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: Nguyễn Nhật Tân MSSV: 22521307

MỤC LỤC

I. Tổng quát về đề tài:

- Tên đề tài: **Chuyển đổi biểu thức Boolean thành Stick diagram.**
- Ngôn ngữ lập trình sử dụng: Python.
- Đầu vào là một biểu thức Boolean.
    - Ví dụ: $Y = \overline{A * (B + C + D)}$ thì Input là $A * (B + C + D)$
- Đầu ra là mô hình Stick Diagram của biểu thức.
- Yêu cầu của đề tài:
    - Vẽ được mô hình Stick Diagram của các biểu thức có số lượng biến từ 3 đến 4 biến.
    - Có thể xử lý được các biểu thức có chứa biến trùng để vẽ được mô hình Stick Diagram.

II. Tìm hiểu về thuật toán:

Để mô phỏng Stick Diagram từ biểu thức Boolean được đưa vào, ta phải thực hiện tuần tự các bước sau:

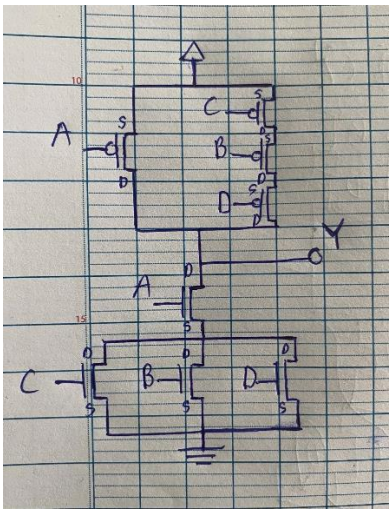
- Tối ưu biểu thức về dạng đơn giản nhất để giảm số lượng biến trùng.
- Vẽ Schemtic Diagram biểu diễn biểu thức Boolean vừa được rút gọn.
    - Biểu diễn mạng **PMOS** (pull-up) và **NMOS** (pull-down).
    - Kết nối VDD,GND, Input và Output vào các điểm thích hợp.
- Tìm đường đi Euler của schematic diagram cho vùng NMOS pull-down hoặc là PMOS pull-up và sau đó đường đi euler của vùng còn lại sẽ giống với vùng vừa tìm được.
- Vẽ Stick Diagram đúng với thứ tự các điểm mà đường đi Euler lần lượt đi qua ứng với mỗi vùng.

⇨ Như vậy kết luận thuật toán để biểu diễn Stick Diagram từ biểu thức Boolean là:
Tối ưu biểu thức → Vẽ Schematic Diagram → Tìm đường đi Euler → Vẽ Stick Diagram.

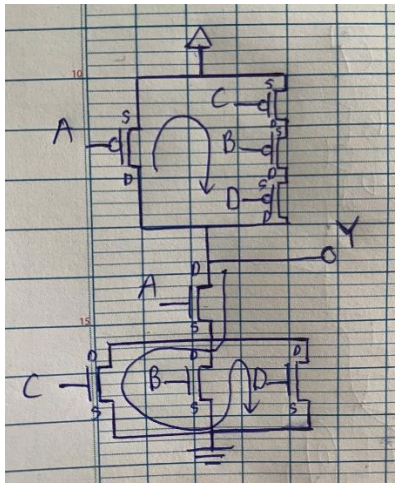
<u>Ví dụ</u>: Input $Y = \overline{A * B + A * C + A * D}$

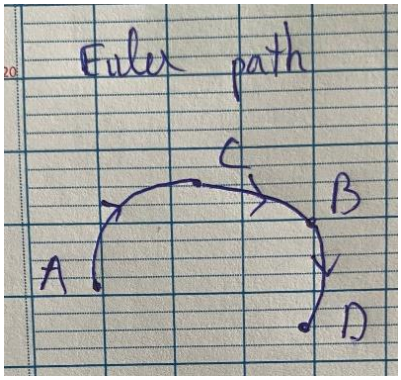
B1: Rút gọn biểu thức $Y = \overline{A * B + A * C + A * D} = \overline{A * (B + C + D)}$

B2: Vẽ Schematic Diagram.

