DẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

BÁO CÁO BÀI TẬP LỚN

Bộ môn: Thiết kế luận lý

SV:Trần Ngọc Cát————————1912750
SV:Diệp Trần Nam————————1914213
SV:Nguyễn Đức Phúc————————1914691

Ngày 18 tháng 6 năm 2020

Mục lục

Chương 1

DẪN NHẬP

Giới thiệu: Hệ thống phát hiện chuỗi được sử dụng trong nhiều lĩnh vực như y học, kinh tế, tài chính, chứng khoán, quản lý mạng truyền thông,v.v. Hệ thống này có chức năng giúp phát hiện các chuỗi kí tự xuất hiện trong một tập dữ liệu theo các quy luật nhất định.

Trong đề tài trên, ta cần phân chia các vấn đề như sau:

- **Một thanh ghi để lưu trữ chuỗi**

  Sử dụng 1 thanh ghi có độ dài 80 bits là *stringo* để lưu trữ chuỗi xem xét

- **Có khả năng nhập từng chuỗi 4 bit vào chuỗi lớn**

  Sử dụng 1 biến *counter* để xác định vị trí cần ghi vào chuỗi lớn, kèm thêm 1 tín hiệu *submit* để xác nhận chuỗi 4 bits sẽ được nhập vào. Với mỗi lần *submit* (nhập vào), *counter* sẽ tăng lên 4 và chờ để nhập chuỗi 4 bits tiếp theo.

- **Có khả năng xóa 4 bits từ chuỗi lớn khi quá trình nhập vào bị sai và cho phép nhập lại**

  Thay đổi 4 bits vừa nhập thành **4'b0000** và đồng thời thay đổi *counter* lùi xuống 4 vị trí

- **Có khả năng xem xét nếu chuỗi nhập vào giống với chuỗi so sánh thì in vị trí ra đèn led**

  Sử dụng biến *counter* để xác định vị trí của chuỗi trùng và đưa ra đèn led

- **Có thể dừng nhập chuỗi bất kì lúc nào và tiến hành xử lý**

  Sử dụng tín hiệu *done* để ra hiệu cho hệ thống rằng quá trình nhập chuỗi đã kết thúc và tiến hành xử lý.

- **Tiến trình xử lý phải quét hết chuỗi để tìm ra chuỗi 4 bits giống với chuỗi so sánh**

  Dựa vào *counter* để xác định số kí tự cần quét trong chuỗi lớn

- **Có khả năng đếm số lượng chuỗi trùng nhau**

  Trong quá trình quét chuỗi, nếu phát hiện trùng nhau, sẽ có 1 thanh ghi tăng giá trị lên 1 và sau đó đưa ra đèn led

- **Có khả năng lưu vị trí của các chuỗi trùng nhau và đưa ra led lần lượt**

  Sử dụng memory (bộ nhớ) để lưu các vị trí trùng nhau.

- **Xử lý vấn đề nhấp nháy led và thay đổi giá trị mỗi 2 giây**

  Sử dụng bộ chia tần số với chu kì 2 giây, lợi dụng trạng thái 1 và 0 của clock đã bị chia để điều khiển sự tắt hay sáng đèn.

- **Có khả năng nhập lại từ đầu**

  Sử dụng tín hiệu *reset*

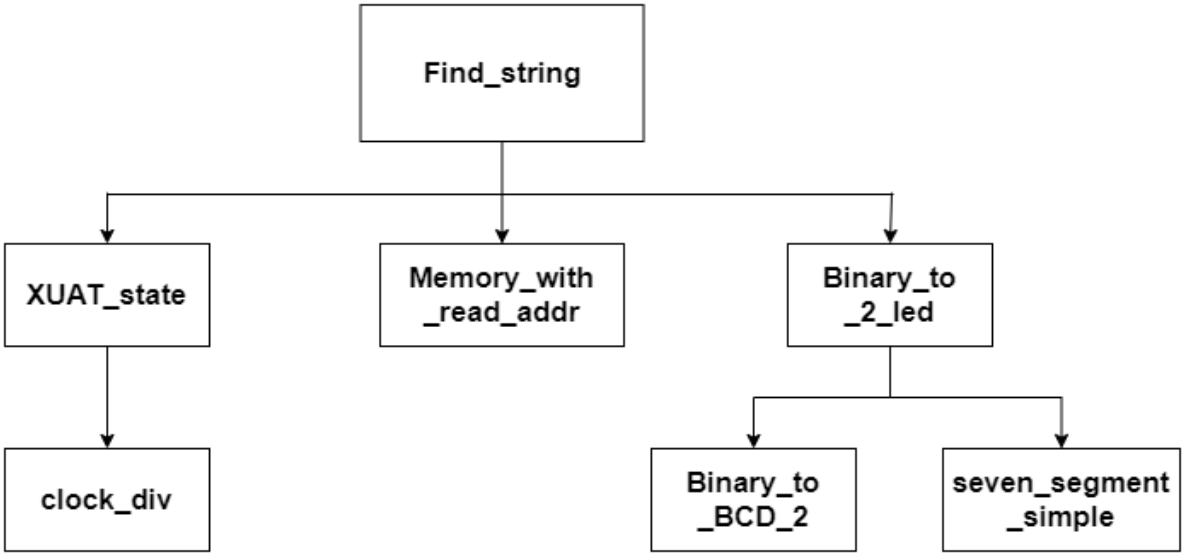
- **Có khả năng thay đổi chuỗi so sánh và tiến hành xử lý (quét lại toàn bộ chuỗi mà không cần phải nhập lại)**

  Sử dụng tín hiệu *roll_back* để thực hiện lại việc xử lý quét chuỗi từ đầu với chuỗi so sánh mới.

