ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KỸ THUẬT MÁY TÍNH

BÁO CÁO BÀI TẬP LỚN

MÔN THIẾT KẾ VI MẠCH SỐ - CE222

THÀNH VIÊN NHÓM:	**NGUYỄN XUÂN TÙNG**	**21521649**	**50%**
	NGUYỄN ĐẶNG ANH KIỆT	**21520312**	**50%**

LỚP CE222.O21

GIẢNG VIÊN HƯỚNG DẪN: NGÔ HIẾU TRƯỜNG

Contents

I. Tổng quát.

Bài tập lớn môn thiết kế vi mạch số CE222 với chủ đề: Chuyển đổi biểu thức Boolean thành Stick diagram.

- Ngôn ngữ lập trình sử dụng: Python.
- Input: Biểu thức Boolean đối xứng.

  Ví dụ: Y = $\overline{(A+B+C)*D}$ thì input là: (A+B+C)*D.

- Output: Stick diagram.

  Điều kiện Input để chương trình hoạt động đúng:

- Biểu thức Boolean phải rõ ràng và tối ưu vì chương trình không có khả năng tối ưu biểu thức.
- Không giới hạn số biến của biểu thức Boolean, tuy nhiên phải lớn hơn 1 và phù hợp nhất là từ 3 ~ 4 biến.

II. Quy trình thực hiện.

Quy trình thực hiện gồm ba giai đoạn:

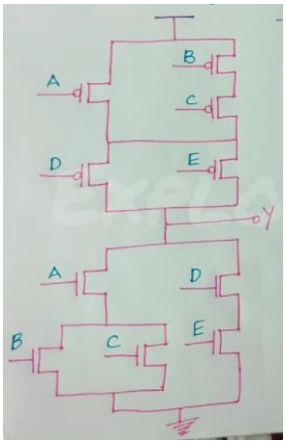
1. Tìm hiểu thuật toán và ngôn ngữ lập trình
2. Hiện thực hóa thuật toán bằng ngôn ngữ Python.
3. Mô phỏng và viết báo cáo.

1. Tìm hiểu thuật toán.

Để mô phỏng Stick Diagram từ biểu thức Boolean đã được tối ưu, ta cần phải có schematic diagram biểu diễn biểu thức Boolean gồm:

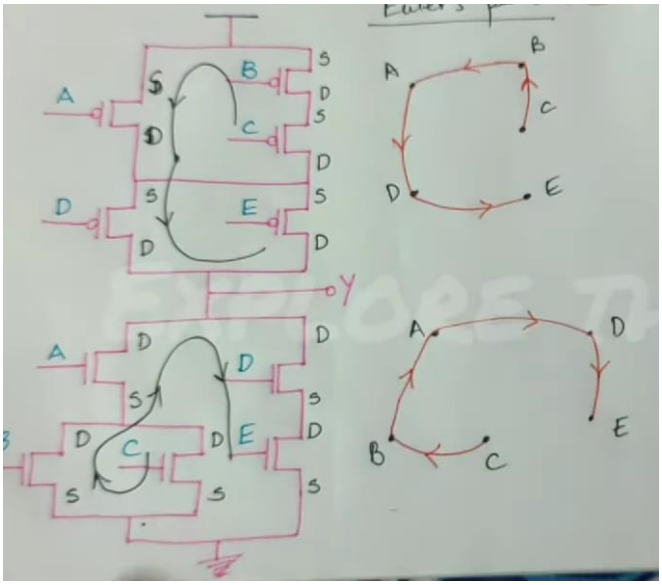
- PMOS pull-up network.
- NMOS pull-down network.
- VDD,GND, Input và Output.

