ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KỸ THUẬT MÁY TÍNH

BÁO CÁO ĐỒ ÁN
THIẾT KẾ VI MẠCH SỐ
Lớp: CE222.P21

CHUYỂN ĐỔI BIỂU THỨC BOOLEAN THÀNH MÔ HÌNH STICK DIAGRAM.

Giảng viên hướng dẫn: Ths. Ngô Hiếu Trường

Sinh viên thực hiện: Lý Chí Hải MSSV: 22520385

Contents

I. Tổng quát:

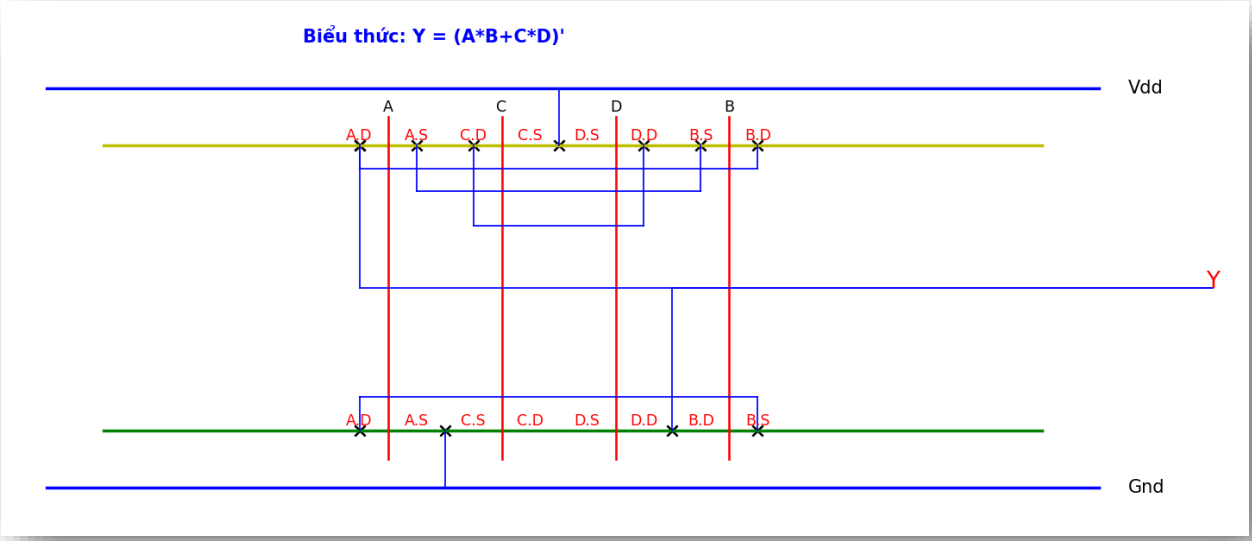
Đồ án giữa kì môn thiết kế vi mạch số CE222 với nội dung đồ án là: Chuyển đổi biểu thức Boolean thành mô hình Stick Diagram.

- Môi trường lập trình: Visual Studio Code.
- Ngôn ngữ xử dụng: Python.
- Input: Biểu thức Boolean đối xứng.

  Ví dụ: Y = (A*B+C*D)'

- Output: Mô hình Stick Diagram.

  Ví dụ:



Hình 1. Ví dụ mô hình Stick Diagram

- Điều kiện Input để chương trình hoạt động đúng:
  - Không giới hạn số biến của biểu thức Boolean, tuy nhiên phải lớn hơn 1 và phù hợp nhất là từ 3 ~ 4 biến.
  - Không xử lý được biểu thức đã tối giản nhưng vẫn có biến trùng lặp.

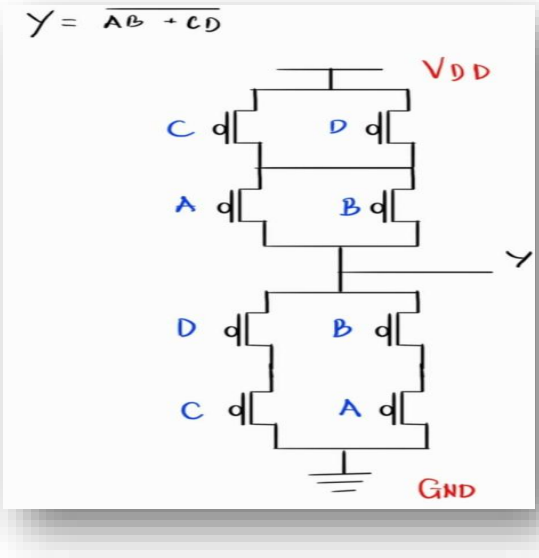
II. Quy trình thực hiện:

Quy trình thực hiện gồm ba giai đoạn:
1. Tìm hiểu thuật toán.
2. Hiện thực hóa thuật toán bằng ngôn ngữ Python.
3. Mô phỏng và viết báo cáo.