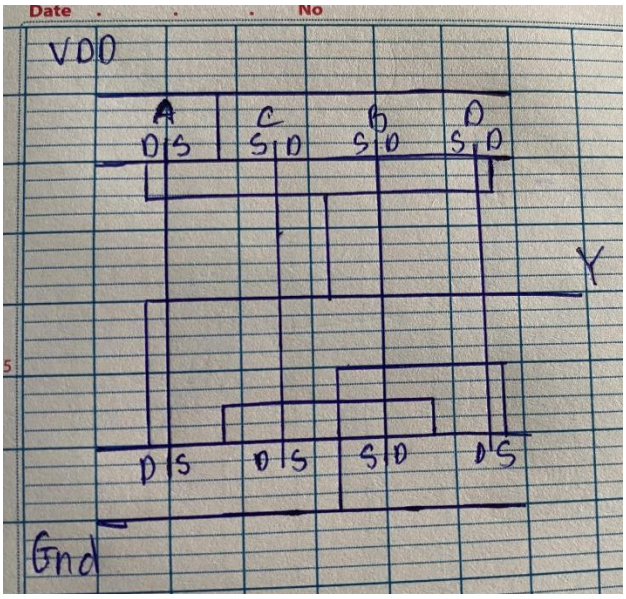


B3: Tìm đường đi Euler





B4: Vẽ Stick Diagram.



III. Hiện thực hóa bài toán bằng ngôn ngữ Python:

1. Tối ưu biểu thức:

Hiện tại khối **tối ưu biểu thức** được áp dụng để tối ưu các biểu thức có thể rút gọn về dạng đơn giản nhất và không còn biến bị trùng.

1.1 Ý tưởng:

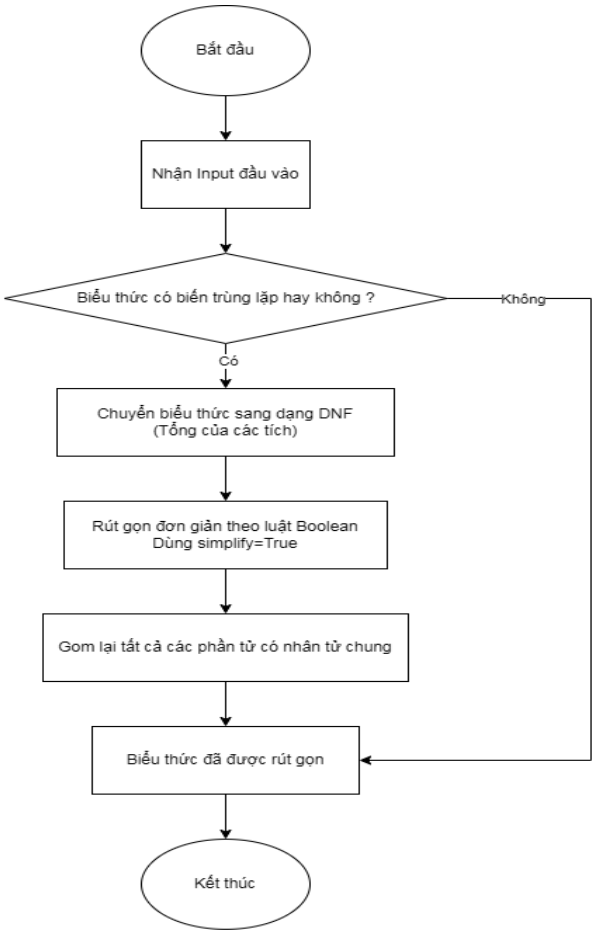
Với các biểu thức có thể rút gọn được tối ưu thì ta sẽ tiến hành áp dụng các định luật Boolean để rút gọn.

Ví dụ:

- (A+B)*(A+B+C) = A+B
- A*B+A*C+A*D = A*(B+C+D)

Đối với các biểu thức có thể rút gọn bằng biểu thức Boolean thì ta sẽ rút gọn một cách dễ dàng nhưng với các biểu thức như A*B+A*C+A*D thì ta cần dùng thêm hàm tìm nhân tử chung và gom các nhóm có nhân tử chung này lại với nhau để rút gọn. Như ví dụ thì sau khi rút gọn sẽ là A*(B+C+D), phần tử A đã được rút gọn từ lặp 3 lần chỉ còn xuất hiện một lần.