Chương 2

THIẾT KẾ

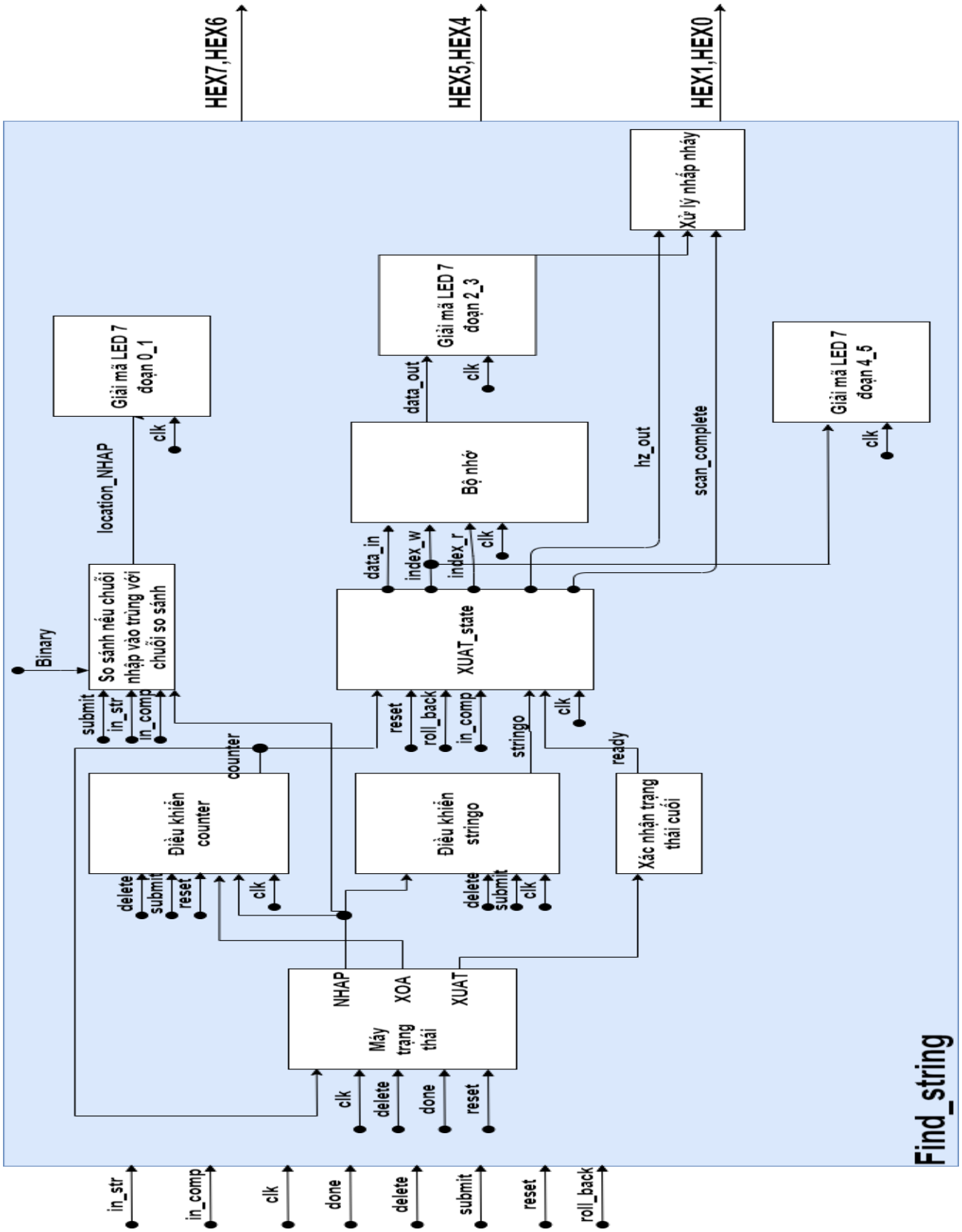
2.1 Cây phân cấp thiết kế



Hình 2.1: Cây phân cấp thiết kế

Khối chính là Find_string, chứa 5 khối là Xuat_state, Memory_with_read_addr và 3 khối Binary_to_2_led. Trong khối Xuat_state có sử dụng bộ chia tần số clock_div, và mỗi khối Binary_to_2_led đều sử dụng Binary_to_BCD_2 và sevent_segment_simple.