1. Tìm hiểu thuật toán:

Khái niệm: Stick diagram là một công cụ quan trọng trong thiết kế vi mạch số ở mức layout sơ bộ, dùng để biểu diễn cấu trúc liên kết và định tuyến của transistor (MOSFET) mà không cần vẽ chi tiết kích thước hay hình học chính xác.

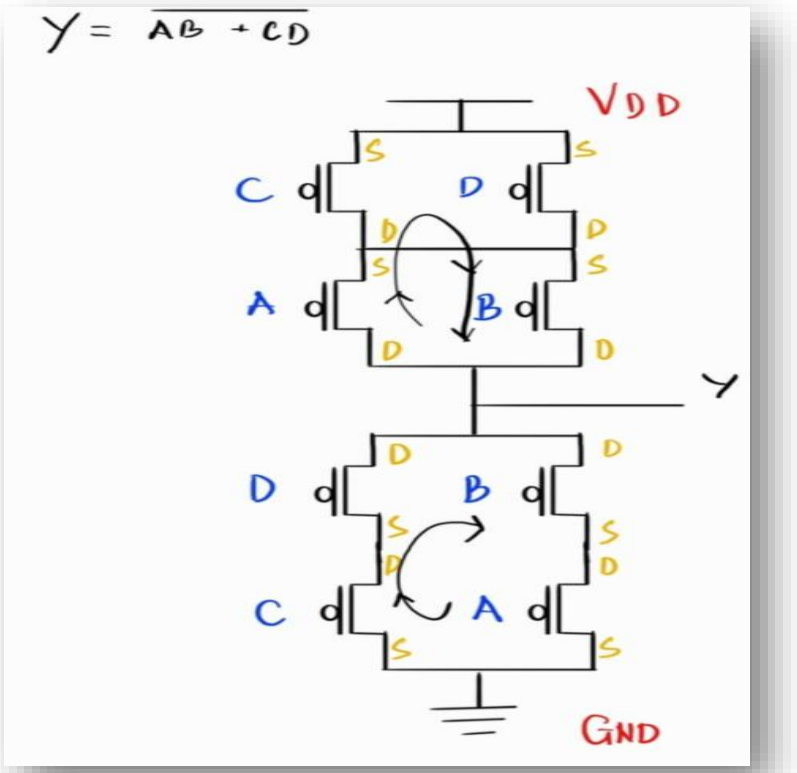
Để mô phỏng Stick Diagram từ biểu thức Boolean, ta cần phải có schematic diagram biểu diễn biểu thức Boolean gồm:
- PMOS pull-up network.
- NMOS pull-down network.
- VDD, GND, Input và Output.
  Ví dụ: Y = (A*B+C*D)'



*Hình 2. Schematic Diagram của Y = (A*B+C*D)'*

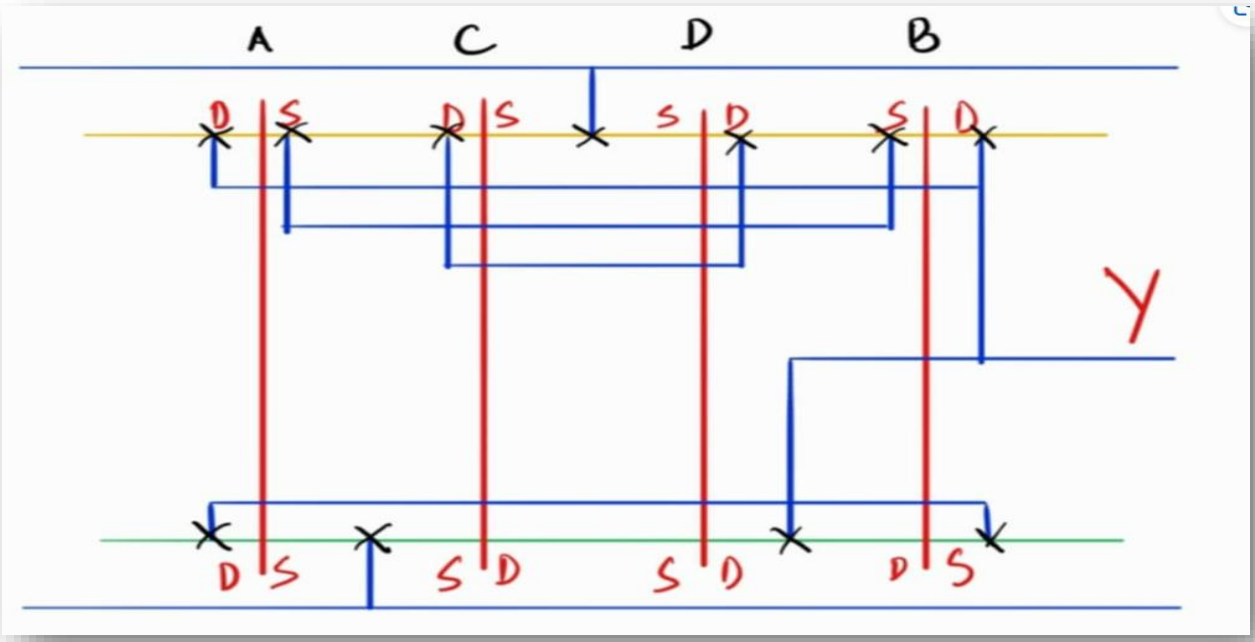
Tiếp theo, ta cần tìm đường đi Euler của schematic diagram cho vùng NMOS pull-down hoặc là PMOS pull-up.