1.2 Sơ đồ thuật toán:



- Đây là thuật toán đơn giản quá trình rút gọn biểu thức.
- Thực tế trong chương trình thì cần phải có nhiều chức năng trung gian như thêm dấu * với các biểu thức chỉ viết AB, sắp xếp lại biểu thức theo thứ A, B, C, D,…

1.3 Kết quả thực hiện được:

Biểu thức: A*B+A*C+A*D = A*(B+C+D)

```
PS C:\Users\Tan\Downloads\Stick_2> python Source.py
A*B+A*C+A*D
A*(B + C + D)
```

Biểu thức: (A+B)*(A+B+C) = A+B

```
PS C:\Users\Tan\Downloads\Stick_2> python Source.py
(A+B)*(A+B+C)
A + B
```

2. Vẽ Schematic Diagram và tìm đường đi Euler bằng Python:

2.1 Ý tưởng:

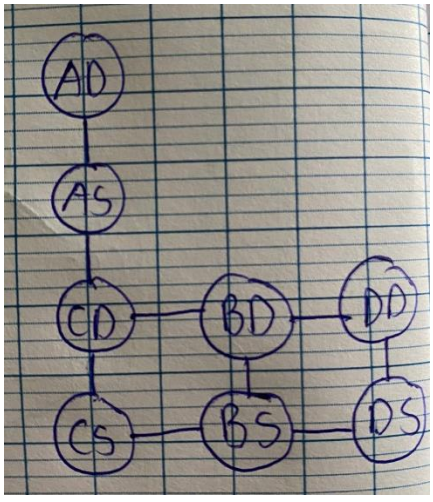
- Ta không thể trực tiếp vẽ sơ đồ mạch (schematic diagram) từ biểu thức Boolean một cách hiệu quả.
- Thay vào đó, ta áp dụng mô hình đồ thị (Graph) để biểu diễn cấu trúc mạch. Do sơ đồ mạch CMOS luôn bao gồm hai mạng riêng biệt là mạng pull-up (PMOS) và mạng pull-down (NMOS), ta sẽ xây dựng hai đồ thị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

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

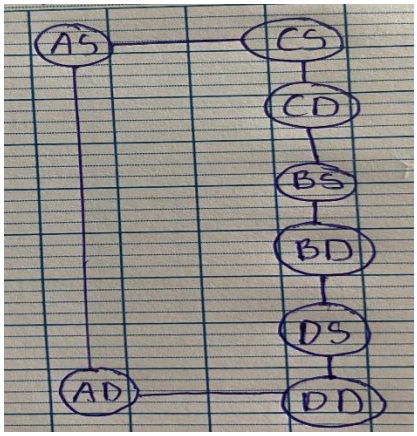
- **Biểu thức này sẽ có các node là A, B, C, D.**
- Tuy nhiên, để biểu diễn các đường đi trong schematic diagram thông qua các cạnh trong đồ thị thì các node trên là chưa đủ. Vì ta sẽ không biết là Source hay Drain của CMOS nối với nhau. Để giải quyết vấn đề trên, ta sẽ thêm vào các node sẽ biểu diễn 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 và (AS, AD) sẽ luôn được đi chung với nhau. Tương tự với các CMOS B, C, D. Như vậy, từ schematic diagram ta có thể mô hình hóa nó trong Python bằng phương pháp sử dụng đồ thị:
    - NMOS:



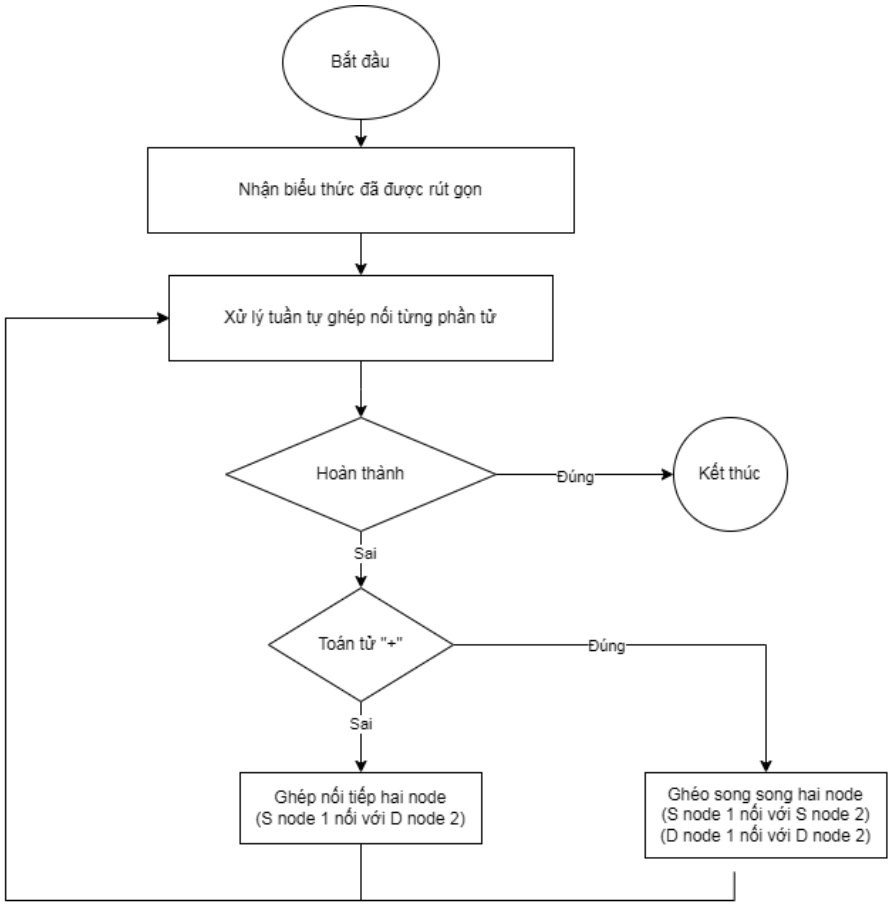
- PMOS:



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

2.2.1 Tạo đồ thị mô hình hóa NMOS pull-down network:

- **Sơ đồ thuật toán:**



- **Giải thích thuật toán:**

  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:

  - Phép OR thì sẽ nối Drain node 1 – Drain node 2 và Source node 1 – Source node 2.
  - Phép AND sẽ được nối nối tiếp Source node 1 – Drain node 2 hoặc Drain node 1– Source node 2.

  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: Biểu thức $Y = \overline{A * (B + C + D)}$, để xử lý tính toán biểu thức hậu tố, ta thực hiện các phép tính sau:

- Lưu B vào node1 và lưu C vào node2. Tiến hành lấy node1 OR node2, vẽ đồ thị mô phỏng.
- Lưu (B+C) vào node1 và lưu D vào node2. Tiến hành lấy node1 OR node2, vẽ đồ thị mô phỏng.
- Lưu A vào node1 và (B+C+D) vào node2. Tiến hành lấy node1 AND node2, vẽ đồ thị mô phỏng.
- **Màn hình kết quả:**

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

Chú ý: Chương trình sẽ tiến hành tạo thêm các đường, ,ví dụ như nếu AS nối với CD mà CD lại nối với BD thì AS sẽ nối với BD. Điều này làm tăng tính đa dạng đường đi cho thuật toán tìm đường đi phía sau. Đồng thời khi CS nối BS và CD nối BD thì DD nối BD và DS nối CS.