2.2 Sơ đồ khối

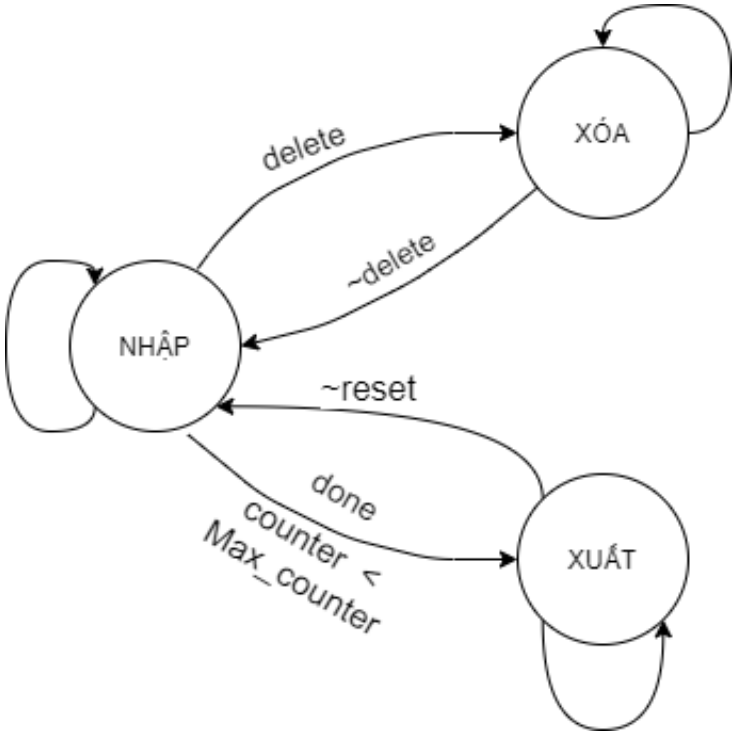


Hình 2.2: Sơ đồ khối

2.3 Chức năng của từng khối

2.3.1 Khối trạng thái

Ta có mô hình của khối trạng thái như sau :



Hình 2.3: Khối trạng thái

Từ trạng thái NHAP, tín hiệu delete sẽ chuyển trạng thái sang XOA, còn khi có tín hiệu *done* hoặc khi đã nhập đủ (80 kí tự), trạng thái sẽ chuyển sang XUAT.

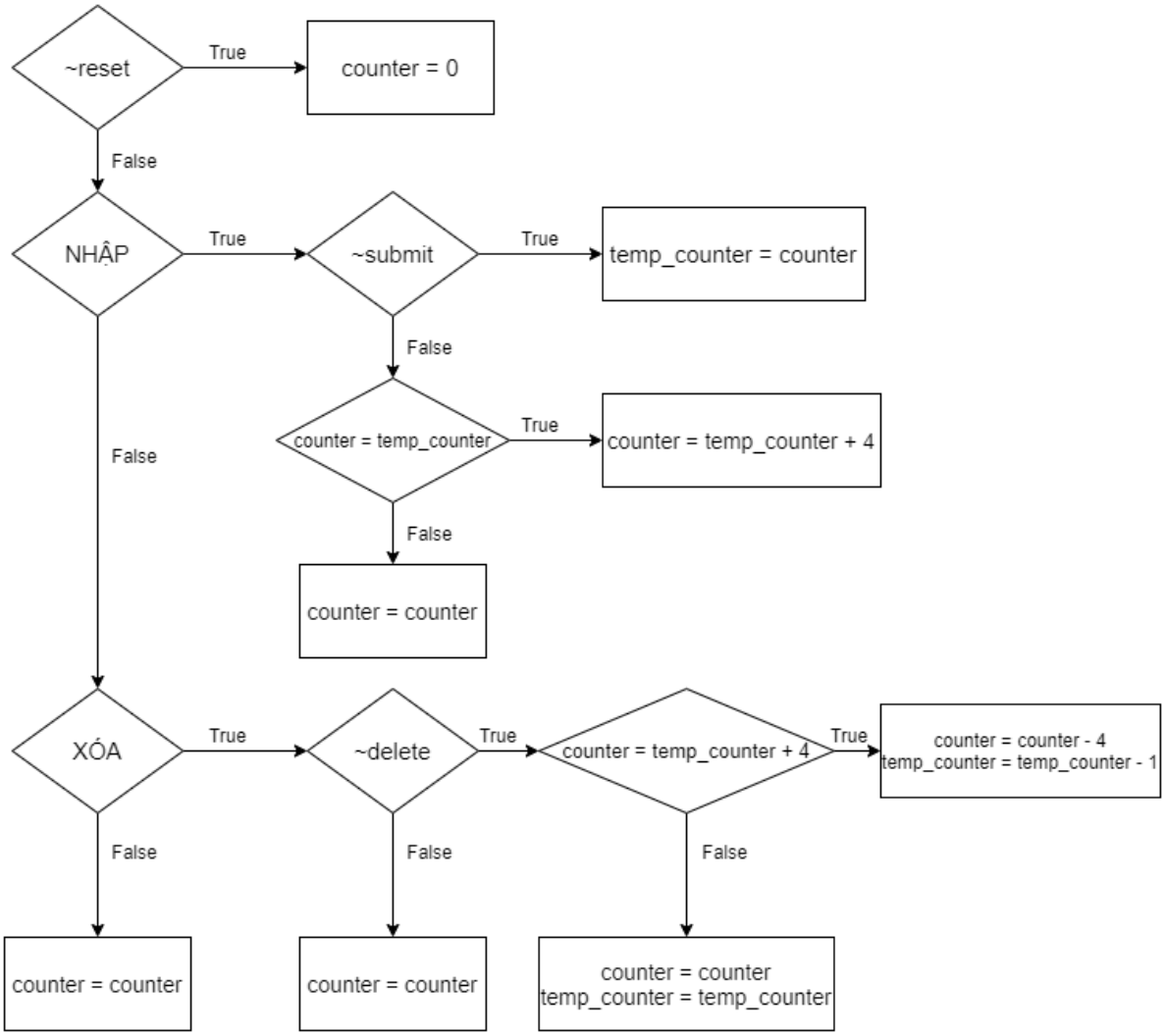
Ở trạng thái XOA, khi thả nút delete ra, trạng thái sẽ quay lại NHAP.