Ví dụ: Y = $\overline{A * (B + C) + D * E}$

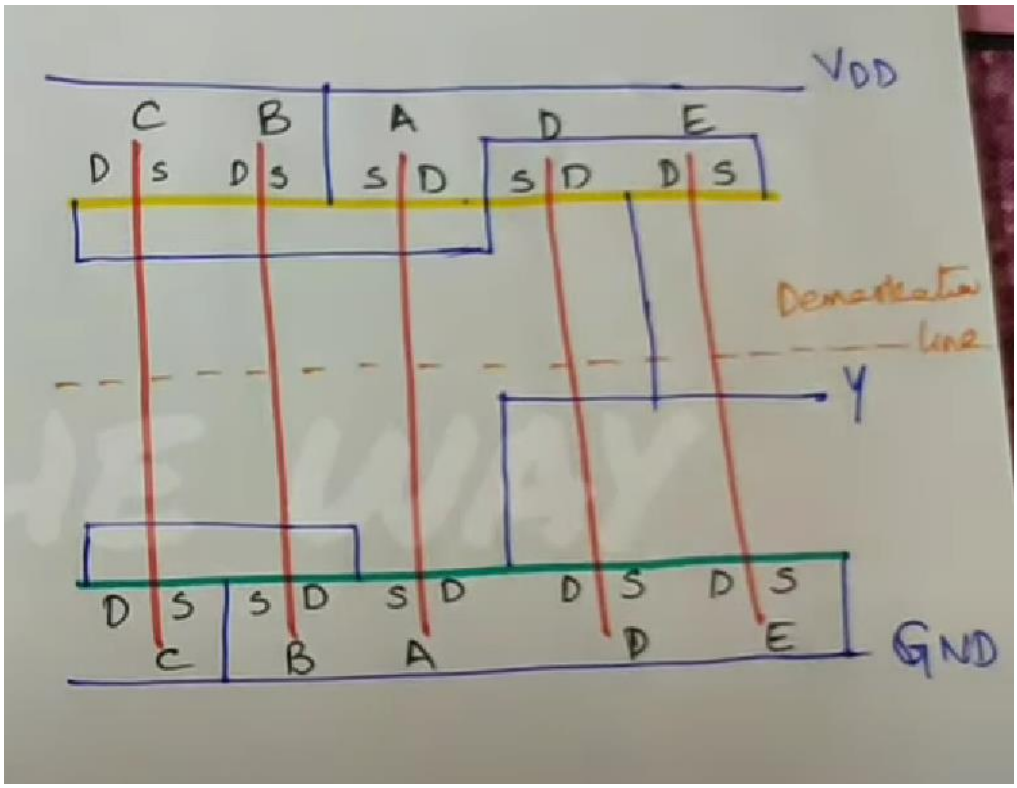


Tiếp theo, ta cần tìm đường đi Euler của schematic diagram cho vùng NMOS pull-down hoặc là PMOS pull-up.

Ở đây, ta sẽ tìm đường đi Euler cho vùng PMOS pull-up đầu tiên và sau đó đường đi Euler của vùng NMOS pull-down sẽ giống với của vùng PMOS pull-up đã tìm được.



Cuối cùng, ta vẽ Stick Diagram đúng với thứ tự các điểm mà đường đi Euler lần lượt đi qua ứng với mỗi vùng.



Như vậy, thuật toán để biểu diễn stick diagram từ biểu thức Boolean là:

Vẽ schematic diagram → Tìm đường đi Euler → Vẽ stick diagram.

2. Hiện thực hóa thuật toán bằng ngôn ngữ Python.

2.1. Vẽ schematic diagram và tìm đường đi Euler trên Python.

Đây là phần khó khăn nhất của bài tập vì từ một biểu thức Boolean ta có thể vẽ được rất nhiều trường hợp của schematic diagram. Nhưng với yêu cầu của bài toán, ta cần phải vẽ được trường hợp của schematic diagram sao cho vùng NMOS pull-down và PMOS pull-up có chung đường đi Euler.

2.1.1. Hướng tiếp cận.

Ta không thể vẽ schematic diagram một cách trực tiếp từ biểu thức Boolean. Thay vào đó, ta sẽ sử dụng Graph (Đồ thị) để mô hình hóa schematic diagram.

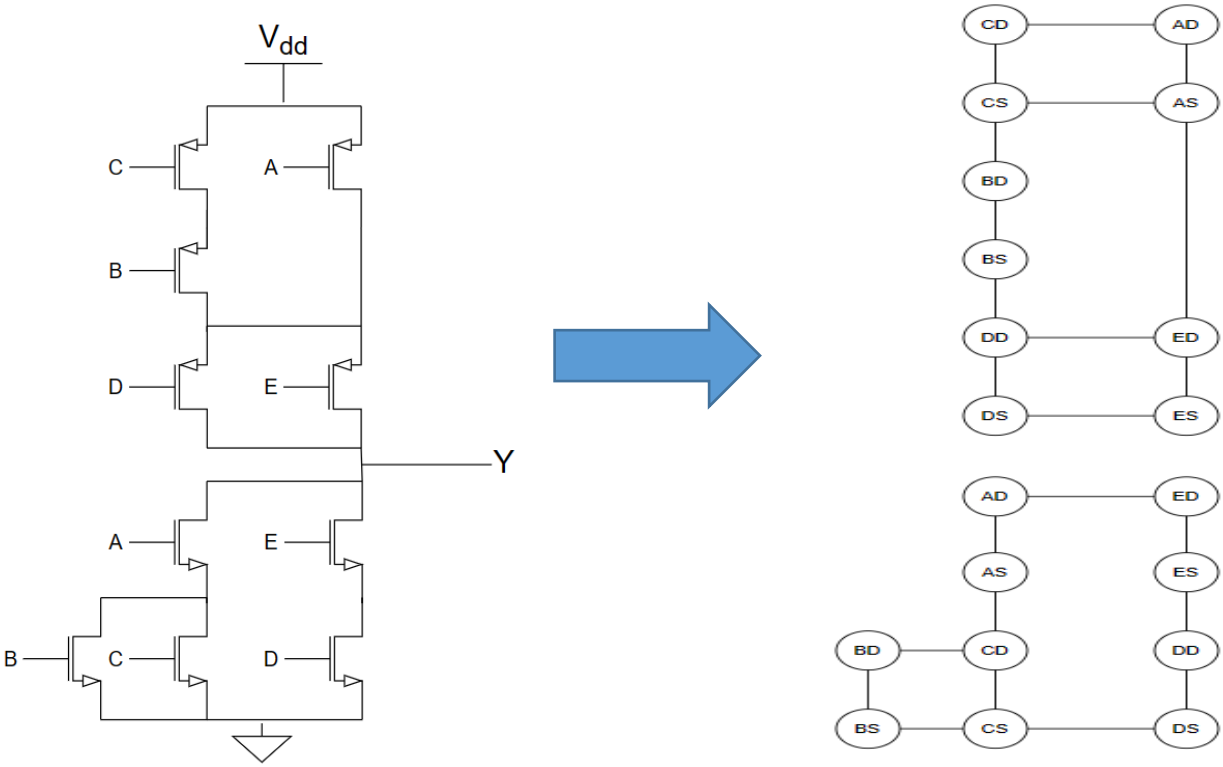
Vì schematic diagram có hai vùng là pull-up và pull-down, do đó trong chương trình, ta sẽ vẽ hai đồ thị riêng biệt tương ứng với hai vùng với các node là các biến đầu vào của biểu thức Boolean.