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.



Hình 3. Đường đi Euler của biểu thức

Sau đó, ta vẽ biểu đồ Sitck Diagram theo thức tự các điểm ứng với đường đã tìm được ở phía trên.



*Hình 4. Mô hình Stick Diagram của biểu thức Y = (A*B+C*D)'*

⇨ Như vậy, ta có quy trình thuật toán để biểu dĩa biểu thức Boolean thành sơ đồ Stick Diagram như sau:

Vẽ Schematic Diagram => Tìm đường đi Eluer => Vẽ mô hình Stick Diagram

*Quy ước cách vẽ mô hình Sitck Diagram:

- Màu xanh lam: Biểu diễn VDD, GND và các đường nối giữa các điểm, output.
- Màu đỏ: Các input A, B, C, D,… và tên các S và D, output.
- Màu vàng: Pdiff
- Màu xanh lục: Ndiff
- Màu đen: Các điểm nối và tên các input.

2. Hiện thực hóa bằng ngôn ngữ Python:

2.1. Vẽ Schematic Diagram và tìm đường đi Euler trên Python:

Phần khó khăn nhất của bài tập này nằm ở việc chuyển biểu thức Boolean thành sơ đồ mạch logic (schematic diagram). Mỗi biểu thức Boolean có thể tương ứng với nhiều dạng sơ đồ mạch khác nhau. Tuy nhiên, yêu cầu đặc biệt của đề bài là phải xây dựng sơ đồ sao cho mạng pull-up (PMOS) và mạng pull-down (NMOS) chia sẻ chung một đường đi Euler.

2.1.1. Hướng tiếp cận:

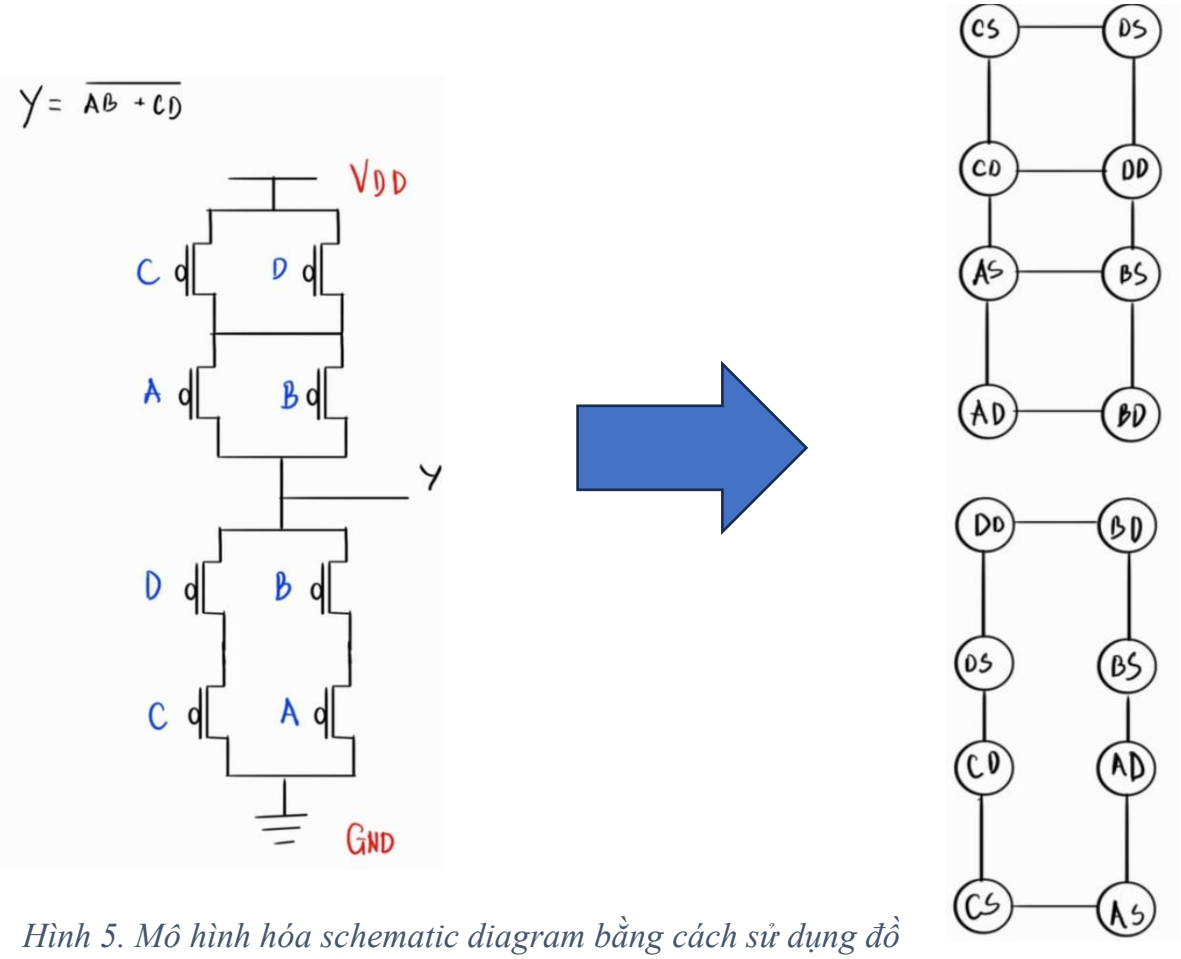
Ta không thể trực tiếp vẽ sơ đồ mạch (schematic diagram) từ biểu thức Boolean một cách hiệu quả. Thay vào đó, ta áp dụng mô hình đồ thị (Graph) để biểu diễn cấu trúc mạch. Do sơ đồ mạch CMOS luôn bao gồm hai mạng riêng biệt là mạng pull-up (PMOS) và mạng pull-down (NMOS), ta sẽ xây dựng hai đồ thị tương ứng cho từng vùng.