Và trạng thái XUAT sẽ giữ mãi cho đến khi có tín hiệu *reset* đưa trạng thái trở về NHAP.

2.3.2 Khối điều khiển *counter*

counter là 1 biến đếm để xác định vị trí kí tự được nhập vào chuỗi lớn, đồng thời cũng được dùng để xác định độ dài của chuỗi nhập vào khi chưa đạt tới giới hạn 80 kí tự.

Ta có mô hình của khối điều khiển *counter* như sau:



Hình 2.4: Khối điều khiển *counter* trong block always

Nút *reset* sẽ khởi động cho *counter* về 0. Theo đó, *counter* sẽ bị thay đổi tương ứng với 2 trạng thái: NHAP và XOA.

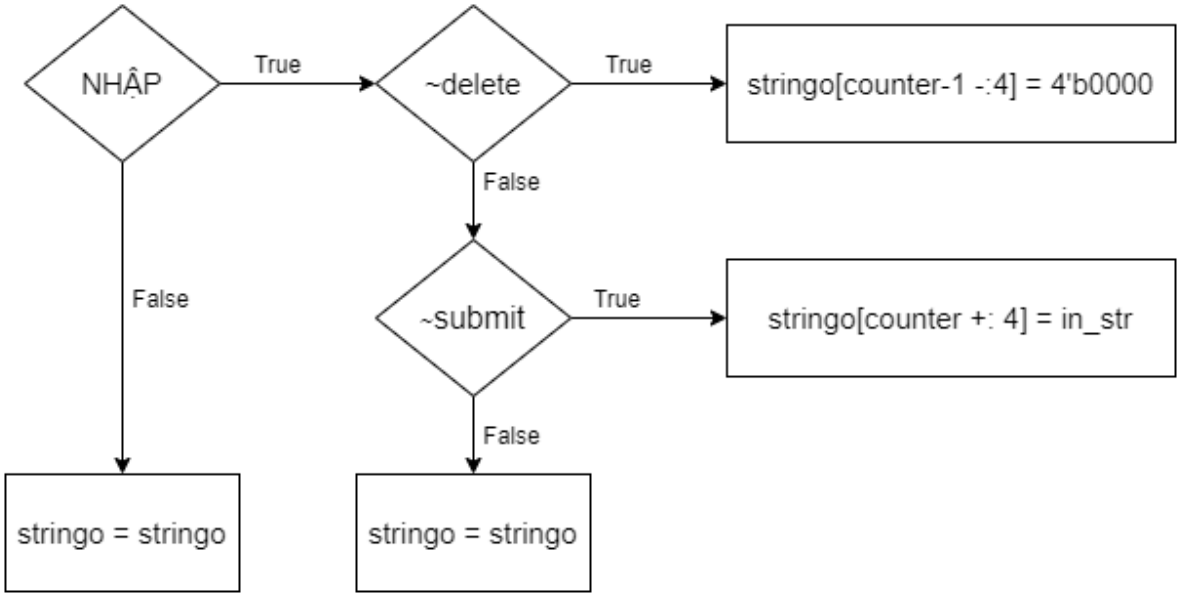
Ở trạng thái NHAP, *counter* chỉ thay đổi khi ta nhập chuỗi 4 bits vào chuỗi lớn. Để tránh việc tăng liên tục của *counter* khi nhấn nút submit, ta điều chỉnh để *counter* chỉ tăng SAU KHI đã thả nút submit ra, bằng cách lấy 1 temp_counter có độ rộng bằng *counter* để nhận giá trị của *counter* khi nhấn nút *submit*, và khi thả ra, khi này temp_*counter* == *counter*, *counter* sẽ tăng lên 3 chỉ 1 lần lúc thỏa điều kiện, sau đó giữ nguyên giá trị.

Trạng thái XOA được dẫn tới chỉ khi người dùng ở trạng thái NHAP và nhấn nút delete. Trong khi nút delete vẫn đang được nhấn và trạng thái đang là XOA, *counter* sẽ bị trừ đi 4 (lùi lại 4 vị trí) và temp_counter cũng bị lùi đi, nhưng chỉ 1 đơn vị. Việc trừ giá trị này chỉ xảy ra 1 lần, và chắc chắn sẽ xảy ra, do mỗi khi tiến hành *submit* dữ liệu ở trạng thái NHAP, *counter* sẽ luôn lớn hơn temp_counter 4 giá trị, thỏa mãn điều kiện (*counter* == temp_counter + 3'b100). Ở đây, temp_counter phải bị thay đổi để tránh khi *counter* trừ đi 4 thì sẽ bằng với temp_counter, lúc quay về trạng thái NHAP khiến điều kiện (*counter* == temp_counter) bị thỏa, dẫn đến *counter* bị tăng lên 4 trở lại và dữ liệu sẽ bị nhập vào KẾ TIẾP chuỗi kí tự bị xóa trước đó.