Y = $\overline{A * (B + C) + D * E}$ thì node là: A, B, C, D ,E.

Tuy nhiên, để biểu diễn các đường đi trong schematic diagram thông qua các cạnh trong đồ thị thì các node trên là chưa đủ. Vì ta sẽ không biết là Source hay Drain của CMOS nối với nhau. Để giải quyết vấn đề trên, ta sẽ thêm vào các node sẽ biểu diễn như sau :

AS, AD, BS, BD, CS, CD, DS, DD, ES, ED.

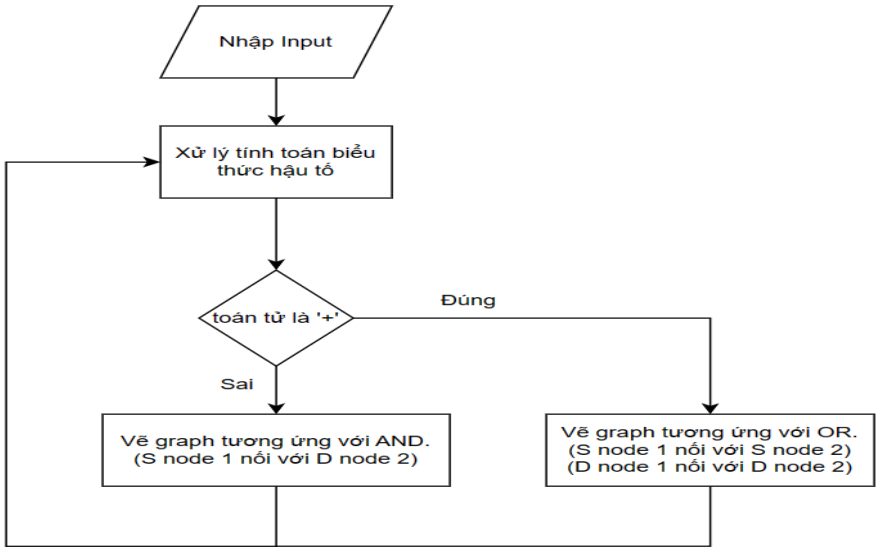
Với ràng buộc là AS và AD lần lượt là đầu Source và Drain của CMOS A và AS và AD sẽ luôn được đi chung với nhau. Như vậy, từ schematic diagram ta có thể mô hình hóa nó trong Python bằng phương pháp sử dụng đồ thị:



2.1.2. Tạo đồ thị mô hình hóa NMOS pull-down network và tìm đường đi Euler.

a. Tạo đồ thị mô hình hóa NMOS pull-down network.

Như đã nói ở trên, ta phải vẽ hai đồ thị sao cho chúng có chung đường đi Euler. Đầu tiên, ta sẽ vẽ vùng pull-down trước theo giải thuật sao đây:



Việc vẽ đồ thị rất phức tạp, giải thuật trên là tóm gọn và đơn giản nhất của thuật toán sử dụng trong Python. Ta cần biết rằng để vẽ đồ thị mô hình hóa phép OR thì Source và Drain của các node song song với nhau sẽ được nối tương ứng: Source – Source, Drain – Drain.

Khi mô hình hóa phép AND thì sẽ có hai trường hợp khả thi: Source node 1 – Drain node 2 hoặc Source node 2 – Drain node 1. Ta phải quy định sử dụng một trường hợp duy nhất.

Xử lý tính toán biểu thức hậu tố là ta sẽ mô phỏng các phép AND và OR theo quy tắc toán tử hậu tố.

Ví dụ: Y = $\overline{A * (B + C) + D * E}$ thì chương trình sẽ lần lượt mô phỏng các phép tính toán trên đồ thị như sau:

1. B + C: Lưu B → node 1, C → node 2. Vẽ đồ thị mô phỏng node 1 OR node 2
2. A * (B + C): Lưu A → node 1, B + C → node 2. Vẽ đồ thị mô phỏng node 1 AND node 2.
3. D * E: Lưu D → node 1, E → node 2. Vẽ đồ thị mô phỏng node 1 AND node 2
4. A * (B + C) + D * E: Lưu A*(B+C) → node 1, D*E → node 2. Vẽ đồ thị mô phỏng node 1 OR node 2.

b. Tìm đường đi Euler cho vùng NMOS pull-down.

Mặc dù thực tế chúng ta sử dụng đường đi Euler nhưng định nghĩa của nó không phù hợp với trong thuật toán. Vì Euler path là đường đi qua hết tất cả các cạnh của đồ thị và chỉ hoạt động đúng khi vẽ schematic diagram, còn trong đồ thị thì không.

Trong sơ đồ mục 2.1.1, muốn tìm được đường đi Euler là bất khả thi vì điều kiện của một đồ thị tồn tại đường đi Euler là đồ thị có đúng hai đỉnh bậc lẻ. Tuy nhiên trong sơ đồ thì ở vùng PMOS pull-up có tới 4 đỉnh bậc 3. Do đó không thể tồn tại đường đi Euler phù hợp cho cả hai đồ thị.