2.2.2 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 đồ thị NMOS được vẽ ở trên có tới 3 đỉnh bậc 3 là CD, BD, BS. 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 vì khi đi qua tất cả các đỉnh thì đồng nghĩa với việc đã đi qua tất cả các CMOS.
- 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.
- 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, …
- **<u>Ví dụ:</u> Biểu thức**

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

Màn hình kết quả:

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

2.3 Tạo đồ thị mô hình hóa PMOS pull-up network và tìm đường đi Euler:

2.3.1 Tìm đường đi Euler cho 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.
- **<u>Ví dụ:</u>** 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 chẳn trong euler path NMOS.
- **Ví dụ:** Biểu thức $Y = \overline{A * (B + C + D)}$

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

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

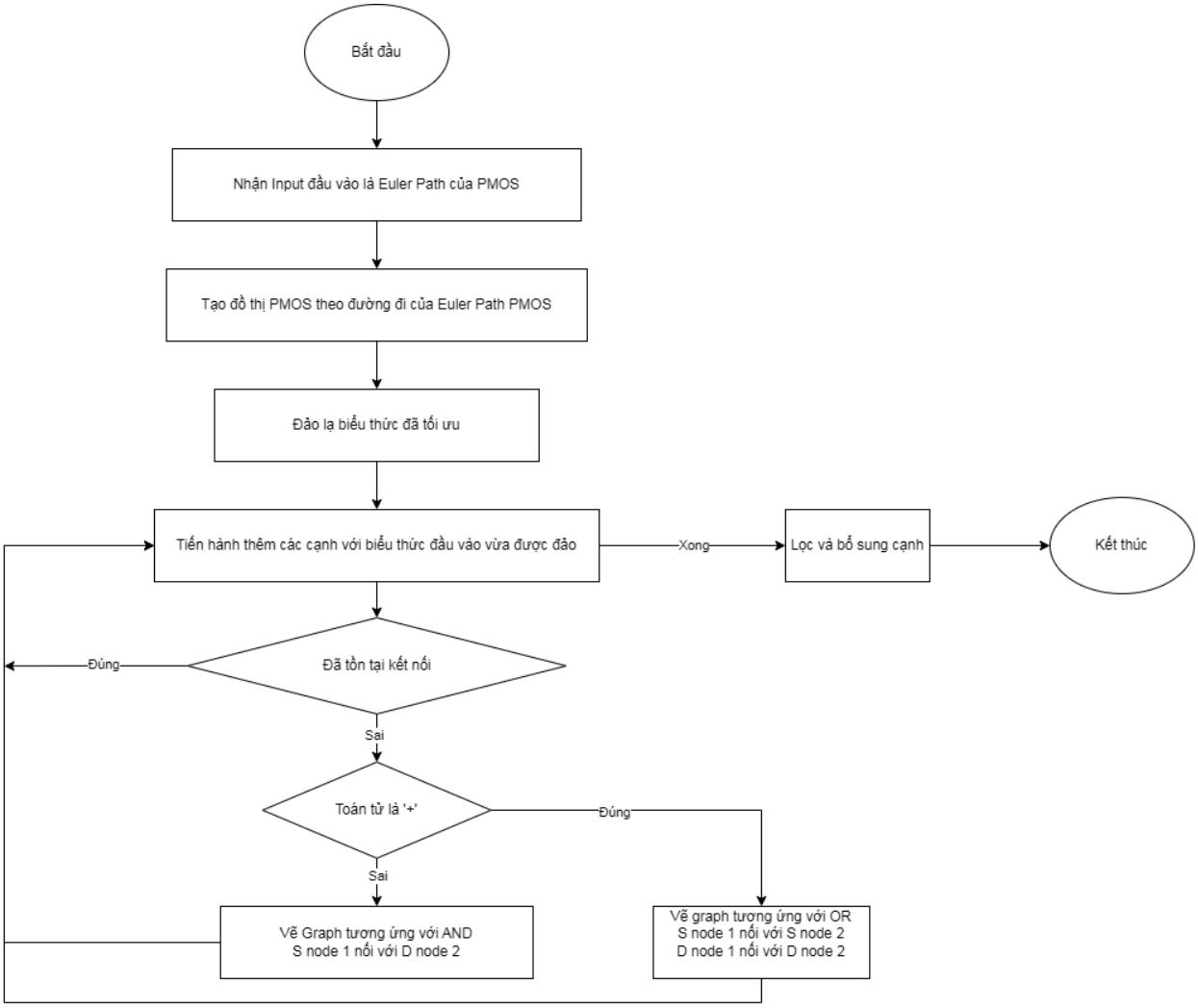
2.3.2 Tạo đồ thị mô hình hóa PMOS pull-up network:

- Để tạo đồ thị cho vùng PMOS pull-up thì ta cần dựa vào Euler path PMOS đã tìm được ở mục 2.3.1. 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.