Y = (A*B+C*D)’ thì các node là: A, B, C, D.

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

Với ràng buộc là AS và AD lần lượt là đầu Source và Drain của CMOS A. 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:

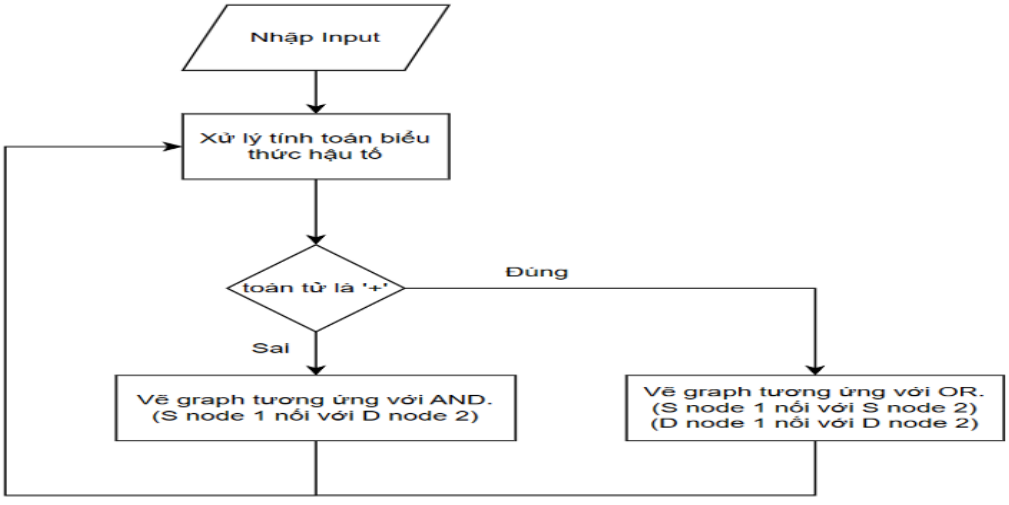


Hình 5. Mô hình hóa schematic diagram bằng cách sử dụng đồ thị

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 pull-down network:

Tạo vùng pull-down theo giải thuật sau:



Hình 6. Giải thuật để tạo vùng pull-down

Việc vẽ đồ thị rất phức tạp, giải thuật trên là tóm gọn và đơn giản nhất của thuật toán sử dụng trong Python. Đối với NMOS pull-down thì ta cần biết rằng để mô hình hóa phép OR hoặc phép AND thì ta cần phải vẽ Source và Drain tương ứng:

- Trong mạng NMOS phép OR thì sẽ nối song song Drain – Drain hoặc Source – Source.
- Phép AND sẽ được nối nối tiếp Source – Drain hoặc Drain – Source.

Việc xử lý tính toán các biểu thức hậu tố là một kỹ thuật phổ biến và đơn giản để tính giá trị của biểu thức mà không cần đến dấu ngoặc.

Ví dụ: ta có biểu thức Y = (A*B+C*D)', để xử lý tính toán biểu thức hậu tố, ta thực hiện các phép tính sau:

- Lưu A vào node1 và lưu B vào node2. Tiến hành lấy node1 AND node2, vẽ đồ thị mô phỏng.
- Lưu C vào node1 và lưu D vào node2. Tiến hành lấy node1 AND node2, vẽ đồ thị mô phỏng.
- Lưu A*B vào node1 và C*D vào node2. Tiến hành lấy node1 OR node2, vẽ đồ thị mô phỏng.

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

Mặc dù trên thực tế chúng ta áp dụng đường đi Euler để hỗ trợ việc vẽ sơ đồ mạch (schematic diagram), định nghĩa chính thức của nó lại không hoàn toàn phù hợp khi sử dụng trong thuật toán. Đường đi Euler được định nghĩa là một đường đi qua tất cả các cạnh của đồ thị đúng một lần, tuy nhiên đặc điểm này chỉ thực sự hữu ích trong bối cảnh bố trí transistor trong sơ đồ mạch, chứ không áp dụng hiệu quả trực tiếp trong việc xử lý đồ thị tổng quát của biểu thức logic.