Thay vì sử dụng đường đi Euler, ta sẽ sử dụng đường đi Hamilton thay thế để có thể phù hợp với cả hai đồ thị. Để tránh việc liệt kê thiếu các cạnh trong đồ thị bằng đường đi Hamilton, ta chỉ cần đọc tất cả các cạnh có trong đồ thị và vẽ Stick Diagram tương ứng với Hamilton tìm được.

Tóm lại, đường đi Euler (đường đi qua tất cả các cạnh trong đồ thị) có thể được thay thế bằng cách kết hợp đường đi Hamilton (đường đi qua tất cả các điểm trong đồ thị) với việc liệt kê tất cả các cạnh của đồ thị.

Thuật toán tìm đường đi Hamilton khá đơn giản, tuy nhiên ta cần thêm ràng buộc là các điểm S và D chung một node luôn được đi chung với nhau. Ví dụ: AS-AD, BS-BD, …

Như vậy, với Y = $\overline{A*(B+C)+D*E}$ thì ta có Euler path (trong Python là Hamilton path) kết hợp với liệt kê các cạnh trong đồ thị như sau:

Euler path NMOS: ['BD', 'BS', 'CS', 'CD', 'AS', 'AD', 'ED', 'ES', 'DD', 'DS']

NMOS edges: [('BS', 'BD'), ('BS', 'CS'), ('BS', 'DS'), ('BD', 'CD'), ('BD', 'AS'), ('CS', 'CD'), ('CD', 'AS'), ('AS', 'AD'), ('AD', 'ED'), ('DS', 'DD'), ('DD', 'ES'), ('ES', 'ED')]

Lưu ý: Vẫn còn nhiều ràng buộc khác liên quan tới việc tìm đường đi Hamilton path để chương trình hoạt động chính xác như là điểm bắt đầu, điểm kết thúc. Code chương trình đã được cài đặt hợp lý để tránh xảy ra lỗi và hoạt động hiệu quả nhất.

2.1.3 Tạo đồ thị mô hình hóa PMOS pull-up network và đường đi Euler.

a. Tìm đường đi Euler cho vùng PMOS pull-up.

Đường đi Euler của vùng PMOS pull-up phải tương tự vùng NMOS pull-down. Lưu ý rằng chúng ta cần phải tìm Euler path cho vùng PMOS pull-up trước khi tạo đồ thị cho PMOS pull-up vì khi tạo đồ thị trước sẽ có rất nhiều trường hợp sai lệch.

Ví dụ: A nối tiếp B, ta mong muốn AS – AD – BS – BD. Tuy nhiên khi tạo đồ thị bằng thuật toán đã dùng ở NMOS pull-down sẽ có sai lệch: AD – AS – BD – BS. Mặc dù trường hợp thứ hai không sai nhưng nó không phải là điều ta mong muốn và nó có thể dẫn tới việc Euler path ở PMOS pull – up khác với NMOS pull – down.

Do đó, ta cần tìm Euler path cho vùng PMOS pull – up trước khi tạo đồ thị. Dựa vào Euler path cho vùng NMOS pull-down, ta sẽ dễ dàng tìm được đường đi thích hợp cho PMOS pull-up bằng cách hoán đổi vị trí S và D của node lẻ trong euler path NMOS.

Ví dụ: **Y = $\overline{\boldsymbol{A*(B+C)+D*E}}$**

Euler path NMOS: ['BD', 'BS', 'CS', 'CD', 'AS', 'AD', 'ED', 'ES', 'DD', 'DS']

B → C → A → E → D.

Các node lẻ sẽ là C và E. Do đó đường đi Euler thích hợp cho PMOS sẽ là:

Euler path PMOS: ['BD', 'BS', 'CD', 'CS', 'AS', 'AD', 'ES', 'ED', 'DD', 'DS']

B → C → A → E → D.

Như vậy, ta đã tìm được Euler path chung cho cả hai đồ thị NMOS pull-down và PMOS pull-up.

b. Tạo đồ thị mô hình hóa PMOS pull-up network.

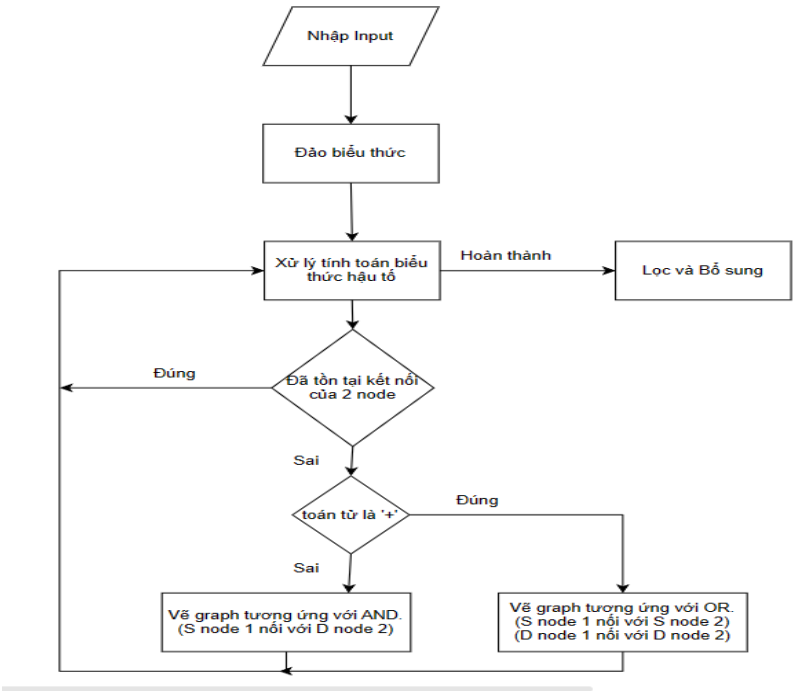
Để tạo đồ thị cho vùng PMOS pull-up thì ta cần dựa vào Euler path PMOS đã tìm được ở mục a. Đồ thị PMOS bắt buộc phải chứa đường đi Euler đã tìm được vì nó là đường đi chung cho cả hai đồ thị. Do đó điều đầu tiên ta cần phải tạo đồ thị ban đầu là đường đi Euler trên.

