

ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG ĐIỆN - ĐIỆN TỬ



THIẾT KẾ VLSI

Thiết kế bộ giải mã Viterbi

NGUYỄN TIẾN THÀNH **VŨ HUY HOÀNG**
thanh.nt224154@sis.hust.edu.vn hoang.vh213936@sis.hust.edu.vn

ĐẶNG QUANG VŨ **VŨ TIẾN LONG**
vu.dq223830@sis.hust.edu.vn long.vt224046@sis.hust.edu.vn

Giảng viên hướng dẫn: TS. Nguyễn Vũ Thắng _____

Hà Nội, 11/2025

MỤC LỤC

DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT	i
DANH MỤC HÌNH VẼ	ii
DANH MỤC BẢNG BIỂU	iii
CHƯƠNG 1. CƠ SỞ LÝ THUYẾT	1
1.1 Tổng quan về mã tích chập	1
1.2 Thuật toán giải mã Viterbi	3
CHƯƠNG 2. YÊU CẦU KỸ THUẬT	8
2.1 Yêu cầu thông số kỹ thuật	8
2.2 Sơ đồ khối các module	8
2.2.1 Module viterbi_decoder	8
2.2.2 Module ctrl	10
2.2.3 Module ham_d	11
2.2.4 Module add_comp	12
2.2.5 Module mem	15
2.2.6 Module tbck_dec	17
TÀI LIỆU THAM KHẢO	20

DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT

Từ viết tắt	Tên tiếng Anh	Nghĩa tiếng Việt
FSM	Finite State Machine	Máy trạng thái hữu hạn
ML	Maximum Likelihood	Cực đại khả năng

DANH MỤC HÌNH VẼ

Hình 1.1	Mã tích chập với tốc độ mã $1/2$	1
Hình 1.2	FSM của mã tích chập $(2, 1, 2)$	2
Hình 1.3	Sơ đồ lưới của mã tích chập $(2, 1, 2)$	3
Hình 1.4	Sơ đồ lưới của mã tích chập dựa theo đầu vào m	4
Hình 1.5	Minh họa giản đồ Trellis tại thời điểm $t = 1$	4
Hình 1.6	Minh họa giản đồ Trellis tại thời điểm $t = 2$	5
Hình 1.7	Minh họa giản đồ Trellis tại thời điểm $t = 3$	5
Hình 1.8	Minh họa giản đồ Trellis tại thời điểm $t = 4$	6
Hình 1.9	Minh họa giản đồ Trellis tại thời điểm $t = 5$	6
Hình 1.10	Minh họa giản đồ Trellis tại thời điểm $t = 6$	6
Hình 1.11	Minh họa giản đồ Trellis tại thời điểm $t = 7$	7
Hình 1.12	Minh họa giản đồ Trellis tại thời điểm $t = 8$	7
Hình 1.13	Xác định đường có khoảng cách Hamming nhỏ nhất trên giản đồ Trellis	7
Hình 2.1	Sơ đồ khối của module <code>viterbi_decoder</code>	8
Hình 2.2	Sơ đồ nối dây bên trong của module <code>viterbi_decoder</code>	9
Hình 2.3	Sơ đồ khối của module <code>ctrl</code>	10
Hình 2.4	Sơ đồ trạng thái FSM của module <code>ctrl</code>	11
Hình 2.5	Sơ đồ khối của module <code>ham_d</code>	12
Hình 2.6	Lưu đồ thuật toán của module <code>ham_d</code>	13
Hình 2.7	Sơ đồ khối của module <code>add_comp</code>	14
Hình 2.8	Lưu đồ thuật toán của module <code>add_comp</code>	16
Hình 2.9	Sơ đồ khối của module <code>mem</code>	16
Hình 2.10	Lưu đồ thuật toán của module <code>mem</code>	18
Hình 2.11	Sơ đồ khối của module <code>tbck_dec</code>	18
Hình 2.12	Sơ đồ trạng thái FSM của module <code>tbck_dec</code>	19