2.3.3 Khối điều khiển chuỗi lớn (stringo)

stringo là chuỗi lớn mà ta nhập vào, đồng thời cũng là đối tượng để quét, tìm và so sánh với chuỗi so sánh.

Ta có mô hình của khối điều khiển chuỗi stringo như sau:



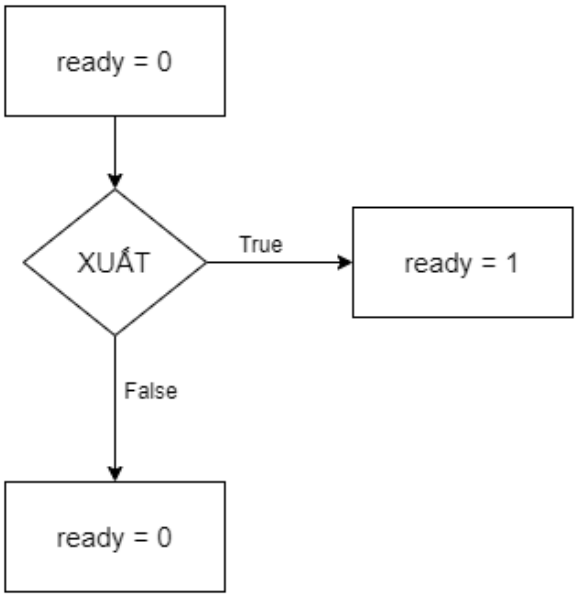
Hình 2.5: Khối điều khiển chuỗi stringo trong block always

stringo bị thay đổi khi trạng thái đang là NHAP. Nếu có tín hiệu delete, 4 kí tự kể từ vị trí *counter* lùi xuống sẽ bị đổi thành 4'b0000. Có tín hiệu *submit*, 4 kí tự kể từ vị trí *counter* sẽ nhận giá trị *in_comp* được nhập vào từ người dùng. Ở các trạng thái/ điều kiện khác, stringo giữ nguyên.

2.3.4 Khối xác nhận trạng thái cuối

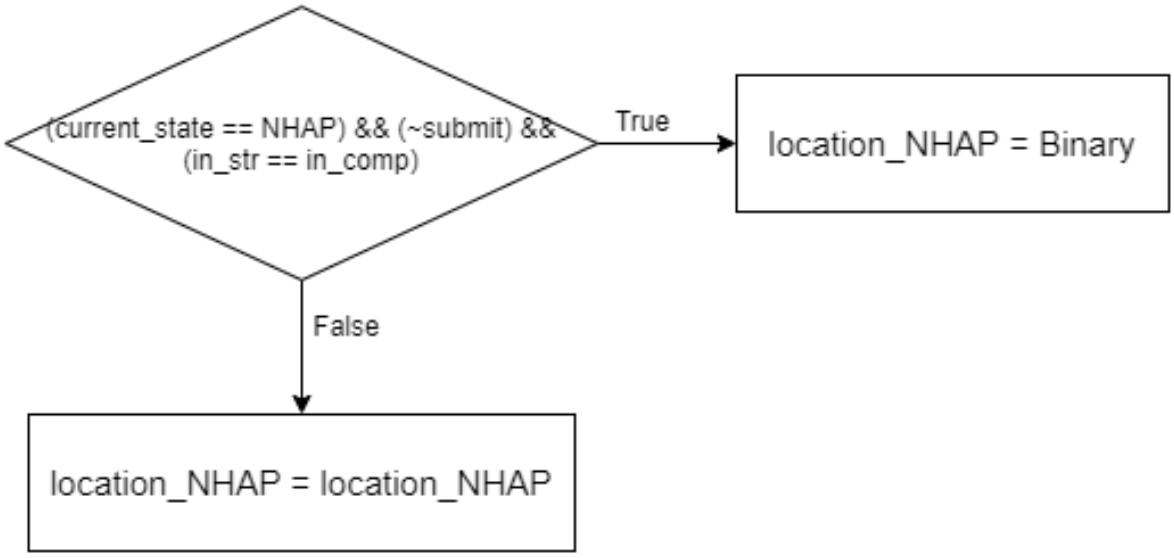
Đây là mạch tổ hợp để điều khiển tín hiệu *ready*, xác nhận "đã đến lúc xử lý dữ liệu". Chỉ khi vào trạng thái XUAT, *ready* mới tích cực (=1), ngược lại, *ready* luôn bằng 0.