Sau đó, ta sẽ áp dụng phương pháp tạo đồ thị ở mục a để tạo đồ thị cho vùng PMOS pull-up. Tuy nhiên, ta sẽ cần phải kiểm tra thêm việc đã tồn tại sự kết nối của hai node chưa vì rất có thể các node và các cạnh đã được hình thành từ việc tạo đồ thị dựa trên Euler path có sẵn.

Cuối cùng, ta cần chú ý rằng thuật toán áp dụng để tạo đồ thị NMOS pull-down là tạo từ một đồ thị rỗng ban đầu. Đối với đồ thị PMOS pull – up đã được hình thành dựa trên Euler path có sẵn thì thuật toán trên sẽ ít nhiều tạo ra sai lệch và kết quả không mong muốn.

Để giải quyết vấn đề trên, ta sẽ sử dụng phương pháp lọc và bổ sung dựa trên các cạnh đã tồn tại ở vùng NMOS pull – down. Ta sẽ liệt kê tất cả các node có thể song song và nối tiếp với nhau trong đồ thị PMOS từ đồ thị NMOS. Sau đó ta so sánh chúng với các node và cạnh trong đồ thị PMOS tạo được. Nếu thiếu thì ta sẽ bổ sung thêm vào đồ thị PMOS và nếu sai thì loại bỏ chúng ra khỏi đồ thị PMOS.

Sơ đồ giải thuật:



Như vậy, với Y = $\overline{A * (B + C) + D * E}$ thì ta có Euler path và đồ thị của vùng PMOS pull-up như sau:

Euler path PMOS: ['BD', 'BS', 'CD', 'CS', 'AS', 'AD', 'ES', 'ED', 'DD', 'DS']

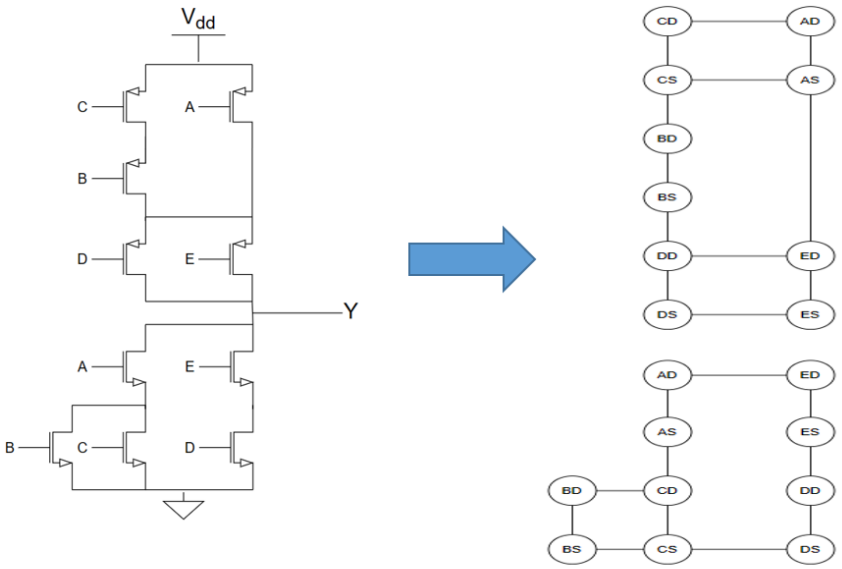
PMOS edges: [('BD', 'BS'), ('BD', 'AD'), ('BS', 'CD'), ('CD', 'CS'), ('CS', 'AS'), ('AS', 'AD'), ('AD', 'ES'), ('ES', 'ED'), ('ES', 'DS'), ('ED', 'DD'), ('DD', 'DS')]

2.1.4. Tìm điểm nối nguồn và output.

Bước cuối cùng để mô hình hóa schematic diagram trên Python bằng phương pháp đồ thị là tìm các điểm nối nguồn và output. Ta cần tìm các điểm nối nguồn và output trong vùng NMOS pull-down và áp dụng giải thuật tương tự cho vùng PMOS pull-up.

Đối với vùng NMOS pull-down thì các điểm có khả năng nối nguồn là các điểm có hậu tố là S và ngược lại, các điểm có khả năng nối output là các điểm có hậu tố là D. Tuy nhiên, không phải tất cả các điểm có hậu tố S sẽ là các điểm nối với nguồn. Các điểm nối nguồn sẽ là điểm mà ngoài việc nối với node D của chính nó thì không tồn tại bất kỳ điểm có hậu tố là D nối với nó, điều ngược lại cũng đúng với điểm nối với output.

Ví dụ:



- Điểm nối nguồn của NMOS pull-down: BS, CS và DS vì chúng có hậu tố là ‘S’ và không tồn tại điểm có hậu tố là ‘D’ kết nối với chúng ngoài chính nó.
- Điểm nối output của NMOS pull-down: AD và BD vì chúng có hậu tố là ‘D’ và không tồn tại điểm có hậu tố là ‘S’ kết nối với chùng ngoài chính nó.