  Ví dụ: Nếu A nối tiếp với B nhưng trong đồ thị từ Euler đã có AS nối BD thì không cần tạo thêm cạnh này nữa và đặc biệt sẽ không tạo cạnh AD nối BS, điểm này rất quan trọng vì nó có thể dẫn đến sai số trong việc kết nối đồ thị.
- 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 thuật toán:



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

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

2.4 Tìm điểm nối VDD, GND và 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.

Điểm có hậu tố "S" (*source*) sẽ được nối với VDD (PMOS) và GND (NMOS) nếu thõa 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.

Điểm có hậu tố "D" (*drain*) sẽ được nối với Output nếu thõa điều kiện sau:

- Chỉ kết nối với điểm S của chính nó
- Không có bất kỳ điểm S nào khác trong đồ thị nối với nó.

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

```
PMOS edges:  [('AD', 'AS'), ('AD', 'DD'), ('AS', 'CS'), ('DD', 'DS'), ('CS', 'CD'), ('CD', 'BS'), ('BS', 'BD'), ('BD', 'DS')]
NMOS edges:  [('BS', 'BD'), ('BS', 'CS'), ('BD', 'CD'), ('BD', 'DD'), ('BD', 'AS'), ('CS', 'CD'), ('CS', 'DS'), ('CD', 'AS'), ('DD', 'DS'), ('AS', 'AD')]
Node connect Source PMOS (VCC):
['AS', 'CS']
Node connect Source NMOS (GND):
['BS', 'CS', 'CS', 'DS']
Node connect Output PMOS (OUT):
['AD', 'DD']
Node connect Output NMOS (OUT):
['AD']
```

2.5 Tổng kết màn hình kết quả:

Với biểu thức đầu vào là $Y = \overline{A * B + A * C + A * D}$

Màn hình kết quả là:

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

3. Vẽ Stick Diagram với thư viện matplotlib trong ngôn ngữ Python:

Quá trình vẽ Stick Diagram sẽ gồm các bước sau:

3.1 Tạo danh sách các phần tử logic (input variables):

- Duyệt qua các node trong Euler path của NMOS. Lấy phần chữ cái của mỗi node (VD: từ "A_S" → "A"), tránh trùng lặp.
- Kết quả: Danh sách các biến logic (inputs) cần hiển thị trục đứng như A, C, B, D…

3.2 Vẽ các đường VDD, GND, Ndiff và Pdiff:

- Vẽ hai đường ngang **VDD** (trên) và **GND** (dưới) cách nhau theo chiều cao đã định
- Vẽ hai đường ngang khác cho **Ndiff** (mạng NMOS) và **Pdiff** (mạng PMOS) – nơi đặt transistor

3.3 Vẽ các đường tín hiệu vào (vertical input lines):

- Với mỗi biến đầu vào, vẽ một đường dọc từ trên (cắt VDD) đến dưới (qua GND)
- Đánh nhãn từng biến tại đầu đường dọc (phía trên VDD)

3.4 Vẽ các node PMOS và kết nối:

- Đặt tọa độ các node trong Euler Path PMOS: Duyệt qua euler_path_pmos, gán tọa độ cho mỗi node trong Euler Path theo thứ tự từ trái qua phải, phía trên Pdiff và ở hai bên cạnh đường dọc của từng đỉnh. Ví dụ: AD và AS sẽ ở hai bên của đường dọc đỉnh A.
- Vẽ kết nối PMOS:
    - Duyệt qua các cạnh trong g_pmos.edges.
    - Lấy tọa độ (x1, y1) và (x2, y2) của hai node được nối bởi cạnh.
    - Nếu khoảng cách giữa x1 và x2 lớn hơn 1 (abs(x2-x1) > 1):
    - Kiểm tra để tránh vẽ trùng lặp hoặc các kết nối không cần thiết. Ví dụ: AS nối CD, CD nối BD, yêu cầu AS phải được nối với BD nhưng vì đã BD đã nối với CD nên không cần vẽ thêm.
    - Vẽ kết nối bằng ba đoạn:
        - Đoạn thẳng đứng từ (x1, y1) xuống (x1, y1-count).
        - Đoạn thẳng đứng từ (x2, y2) xuống (x2, y2-count).
        - Đoạn ngang nối (x1, y1-count) và (x2, y2-count).
        - Vẽ dấu "x" tại (x1, y1) và (x2, y2) để biểu thị node.
        - Tăng count để các kết nối không chồng lấn.