Ta có mô hình của khối xác nhận trạng thái cuối như sau:



Hình 2.6: Khối Điều khiển trạng thái cuối

2.3.5 Khối so sánh nếu chuỗi nhập và chuỗi so sánh trùng nhau



Hình 2.7: Khối so sánh chuỗi nhập và chuỗi so sánh

Nói khối nghe có vẻ to lớn, nhưng đẩy chỉ là 1 câu lệnh assign tạo nên latch, xem xét nếu ở trạng thái NHAP, người dùng bấm nút *submit* và chuỗi nhập vào trùng với chuỗi so sánh, location_NHAP sẽ nhận lấy *counter* + 1, còn không thì giữ nguyên giá trị. location_NHAP này sẽ được đưa vào module Binary_to_2_led để đưa ra led biểu hiện ra ngoài, cho biết vị trí trùng nhau tại thời điểm nhập.

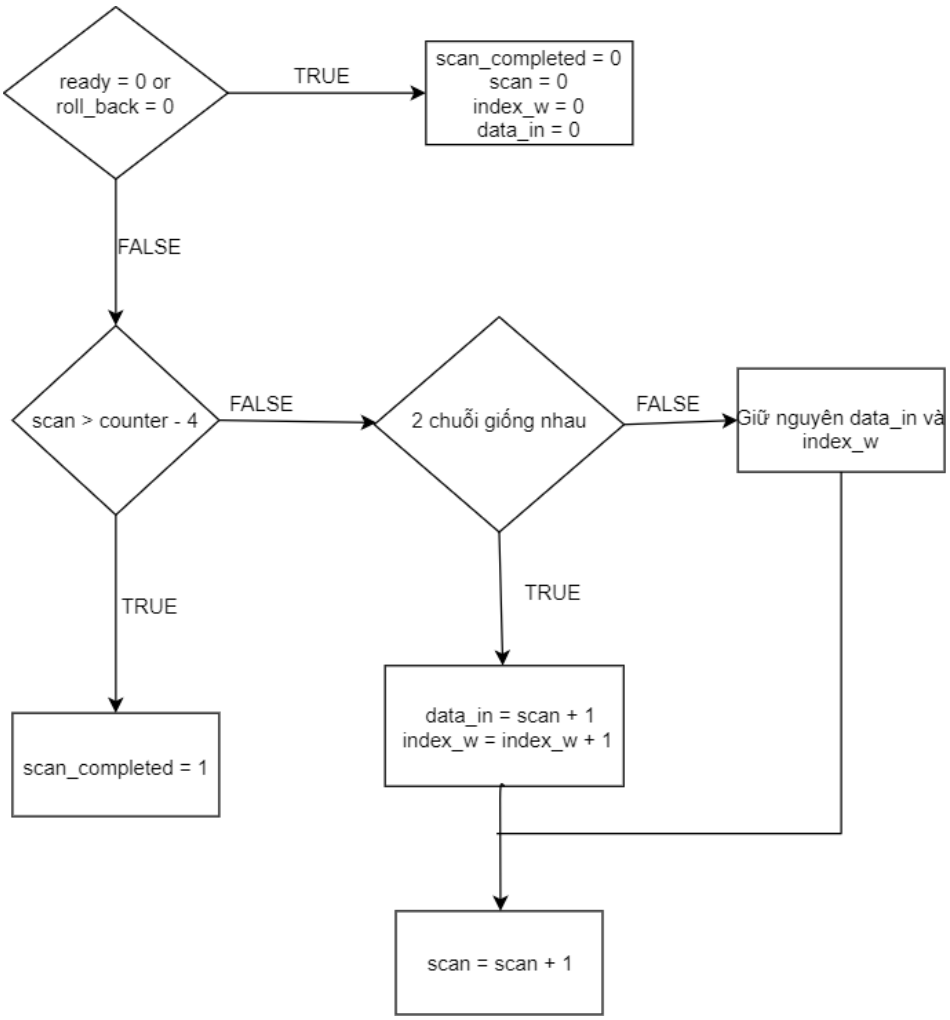
2.3.6 Khối xử lý (XUAT_state)

Đây là khối đảm nhiệm việc quét chuỗi để tìm ra các vị trí trùng nhau và số chuỗi trùng nhau.

Trong khối xử lý này, ta lại có 2 khối chức năng chính

- **Khối quét**

  Khi chưa bước vào trạng thái XUAT, tức *ready* = 0, hay khi người dùng bấm nút *roll_back*, các giá trị sẽ được khởi động mặc định.