2.1.5. Tổng kết mô hình hóa biểu thức Boolean sang schematic diagram.

Chương trình sẽ thực hiện lần lượt các bước sau đây để mô hình hóa biểu thức Boolean sang schematic diagram trên Python:

1. Xử lý tính toán biểu thức hậu tố đầu vào (NMOS pull-down network).

2. Tạo đồ thị mô hình hóa NMOS pull-down network.
3. Tìm đường đi Euler cho NMOS pull-down network.
4. Tìm đường đi Euler cho PMOS pull-up network.
5. Đảo biểu thức đầu vào và Xử lý tính toán biểu thức hậu tố (PMOS pull-up network)
6. Tạo đồ thị mô hình hóa PMOS pull-up network dựa trên Euler path tìm đường ở mục 4.
7. Tìm các điểm nối nguồn và đất cho cả hai đồ thị.

Terminal trong Python thể hiện kết quả:

```
PS D:\Study\UIT\TKVMS-TruongNH\assignment\boolean-logic-to-stick-diagram> python -u "d:\Study\UIT\TKVMS-TruongNH\assignment\boolean-logic-to-stick-diagram
\expression_euler_path.py"
Expression input:
A*(B+C)+D*E
Euler path NMOS:  ['BD', 'BS', 'CS', 'CD', 'AS', 'AD', 'ED', 'ES', 'DD', 'DS']
Euler path PMOS:  ['BD', 'BS', 'CD', 'CS', 'AS', 'AD', 'ES', 'ED', 'DD', 'DS']
NMOS edges:  [('BS', 'BD'), ('BS', 'CS'), ('BS', 'DS'), ('BD', 'CD'), ('BD', 'AS'), ('CS', 'CD'), ('CD', 'AS'), ('AS', 'AD'), ('AD', 'ED'), ('DS', 'DD'),
('DD', 'ES'), ('ES', 'ED')]
pMOS edges:  [('BD', 'BS'), ('BD', 'AD'), ('BS', 'CD'), ('CD', 'CS'), ('CS', 'AS'), ('AS', 'AD'), ('AD', 'ES'), ('ES', 'ED'), ('ES', 'DS'), ('ED', 'DD'),
('DD', 'DS')]
Node connect Source NMOS (GND):
['BS', 'CS', 'BS', 'DS']
Node connect Output NMOS (OUT):
['AD', 'ED']
Node connect Source PMOS (VCC):
['CS', 'AS']
Node connect Output PMOS (OUT):
['ED', 'DD']
```

2.2. Vẽ Stick diagram bằng ngôn ngữ Python.

Với việc vẽ đồ thị bằng ngôn ngữ Python, ta sử dụng thư viện matplotlib.

2.2.1. Vẽ các đường ndiff , pdiff, VDD, GND và các đường đại diện cho các phần tử logic

Đầu tiên với các đường VDD, GND, ndiff, pdiff ta sẽ ước lượng để vẽ như sau:

```
ax.plot([-4.5, 14], [6, 6], 'b-', linewidth=2)
ax.text(14.5, 6, 'Vdd', verticalalignment='center', fontsize=12)
ax.plot([-4.5, 14], [-1, -1], 'b-', linewidth=2)
ax.text(14.5, -1, 'Gnd', verticalalignment='center', fontsize=12)
ax.plot([-3.5, 13], [5, 5], 'y-', linewidth=2)
ax.plot([-3.5, 13], [0, 0], 'g-', linewidth=2)
```

Trong đoạn code, đường VDD là đường nối hai điểm có tọa độ (-4.5,6) và (14,6), tương tự với đường GND. Đường pdiff là đường nối hai điểm có tọa độ (-3.5,5) và (13,5) và có màu vàng (‘y-’), tương tự với đường ndiff (màu xanh ‘g-‘).

Lưu ý: trong trường hợp số lượng biến quá nhiều, cần tối ưu các tọa độ bằng các biến phụ thuộc vào số lượng phần tử logic.

Tiếp theo, định nghĩa hàm để tách thứ tự các phần tử logic từ Euler path NMOS và chạy vòng lặp for để vẽ các đường tương ứng với từng phần tử:

```
def create_logic_element(euler_path_nmos):
    logic_elements = []
    seen = set()
    for node in euler_path_nmos:
        if node[0] not in seen:
            logic_elements.append(node[0])
            seen.add(node[0])
    return logic_elements
```

```
for element in logic_elements:
    x = 1.5 + logic_elements.index(element) * 2
    count = logic_elements.index(element)*2
    ax.plot([x, x], [-0.5, 5.5], 'r-')
    ax.text(x, 5.6, element, horizontalalignment='center', fontsize=10, color='black')
```

2.2.2. Vẽ đường NMOS, PMOS

Từ Euler path PMOS tìm được ở trên ta chạy vòng lặp for để ghi các đỉnh trong mảng ra, mỗi đỉnh cách đều nhau và tất cả cùng nằm trên đường thẳng tọa độ y là 5 đối với PMOS (đường pdiff) và y là 0 đối với NMOS (đường ndiff).

```
for pmos in euler_path_pmos:
    x = 1 + euler_path_pmos.index(pmos)
    ax.text(x,5.1,pmos,horizontalalignment='center', fontsize=10, color='red')
    coordinates_pmos[pmos] = (x, 5)
```

```
for nmos in euler_path_nmos:
    x = 1 + euler_path_nmos.index(nmos)
    ax.text(x,0.1,nmos,horizontalalignment='center', fontsize=10, color='red')
    coordinates_nmos[nmos] = (x, 0)
```

Đối với nmos ta thực hiện tương tự. Đồng thời mỗi lần ghi thì tạo một danh sách coordinates_pmos(coordinates_nmos) để lưu tên và tọa độ ứng với đỉnh đó.

