltage (VD
D)')
plt.legend()
plt.grid(True)
plt.gca().invert_xaxis()  # Inverting the x-axis to show the
decrease in VDD
plt.show()

(min_EDP_slow_VDD, min_EDP_typical_VDD, min_EDP_fast_VDD)
```

Result:

```
(0.7, 0.7, 0.5)
```



C

1. Determine the propagation delays at $V_{DD} = 0.6V$ and $V_{DD} = 1V$ from the provided data table. Call these $t_{p0.6}$ and $t_{p1.0}$, respectively.
2. Calculate the delay ratio $\frac{t_{p1.0}}{t_{p0.6}}$.
3. Apply this ratio to the target clock frequency to adjust for the change in propagation delay due to the change in supply voltage.

   $f_{1.0} = f_{0.6} \times \frac{t_{p0.6}}{t_{p1.0}}$
   where
   $f_{0.6}$ is the target frequency at $V_{DD} = 0.6V$, and $f_{1.0}$is the equivalent frequency at $V_{DD} = 1V$.

$V_{DD} = 0.6V$ is 66 ps (from the table) and at $V_{DD} = 1V$ is 28 ps. Using these values:

$f_{1.0} = 250MHz \times \frac{66ps}{28ps}$

$= 589.285714MHz$

Problem 2.

a) The key findings of the paper demonstrate that Amber, a reconfigurable System on Chip (SoC), substantially enhances the efficiency and performance of imaging, vision, and machine learning (ML) applications. This is achieved through three major technological innovations: fast dynamic partial reconfiguration (DPR), on-chip streaming memories specifically designed for affine access patterns, and reduced overhead for complex arithmetic operations. These advancements allow Amber to support multiple computational kernels simultaneously with rapid reconfiguration times. Additionally, it reduces both the physical area and power consumption by optimizing the way memory is accessed and handled, and improves energy efficiency while reducing the complexity and space required for complex arithmetic operations in hardware.

b) Despite these advancements, the proposed system has notable limitations. The integration of such advanced features into a cohesive system presents a complex challenge, potentially complicating the system-level integration. This complexity might also extend to the compiler technology required to efficiently map high-level application code onto Amber's architecture, potentially limiting its accessibility and ease of use. Furthermore, while Amber shows considerable improvements over existing FPGA-based systems, the paper does not fully address real-world application issues such as scalability and broader technological compatibility.

Problem 3.

A

$D = x_2 + \frac{(4+4)}{x_2} + \frac{x_4}{4} + \frac{16}{x_4} + 4 \cdot \gamma$

$\frac{d}{dx_2} D = 1 - \frac{8}{{x_2}^2} = 0$

$\frac{d}{dx_4} D = \frac{1}{4} - \frac{16}{{x_4}^2} = 0$

$x_2 = 2\sqrt{2}$

$x_4 = 8$

D ≈ 11.65

B

Assume voltage is VDD

$E = \alpha \cdot VDD^2 \cdot C$

$= VDD^2 \cdot \alpha \cdot (1 + x_2 + 4 + x_4 + 20)$

$= 3.583 VDD^2$

C

$1.1 \cdot D_{min} \approx 12.826$

```
import numpy as np

def find_optimal_sizing_v2():

    x, y = np.meshgrid(np.linspace(0.01, 5.0, 500), np.linspa
ce(0.01, 5.0, 500))

    d = x + 8 / x + y / 4 + 16 / y
    condition = d <= 12.826
    sumxy = x + y

    minsum = np.min(sumxy[condition])
    idx = np.argmin(sumxy[condition])
    optimal_x = x[condition].flat[idx]
    optimal_y = y[condition].flat[idx]
    optimal_d = d[condition].flat[idx]

    return optimal_d, optimal_x, optimal_y, minsum

# Execute the function
find_optimal_sizing_v2()
```

Result:

```
(12.82480259870065, 1.74, 2.76, 4.5)
```

Delay is about 12.824802;

optimal $x_2 = 1.74$;

optimal $x_4 = 2.76$;

$E = 3.0VDD^2$