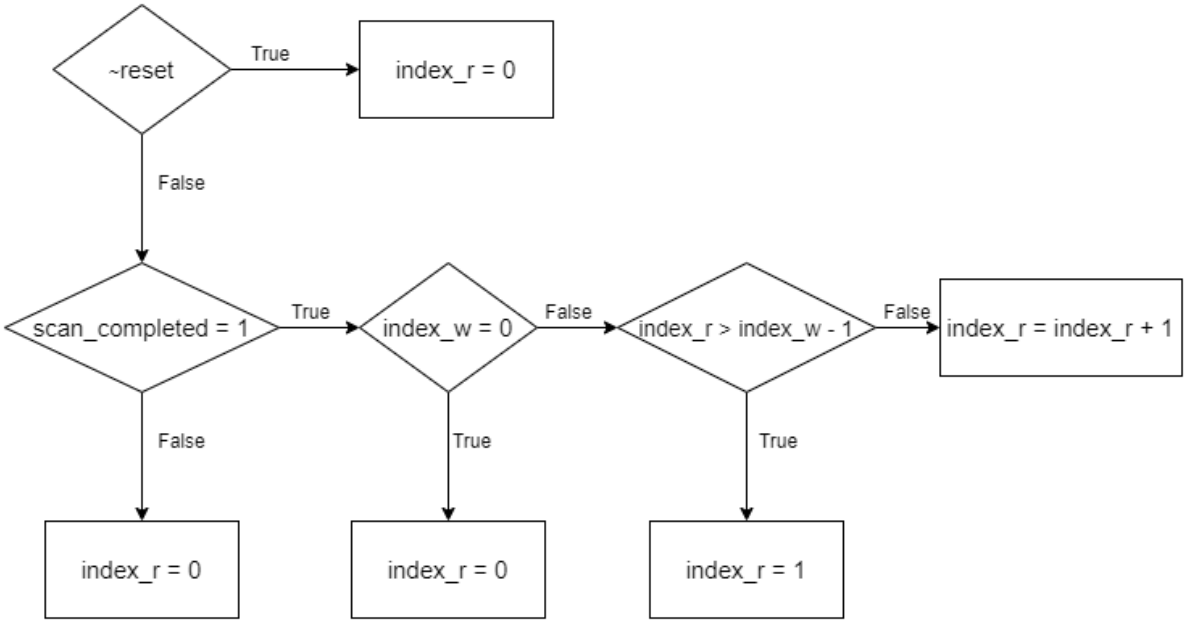


Hình 2.8: Khối quét được thực hiện trong khối always

Khi có tín hiệu xử lý (*ready* = 1 và không có *roll_back*), ta sẽ tiến hành "quét" chuỗi stringo từng chuỗi 4 bits một, bắt đầu từ đầu chuỗi stringo đến khi phần quét còn lại chỉ còn 4 kí tự. **Nếu** ta tìm được chuỗi 4 kí tự nào trùng với chuỗi so sánh (*in_comp*), *data_in* sẽ nhận giá trị scan tại đó + 1, đồng thời tăng *index_w* lên 1 đơn vị. *data_in* chính là vị trí trong chuỗi mà ta tìm thấy nó trùng với chuỗi so sánh, và *index_w* được nối với write_addr của memory, lựa chọn vị trí của bộ nhớ lưu trữ vị trí trùng nhau mà ta vừa tìm được. So sánh xong, scan sẽ tăng lên 1 cho đến khi đã quét đủ, lúc này khối sẽ đưa ra tín hiệu *scan_complete* báo hiệu đã quét hoàn tất.

Khi ta thay đổi chuỗi so sánh và nhấn *roll_back*, khối này sẽ thực hiện lại việc quét chuỗi từ đầu và so sánh với *in_comp* mới mà ta không cần phải nhập lại chuỗi stringo.

- **Khối lựa chọn *index_r* và *data_in* để đưa vào bộ nhớ**



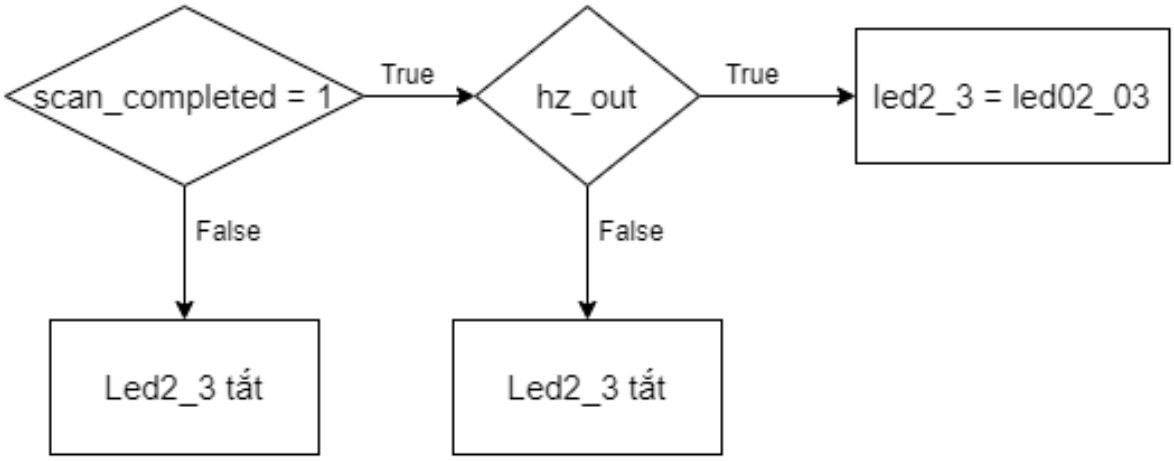
Hình 2.9: Khối điều khiển index_r và data_in trong khối always