Tiếp đến, ta sẽ duyệt qua các cạnh của đồ thị PMOS, dựa vào tọa độ x của từng đỉnh (đã lưu trong danh sách coordinates_pmos) để vẽ cạnh, vì các đỉnh liền kề nhau đã có sẵn trong Euler path PMOS nên ta chỉ xét các đỉnh ở xa nhau (tức khoảng cách lớn hơn 1).

```
count = 0.2
connected_pmos = []
for edge in g_pmos.edges:
    node1, node2 = edge
    x1, y1 = coordinates_pmos[node1]
    x2, y2 = coordinates_pmos[node2]
    if(abs(x2-x1) > 1):
        connected_pmos.append((x1,x2))
        if ((x1,x2+1) in connected_pmos) or ((x1,x2-1) in connected_pmos) or (((x2,x1+1) in connected_pmos) or ((x2,x1-1) in connected_pmos)):
            pass
        else:
            if ((x1+1,x2) in connected_pmos) or ((x1+1,x2) in connected_pmos) or (((x2+1,x1) in connected_pmos) or ((x2-1,x1) in connected_pmos)):
                pass
            else:
                ax.plot([x1, x1], [y1, y1-count], 'b-', linewidth=1)
                ax.plot([x2, x2], [y2, y2-count], 'b-', linewidth=1)
                ax.plot([x1, x2], [y1-count, y2-count], 'b-', linewidth=1)
                ax.scatter(x1, y1, color='black', marker='x', s=50)
                ax.scatter(x2, y2, color='black', marker='x', s=50)
    count += 0.2
```

Trong đoạn code trên, điều kiện kiểm tra để có vẽ cạnh hay không là: đối với đỉnh đầu tiên (x1) thì kiểm tra xem đã có kết nối với các đỉnh liền kề với đỉnh thứ hai (x2) hay chưa (x2 -1 và x2 +1), làm tương tự với đỉnh thứ hai (node2).

Nếu điều kiện kiểm tra vừa rồi là đúng thì bỏ qua không vẽ cạnh, nếu sai thì tiếp tục kiểm tra: đối với đỉnh đầu tiên (x1), kiểm tra xem đã có kết nối giữa các điểm liền kề (x1 – 1 và x1 + 1) với đỉnh thứ hai (x2) hay chưa, tương tự với đỉnh thứ hai (node2).

Nếu điều kiện kiểm tra lần 2 vẫn sai (thỏa mãn việc chưa có kết nối giữa hai đỉnh) thì tiến hành vẽ cạnh. Biến count là biến dùng để điều chỉnh độ cao của các đường nối cho các cặp đỉnh tiếp theo.

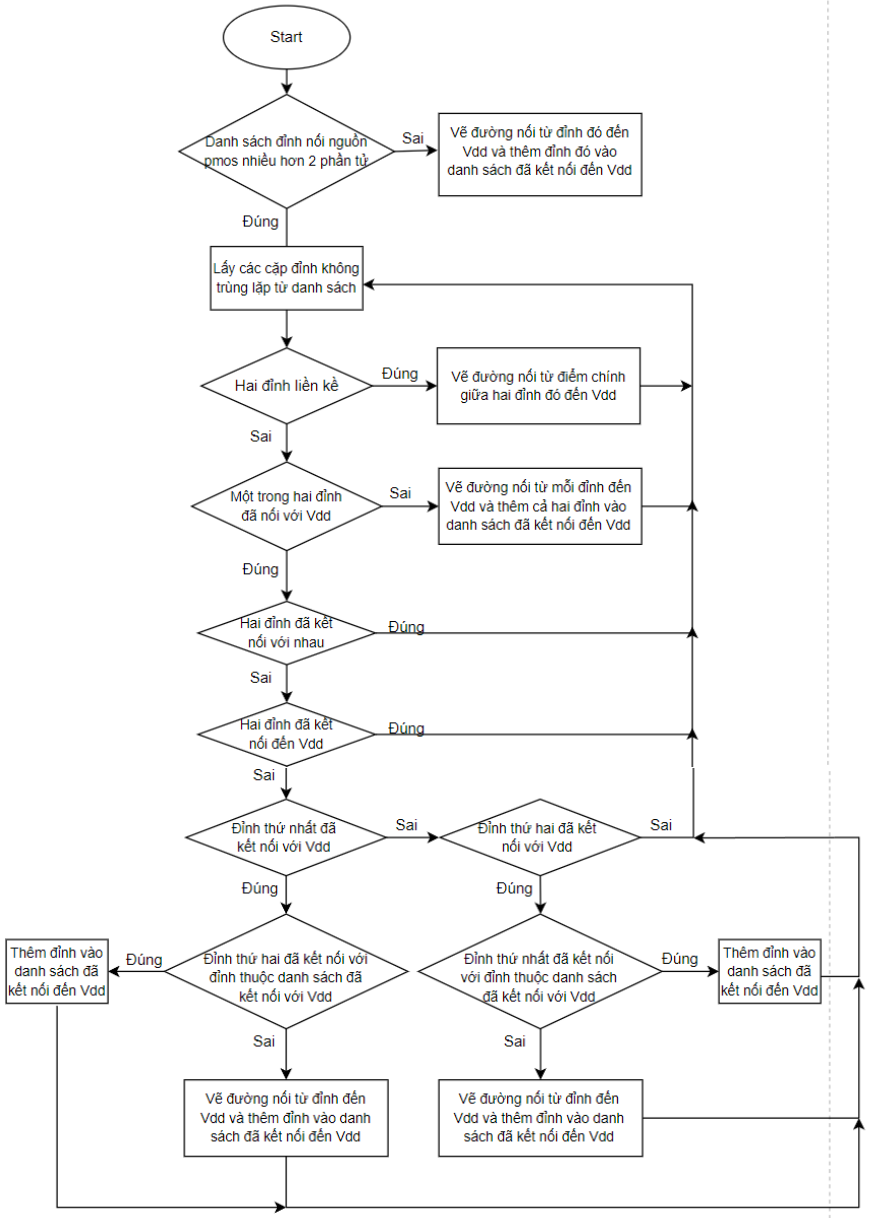
Với đồ thị NMOS ta thực hiện tương tự các công việc trên.