Trong hình 5 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 thay thế bằng đường đi Hamilton để phù hợp với cả hai đồ thị pull-up và pull-down. Đường đi Hamilton cho phép duyệt qua tất cả các đỉnh của đồ thị đúng một lần, điều này đảm bảo tính khả thi khi áp dụng lên cả hai mạng transistor. Để đảm bảo không bỏ sót cạnh nào trong quá trình triển khai, ta sẽ đọc toàn bộ các cạnh của đồ thị trước, sau đó xây dựng Stick Diagram dựa trên thứ tự các đỉnh trong đường đi Hamilton đã tìm được.

Tóm lại, thay vì sử dụng đường đi Euler — vốn yêu cầu đi qua tất cả các cạnh của đồ thị — ta có thể thay thế bằng đường đi Hamilton, tức là đi qua tất cả các đỉnh một lần duy nhất. Để không bỏ sót các cạnh, ta kết hợp đường đi Hamilton với việc liệt kê toàn bộ các

cạnh của đồ thị trước đó. Cách tiếp cận này vừa phù hợp với việc mô hình hóa sơ đồ stick, vừa đảm bảo khả thi trong cả hai mạng pull-up và pull-down.

Thuật toán tìm đường đi Hamilton nhìn chung khá đơn giản. Tuy nhiên, trong trường hợp này, ta cần thêm một ràng buộc đặc biệt: các điểm nguồn và đích (S và D) của cùng một biến phải luôn được đi liền kề với nhau trong đường đi. Ví dụ, nếu có cặp biến A thì các node AS và AD phải xuất hiện liền nhau trong thứ tự duyệt: AS–AD hoặc AD–AS. Tương tự cho BS–BD, CS–CD, …

Như vậy, với Y = (A*B+C*D)′ thì ta có Hamilton path kết hợp với liệt kê các cạnh trong đồ thị như sau:

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

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

Lưu ý: Việc tìm đường đi Hamilton vẫn còn nhiều ràng buộc để đảm bảo chương trình hoạt động chính xác. Một số yếu tố cần xem xét bao gồm: điểm bắt đầu, điểm kết thúc, và quy tắc đi liền nhau đối với các cặp điểm S và D cùng node (ví dụ: AS–AD, BS–BD, ...). Thuật toán đã được thiết kế và cài đặt một cách hợp lý nhằm tránh lỗi và đảm bảo hiệu quả xử lý cao nhất trong quá trình dựng stick diagram từ biểu thức logic.

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

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

Để tạo đồ thị cho vùng PMOS (pull-up network), bước đầu tiên là dựa trên đường đi Euler đã được xác định trong mục a. Đây là đường đi bắt buộc phải có trong đồ thị PMOS vì nó đóng vai trò là đường dẫn chung cho cả hai mạng transistor (pull-up và pull-down). Do đó, khi khởi tạo đồ thị PMOS, ta cần đảm bảo rằng đồ thị ban đầu phải chứa đầy đủ các cạnh và đỉnh thuộc đường đi Euler nói trên.

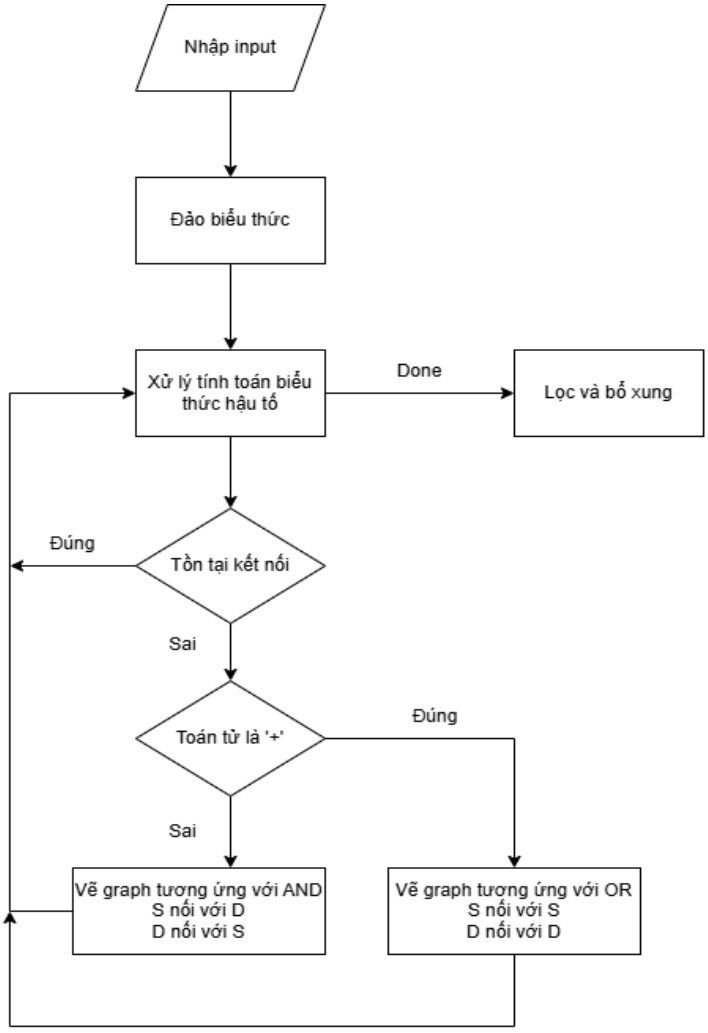
Sau khi khởi tạo đồ thị ban đầu từ đường đi Euler, ta tiếp tục áp dụng phương pháp tạo đồ thị đã trình bày trong mục a để xây dựng đầy đủ đồ thị cho vùng PMOS (pull-up network). Tuy nhiên, trong quá trình thêm các cạnh, cần kiểm tra xem kết nối giữa hai node đã tồn tại hay chưa, bởi vì một số node và cạnh có thể đã được tạo sẵn từ bước khởi tạo dựa trên đường đi Euler. Việc này giúp tránh tạo các cạnh trùng lặp không cần thiết, đảm bảo đồ thị được xây dựng chính xác và tối ưu.