3.5 Vẽ các node NMOS và kết nối:

Thuật toán tương tự như bên PMOS:

- Đặt tọa độ các node trong Euler Path NMOS. Chú ý vẽ ở phía dưới Ndiff.

- Vẽ kết nối NMOS

3.6 Vẽ kết nối đến VDD, GND và Output:

Đối với vẽ đường kết nối với VDD trong PMOS:

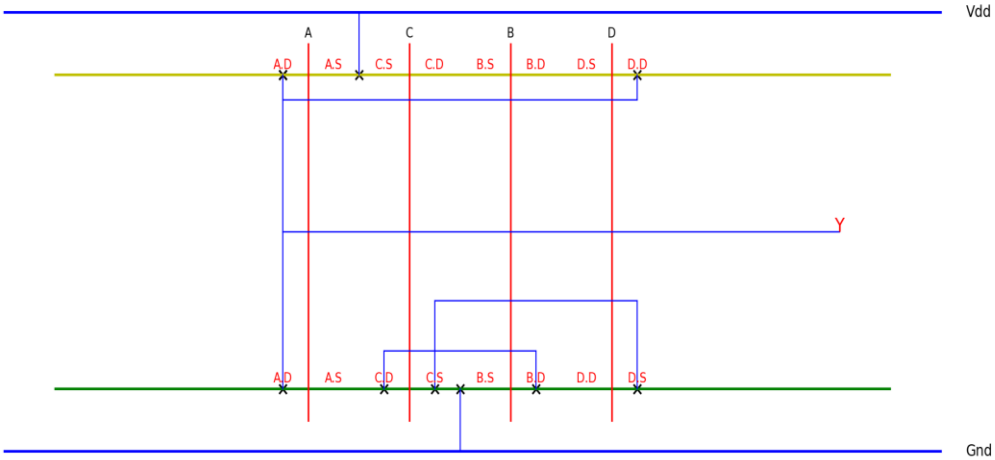
Ta sẽ lần lượt kiểm tra từng cặp node trong danh sách các node sẽ nối với VDD vừa tìm được ở phần 2.4

Giả sử gọi (x1, x2) lần lượt là các node khác nhau trong danh sách:

- Nếu x1 liền kề x2, vẽ đường thẳng từ điểm giữa của hai node đến VDD
- Nếu x1 không kề x2:
  - Kiểm tra xem cả hai x1, x2 đã được nối với VDD hay chưa:
    - Nếu cả hai đã được kết nối với VDD rồi thì bỏ qua
    - Nếu chưa thì tiếp tục kiểm tra xem một trong hai đã được nối với VDD chưa:
      - Nếu cả hai đều chưa thì nối cả hai lên VDD riêng biệt
      - Nếu có một node kết nối với VDD rồi thì kiểm tra thêm nếu node còn lại đã nối với một đỉnh có nối với VDD thì bỏ qua, ngược lại thì tự nối cạnh từ node đó tới VDD.
- Ngoại lệ nếu danh sách chỉ có 1 node duy nhất thì nối thẳng node đó đến VDD.

Đối với vẽ đường kết nối GND và Output cũng tương tự như trên.

3.7 Màn hình kết quả:



IV. 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 từ 3 đến 4 biến.
- 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 biểu thức sau khi tối ưu vẫn còn biến trùng.
- Việc vẽ các đường như VDD, GND, ndiff và pdiff chỉ là tương đối (phù hợp cho biểu thức 3 – 4 biến, còn 5 biến trở lên dễ xảy ra sai sót).