Khối này sẽ xử lý vấn đề: cứ 2 giây thì sẽ thay đổi giá trị ở led 1 lần. Vì các vị trí trùng nhau của stringo và chuỗi so sánh được lưu trữ trong bộ nhớ, nên khối này phải sẽ tạo ra các địa chỉ đọc (*index_r*) tăng dần và vòng ngược lại, dựa trên có bao nhiêu *index_w* đã được viết vào bộ nhớ. Công việc trên được thực hiện chỉ khi khối quét đã hoàn thành (tín hiệu *scan_completed* = 1) Đầu tiên, cần phải khởi trị *index_r* bằng tín hiệu *reset* bất đồng bộ. Khối này sẽ nhận 1 tín hiệu hz_out được tạo ra từ bộ chia tần số có chu kì 2 giây, và xét cứ mỗi lần có kích cạnh xuống của hz_out thì sẽ tăng giá trị *index_r* lên 1 tới *index_w* rồi vòng ngược lại về 1. Ta sẽ chỉ đọc dữ liệu của memory từ vị trí số 1 do trong khối quét, dữ liệu cũng chỉ được đọc vào memory bắt đầu từ vị trí số 1. Coi vị trí thứ 0 là rỗng, tức lúc này *index_w* = 0, không tìm được chuỗi trùng, thì *index_r* cũng = 0.

2.3.7 Bộ nhớ

Bộ nhớ sâu 256 hàng, mỗi hàng rộng 8 bits, nhận các tín hiệu *data_in*, *index_r*, *index_w* từ khối Xử lý để ghi và đọc dữ liệu - vị trí trùng nhau của stringo và *in_comp*. *in_comp* của bộ nhớ được nối với khối Binary_to_2_led thứ 2, lần lượt xuất ra đèn led các vị trí trùng nhau mà bộ nhớ đang lưu trữ.

2.3.8 Nhấp nháy đèn



Hình 2.10: Điều khiển sự nhấp nháy của đèn trong khối always

Tuy nhiên, ta cũng cần phải khiến các đèn nhấp nháy. Vì thế, lợi dụng tín hiệu hz_out được tạo ra từ khối Xử lý, với chu kì 2 giây, 1 giây hz_out = 1 thì đèn led sẽ sáng, ngược lại, 1 giây hz_out = 0 thì đèn led sẽ tắt.

2.3.9 Tổng các vị trí trùng nhau

index_w sẽ đảm nhiệm công việc này. Trong khi lựa chọn vị trí để ghi vào memory, *index_w* cũng đồng thời cho ta biết được số lượng các vị trí trùng nhau mà khối Xử lí tìm được. Chính vì thế, khối Binary_to_2_led thứ 3 sẽ nhận lấy *index_w* và xuất ra đèn led ngoài, cho biết tổng các chuỗi trùng nhau tìm được.

Chương 3

HIỆN THỰC

Các port được sử dụng trong bài:

- **input** **reset** Reset hệ thống về trạng thái ban đầu
- **input** **delete** Xóa chuỗi vừa nhập.
- **input** **submit** Xác nhận nhập chuỗi
- **input** **roll_back** Cho phép thay đổi chuỗi so sánh mới và thực hiện quét lại chuỗi dữ liệu để tìm ra các vị trí trùng nhau mới
- **input** **clk** Xung clock sử dụng cho các khối mạch tuần tự
- **input** **done** Tín hiệu cho biết chuỗi đã được nhập xong khi chưa nhập đủ số lượng kí tự tối đa
- **input** **in_str** Input chuỗi cần nhập vào
- **input** **in_comp** Input chuỗi cần so sánh
- **output** **led0_1** Đèn led hiển thị vị trí trùng nhau khi chuỗi nhập vào trùng với chuỗi so sánh
- **output** **led2_3** Đèn led hiển thị vị trí trùng nhau khi đã tiến hành xử lý xong, nhấp nháy và thay đổi giá trị mỗi 2 giây
- **output** **led4_5** Đèn led hiển thị tổng số chuỗi trùng nhau tìm thấy được trong quá trình xử lý.

Chương 4

KẾT LUẬN

4.1 Phân công công việc trong nhóm

- Trần Ngọc Cát: phân tích ý tưởng, viết code verilog.
- Diệp Trần Nam: phân tích ý tưởng, vẽ các sơ đồ khối.
- Nguyễn Đức Phúc: phân tích và thống nhất ý tưởng, viết báo cáo.

4.2 Ứng dụng

Đây là 1 ứng dụng quan trọng trong đời sống, bởi lẽ, trong thời đại thông tin tràn ngập và mọi công việc đang đều ứng dụng dữ liệu thông tin, việc tìm kiếm 1 dữ liệu, giá trị cụ thể sẽ đóng phần quan trọng trong việc giúp con người chọn lọc ra những thông tin cần thiết, phù hợp và chính xác, tiết kiệm thời gian và sức lực. Việc tìm kiếm 1 chuỗi cụ thể trong loạt thông tin hiện nay đang được sử dụng ở gần như khắp mọi nơi.

4.3 Hướng phát triển

- Cho phép tìm kiếm dữ liệu từ 1 vị trí cụ thể
- Cho phép tìm kiếm dữ liệu trong 1 phạm vi nhất định
- Tăng giới hạn dữ liệu có thể lưu trữ được
- Khả năng xóa nhiều cụm dữ liệu ngược về khi nhập sai
- Có thể nhập vào 1 lượng lớn dữ liệu cùng lúc