Cuối cùng, cần lưu ý rằng thuật toán áp dụng để tạo đồ thị cho vùng NMOS (pull-down network) khác biệt so với PMOS. Trong trường hợp NMOS, đồ thị bắt đầu từ một đồ thị rỗng và được xây dựng từng bước. Ngược lại, đối với đồ thị PMOS pull-up, được hình thành từ một đường đi Euler đã có sẵn, thuật toán có thể dẫn đến một số sai lệch trong quá trình xây dựng, bởi các ràng buộc và kết nối đã tồn tại từ trước. Vì vậy, việc áp dụng thuật

toán này cho đồ thị PMOS cần được điều chỉnh kỹ càng để tránh kết quả không chính xác hoặc không mong muốn.

Để giải quyết vấn đề này, chúng ta sẽ áp dụng phương pháp lọc và bổ sung dựa trên các cạnh đã tồn tại trong vùng NMOS pull-down. Cụ thể, chúng ta sẽ liệt kê tất cả các node có thể kết nối song song và nối tiếp trong đồ thị PMOS từ đồ thị NMOS. Sau đó, tiến hành so sánh chúng với các node và cạnh trong đồ thị PMOS đã được tạo ra. Nếu phát hiện thiếu các node hoặc cạnh, chúng ta sẽ bổ sung thêm vào đồ thị PMOS. Ngược lại, nếu có những node hoặc cạnh sai lệch, chúng ta sẽ loại bỏ chúng khỏi đồ thị PMOS để đảm bảo tính chính xác và phù hợp với yêu cầu thiết kế.

Từ đó ta sẽ có được sơ đồ giải thuật sau:



Hình 7. Giải thuật để tạo vùng pull-up

b. Tìm đường đi Euler cho PMOS pull-up:

Đường đi Euler của vùng PMOS pull-up cần tương đồng với đường đi của vùng NMOS pull-down. Do đó, điều quan trọng là phải xác định Euler path cho vùng PMOS pull-up trước khi tiến hành xây dựng đồ thị PMOS. Việc tạo đồ thị trước có thể dẫn đến nhiều trường hợp sai lệch trong kết nối, ảnh hưởng đến tính đồng nhất giữa hai vùng.

Ví dụ, nếu mong muốn kết nối A nối tiếp B theo thứ tự: AS – AD – BS – BD, thì thuật toán tạo đồ thị như đã áp dụng cho vùng NMOS pull-down có thể vô tình tạo ra thứ tự không như mong muốn, chẳng hạn: AD – AS – BD – BS. Dù thứ tự này không sai về mặt logic, nhưng nó có thể khiến Euler path trong vùng PMOS khác với vùng NMOS, từ đó phá vỡ yêu cầu về đường đi chung cho cả hai vùng.

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 chẵn trong euler path NMOS.

Ví dụ: Y = (A*B+C*D)’

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

A => C => D => B

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

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

A => C => D => B

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

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

2.1.4. Tìm điểm nối nguồn vào output:

Bước cuối cùng trong việc mô hình hóa *schematic diagram* bằng Python thông qua phương pháp đồ thị là xác định các điểm nối với nguồn VDD, GND và điểm nối output (Y). Việc này cần được thực hiện cho cả vùng NMOS pull-down và áp dụng tương tự cho vùng PMOS pull-up.

- Các điểm có hậu tố "S" (*source*) là các ứng viên kết nối với nguồn (VDD, GND)
- Ngược lại, các điểm có hậu tố "D" (*drain*) là các ứng viên có thể kết nối với output Y.

Tuy nhiên, không phải mọi điểm có hậu tố S đều được xem là điểm nối với nguồn. Một điểm có hậu tố "S" chỉ được chọn làm điểm nối nguồn nếu thỏa mãn điều kiện sau:

- Nó chỉ kết nối với điểm D của chính nó
- Không có bất kỳ điểm D nào khác trong đồ thị nối với nó.