2.2.3. Vẽ các đường nối nguồn và ngõ ra

Trước tiên ta sẽ tạo một danh sách các đỉnh đã kết nối đến VDD (VDD_already_connected), ta sẽ thêm vào danh sách này các đỉnh liền kề và thuộc danh sách các đỉnh kết nối đến VDD(source_nodes_pmos):

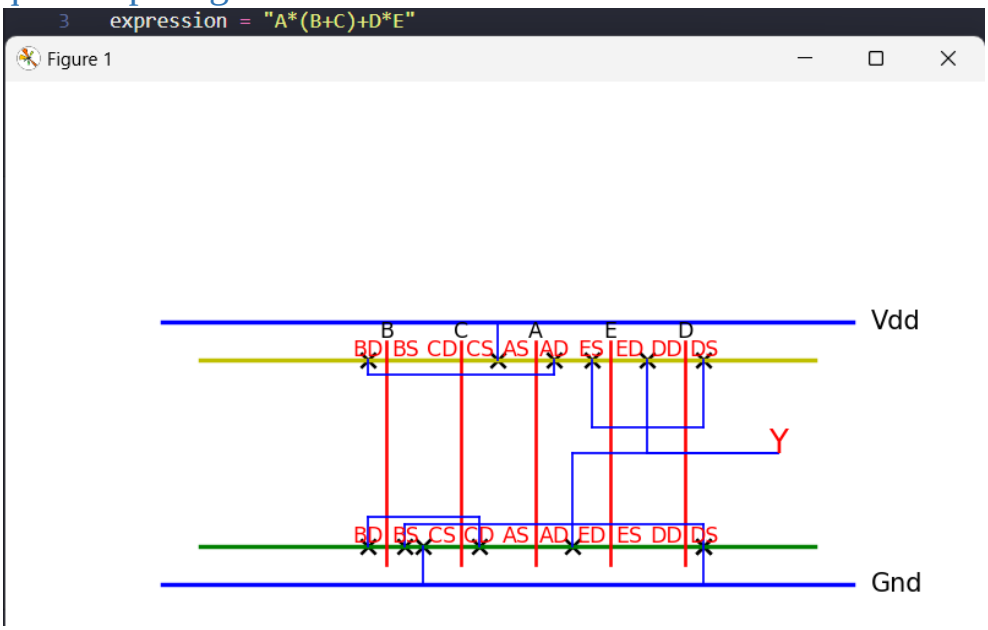
```
vdd_already_connected = []
for node1, node2 in combinations(set(source_nodes_pmos), 2):
    x1,y1 = coordinates_pmos[node1]
    x2,y2 = coordinates_pmos[node2]
    if(abs(x2-x1) == 1):
        vdd_already_connected.append(x2)
        vdd_already_connected.append(x1)
```

Ta thực hiện việc này vì mục đích khi vẽ sẽ ưu tiên các đỉnh liền kề để vẽ duy nhất một đường ở giữa hai đỉnh đó và các đỉnh không liền kề sẽ được vẽ theo sơ đồ giải thuật sau:



Đến đây ta đã hoàn thành xong việc vẽ các đường kết nối với VDD, các đường kết nối đến GND, output còn lại đều được vẽ theo cách tương tự như VDD.

3. Kết quả mô phỏng



III. Tổng kết

Từ bài tập lớn chuyển đổi biểu thức Boolean thành Stick Diagram, nhóm đã đạt được một số yêu cầu sau:

- Biết cách vẽ Stick Diagram của biểu thức Boolean bất kỳ.
- Tìm được đường đi Euler cho vùng NMOS,PMOS.
- Tìm được điểm nối nguồn và output của từng vùng NMOS,PMOS.
- Sử dụng ngôn ngữ Python để thực hiện việc vẽ Stick Diagram.

Tuy nhiên, chương trình của nhóm vẫn còn cần phải được cải thiện ở một số điểm như sau:

- Chưa có khả năng tối ưu biểu thức đầu vào (biểu thức phải được tiền xử lý trước khi nhập vào chương trình).
- Số lượng phần tử logic quá lớn có thể dẫn đến sai sót.
- Việc vẽ các đường như VDD, GND, ndiff và pdiff chỉ là tương đối (phù hợp cho biểu thức 3 – 6 biến