Điều kiện ngược lại cũng được áp dụng cho việc xác định điểm nối với output:

- Một điểm có hậu tố D chỉ được xem là điểm nối output nếu nó chỉ kết nối với điểm S của chính nó, và không bị kết nối bởi điểm D nào khác.

Việc xác định chính xác các điểm này là quan trọng để mô phỏng hoạt động logic đúng của mạch CMOS.

Ví dụ: Hình 5, mục 2.1.1.

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

7. Tìm các điểm nối nguồn và đất cho cả hai đồ thị.

Terminal trong Python thể hiện kết quả:

```
Y = (A*B+C*D)'
Euler path PMOS:  ['AD', 'AS', 'CD', 'CS', 'DS', 'DD', 'BS', 'BD']
Euler path NMOS:  ['AD', 'AS', 'CS', 'CD', 'DS', 'DD', 'BD', 'BS']
PMOS edges:  [('AD', 'AS'), ('AD', 'BD'), ('AS', 'CD'), ('AS', 'BS'), ('BD', 'BS'), ('CD', 'CS'), ('CD', 'DD'), ('BS', 'DD'), ('CS', 'DS'), ('DD', 'DS')]
NMOS edges:  [('AS', 'AD'), ('AS', 'CS'), ('AD', 'BS'), ('CS', 'CD'), ('BS', 'BD'), ('BD', 'DD'), ('DD', 'DS'), ('CD', 'DS')]
Node connect Source PMOS (VCC):
['CS', 'DS']
Node connect Source NMOS (GND):
['AS', 'CS']
Node connect Output PMOS (OUT):
['AD', 'BD']
Node connect Output NMOS (OUT):
['BD', 'DD']
```

Hình 8. Terminal hiển thị kết quả trong Python

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, ghi lại biểu thức nhập vào và vẽ các đường đại diện cho các phần tử logic:

Đầu tiên với các đường VDD, GND, ndiff, pdiff và hiển thị biểu thức đã nhập ta sẽ ước lượng để vẽ như sau:

```
ax.text(0, 6.8, "Biểu thức: " + expression_str, fontsize=12, fontweight='bold', color='blue')
ax.set_aspect('equal')
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)
```

Hình 9. Code vẽ Ndiff, Pdiff, VDD, GND và hiển thị biểu thức đã nhập

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-‘).

Biểu thức được nhập sẽ được hiển thị ở tọa độ (0, 6.8) và có màu xanh.

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')
```

Hình 10. Code để vẽ đường thẳng tương ứng với từng phần tử nhập vào

2.2.2. Vẽ đường NMOS và 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).

```
coordinates_pmos = {}
for pmos in euler_path_pmos:
    x = 1 + euler_path_pmos.index(pmos)

    pmos_base = pmos[:-1]  # bỏ ký tự S/D cuối
    label = var_mapping.get(pmos_base, pmos_base) + "." + pmos[-1]
    ax.text(x, 5.1, label, horizontalalignment='center', fontsize=10, color='red')
    coordinates_pmos[pmos] = (x, 5)
```

```
coordinates_nmos = {}
for nmos in euler_path_nmos:
    x = 1 + euler_path_nmos.index(nmos)
    (variable) nmos_base: Any
    nmos_base = nmos[:-1]
    label = var_mapping.get(nmos_base, nmos_base) + "." + nmos[-1]
    ax.text(x, 0.1, label, horizontalalignment='center', fontsize=10, color='red')
    coordinates_nmos[nmos] = (x, 0)
```

Hình 11. Code để tạo các đỉnh

Tiếp theo, ta sẽ duyệt qua tất cả các cạnh của đồ thị PMOS để tiến hành vẽ sơ đồ kết nối (Stick Diagram). Việc này sẽ dựa trên tọa độ x của từng đỉnh, vốn đã được lưu trong danh sách coordinates_pmos.

Do các đỉnh liền kề nhau trong đường đi Euler PMOS đã được xử lý trước đó, nên chỉ cần xét các đỉnh không liền kề—tức là những đỉnh có khoảng cách tọa độ x lớn hơn 1. Việc này giúp tránh vẽ lại các cạnh đã có, đồng thời đảm bảo rằng mọi kết nối cần thiết giữa các node xa nhau vẫn được thể hiện đầy đủ trong sơ đồ.

```
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
coordinates_nmos = {}
```

Hình 12. Code vẽ các đường PMOS

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 đường nối nguồn và ngõ ra:

```
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)
```

Hình 13. Xác định các điểm đã được kết nối trực tiếp

Với mỗi cặp node trong source_nodes_pmos, nếu tọa độ x của chúng chênh lệch đúng 1 đơn vị, tức là liền kề nhau, thì ta xem như chúng đã được kết nối (theo đường đi Euler). Hai điểm đó sẽ được thêm vào danh sách vdd_already_connected.

```
if(abs(x2-x1) == 1):
    x3 = (x1 + x2) /2
    y3 = (y1 + y2) /2
    ax.plot([x3,x3],[5,6],'b', linewidth=1)
    ax.scatter(x3, y3, color='black', marker='x', s=50)
```

Hình 14. Nối các node liền kề bằng đoạn đường thẳng trung gian

Nếu hai node liền kề, ta vẽ một đoạn thẳng đứng từ giữa khoảng của chúng lên VDD, đánh dấu điểm nối bằng dấu x.

```
if x1 in vdd_already_connected or x2 in vdd_already_connected:
    if (x1,x2) in connected_pmos or (x2,x1) in connected_pmos:
        continue
    else:
        if(x1 in vdd_already_connected and x2 in vdd_already_connected):
            continue
        if x1 in vdd_already_connected:
            for i in vdd_already_connected:
                if (x2,i) in connected_pmos or (i,x2) in connected_pmos:
                    vdd_already_connected.append(x2)
                    continue
            ax.plot([x2,x2],[5,6],'b', linewidth=1)
            ax.scatter(x2, y2, color='blue', marker='x', s=50)
            vdd_already_connected.append(x2)
            continue
        if x2 in vdd_already_connected:
            for i in vdd_already_connected:
                if (x1,i) in connected_pmos or (i,x1) in connected_pmos:
                    vdd_already_connected.append(x1)
                    continue
            ax.plot([x1,x1],[5,6],'b', linewidth=1)
            ax.scatter(x1, y1, color='blue', marker='x', s=50)
            vdd_already_connected.append(x1)
            continue
```

Nếu một trong hai node đã được nối VDD, kiểm tra xem có node nào đã được nối gián tiếp qua các node khác chưa

```
else:
    if (x1,x2) in connected_pmos or (x2,x1) in connected_pmos:
        ax.plot([x1,x1],[5,6],'b', linewidth=1)
        ax.scatter(x1, y1, color='blue', marker='x', s=50)
        vdd_already_connected.append(x1)
        vdd_already_connected.append(x2)
    else:
        ax.plot([x1,x1],[5,6],'b', linewidth=1)
        ax.scatter(x1, y1, color='blue', marker='x', s=50)
        ax.plot([x2,x2],[5,6],'b', linewidth=1)
        ax.scatter(x2, y2, color='blue', marker='x', s=50)
        vdd_already_connected.append(x1)
        vdd_already_connected.append(x2)
```

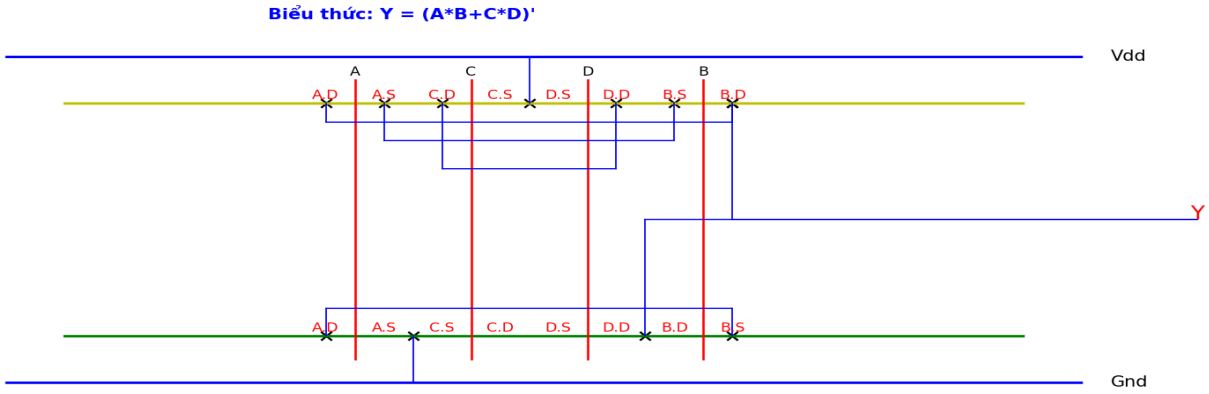
Nếu cả 2 đã được kết nối thì dẩy 2 node vào stack. Nếu **cả hai node chưa nối**, thì nối cả hai lên VDD riêng biệt.

```
else:
    for node in source_nodes_pmos:
        x1,y1 = coordinates_pmos[node]
        ax.plot([x1,x1],[5,6],'b', linewidth=1)
        ax.scatter(x1, y1, color='black', marker='x', s=50)
```

Nếu chỉ có một node nguồn, thì ta vẽ trực tiếp đoạn nối từ node đó lên VDD, đơn giản.

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:



Hình 15. Kết quả mô phỏng

III. Tổng kết:

Từ đồ án giữa kì chuyển đổi biểu thức Boolean thành Stick Diagram, 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:

• Số lượng phần tử logic quá lớn có thể dẫn đến sai sót, không xử lí được biến trùng.

• Việc vẽ các đường như VDD, GND, ndiff và pdiff chỉ là tương đối (phù hợp cho biểu thức 3 – 6 biến