DANH MỤC BẢNG BIỂU

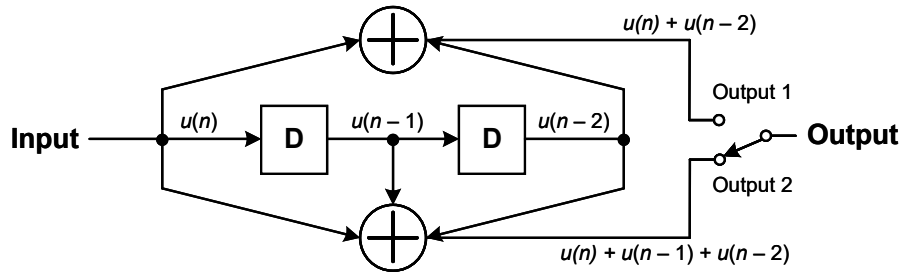
Bảng 1.1	Bảng trạng thái của mã tích chập (2, 1, 2)	2
Bảng 1.2	Bảng trạng thái của mã tích chập dựa theo đầu vào m	4
Bảng 2.1	Bảng mô tả các chân I/O của module <code>viterbi_decoder</code> . . .	9
Bảng 2.2	Bảng mô tả các chân I/O của module <code>ctrl</code>	11
Bảng 2.3	Bảng mô tả các chân I/O của module <code>ham_d</code>	13
Bảng 2.4	Bảng mô tả các chân I/O của module <code>add_comp</code>	15
Bảng 2.5	Bảng mô tả các chân I/O của module <code>mem</code>	17
Bảng 2.6	Bảng mô tả các chân I/O của module <code>tbck_dec</code>	19

CHƯƠNG 1. CƠ SỞ LÝ THUYẾT

1.1 Tổng quan về mã tích chập

Mã tích chập là một dạng mã tuyến tính, trong đó quá trình mã hóa được thực hiện tương tự như một bộ lọc số thực hiện phép tích chập giữa chuỗi dữ liệu đầu vào và các hệ số mã. Có thể coi bộ mã hóa tích chập như một tập hợp các hệ thống tuyến tính, bất biến theo thời gian, trong đó mỗi đầu ra là kết quả của quá trình lọc và tổ hợp tuyến tính các bit đầu vào hiện tại và quá khứ.

Đầu vào của bộ mã hóa tích chập là một dòng dữ liệu được biểu diễn dưới dạng vector. Mã tích chập có hai tham số quan trọng là tốc độ mã và chiều dài ràng buộc. Tốc độ mã của mã tích chập được ký hiệu $R = \frac{k}{n}$, trong đó k là số bit đầu vào và n là số bit đầu ra ở mỗi chu kỳ mã hóa. Một tham số quan trọng khác là chiều dài ràng buộc K biểu thị số lượng phần tử nhớ (hay số D-FF) trong thanh ghi dịch của bộ mã hóa, phản ánh mức độ phụ thuộc của đầu ra vào các bit đầu vào trước đó. Chẳng hạn với mã tích chập có tốc độ mã 1/2 được biểu diễn trên Hình 1.1



Hình 1.1 Mã tích chập với tốc độ mã 1/2

Ta xét một đầu vào cụ thể, chẳng hạn $m = \{1, 1, 0, 0, 1, 0, 1, 0\}$, khi đó ta sẽ biểu diễn được dưới dạng đa thức $m(x) = 1 + x + x^4 + x^6$. Với mã tích chập như trong Hình 1.1 thì ta có hai đa thức sinh $g^{(1)}(x) = 1 + x^2$ và $g^{(2)}(x) = 1 + x + x^2$. Khi đó đầu ra dạng đa thức là:

$$c^{(1)}(x) = m(x) \cdot g^{(1)}(x) = (1 + x + x^4 + x^6)(1 + x^2) = 1 + x + x^2 + x^3 + x^4 + x^8$$

$$c^{(2)}(x) = m(x) \cdot g^{(2)}(x) = (1 + x + x^4 + x^6)(1 + x + x^2) = 1 + x^3 + x^4 + x^5 + x^7 + x^8$$

Hay đầu ra còn được viết dưới dạng bit:

$$c^{(1)} = \{1, 1, 1, 1, 1, 0, 0, 1\}$$

$$c^{(2)} = \{1, 0, 0, 1, 1, 1, 0, 1\}$$

và dạng vector:

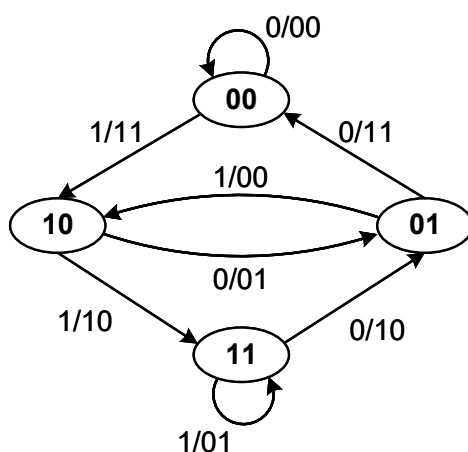
$$c = \{11, 10, 10, 11, 11, 01, 00, 01, 11\}$$

Mã tích chập thực chất là một máy trạng thái hữu hạn (FSM) do trạng thái của bộ mã hóa được xác định bởi giá trị trong các D Flip-Flops với số lượng trạng thái là 2^{K-1} . Hơn nữa, ở mỗi chu kỳ thời gian, một bit đầu vào vào thanh ghi, làm thay đổi trạng thái và sinh ra m bit đầu ra theo các đa thức sinh. Trong thiết kế bộ giải mã Viterbi cho mã tích chập (2, 1, 2), ta có Bảng 1.1 là bảng sự thật với S_1, S_2 là các trạng thái lần lượt cho $u(n-1), u(n-2)$ ban đầu và đầu vào Input, V_1, V_2 là đầu ra, S'_1, S'_2 là trạng thái kế tiếp.

Bảng 1.1 Bảng trạng thái của mã tích chập (2, 1, 2)

Input	S_1	S_2	V_1	V_2	S'_1	S'_2
0	0	0	0	0	0	0
0	0	1	1	1	0	0
0	1	0	1	0	0	1
0	1	1	0	1	0	1
1	0	0	1	1	1	0
1	0	1	1	0	1	0
1	1	0	0	1	1	1
1	1	1	1	0	1	1

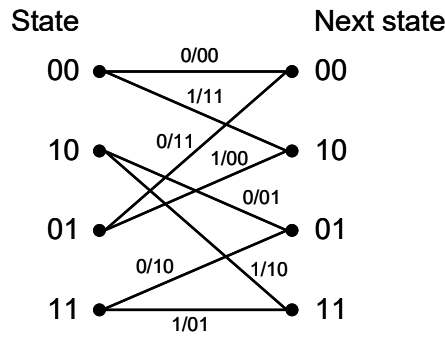
Khi đó ta có Hình 1.2 là FSM của mã tích chập (2, 1, 2).



Hình 1.2 FSM của mã tích chập (2, 1, 2)

Từ FSM ta có thể xây dựng sơ đồ lưới tương ứng như trên Hình 1.3.

Sơ đồ lưới này là tiền đề để xây dựng một giản đồ lưới lớn hơn phục vụ cho thuật toán Viterbi.



Hình 1.3 Sơ đồ lưới của mã tích chập (2, 1, 2)

1.2 Thuật toán giải mã Viterbi

Thuật toán Viterbi là một thuật toán giải mã cực đại khả năng (Maximum Likelihood - ML) được sử dụng phổ biến trong các bộ giải mã mã tích chập (convolutional decoder). Thuật toán này biểu diễn quá trình truyền mã bằng một đồ thị trạng thái (trellis), trong đó mỗi đỉnh biểu diễn một trạng thái bộ nhớ của bộ mã hóa, còn các cạnh thể hiện các chuyển trạng thái ứng với bit đầu vào.

Ở mỗi bước thời gian, thuật toán tính độ đo đường đi (path metric) cho từng trạng thái - là tổng của các độ đo nhánh (branch metric) đã đi qua. Branch metric thường được tính bằng khoảng cách Hamming giữa chuỗi đầu ra lý thuyết và chuỗi đầu ra nhận được (với kênh nhị phân), hoặc bằng khoảng cách Euclid trong các kênh khác. Với mỗi trạng thái, chỉ đường đi có metric nhỏ nhất được giữ lại (gọi là survivor path). Sau khi duyệt toàn bộ chuỗi, thuật toán truy ngược (traceback) để tìm đường đi tối ưu, tương ứng với chuỗi bit đầu vào có xác suất lớn nhất.

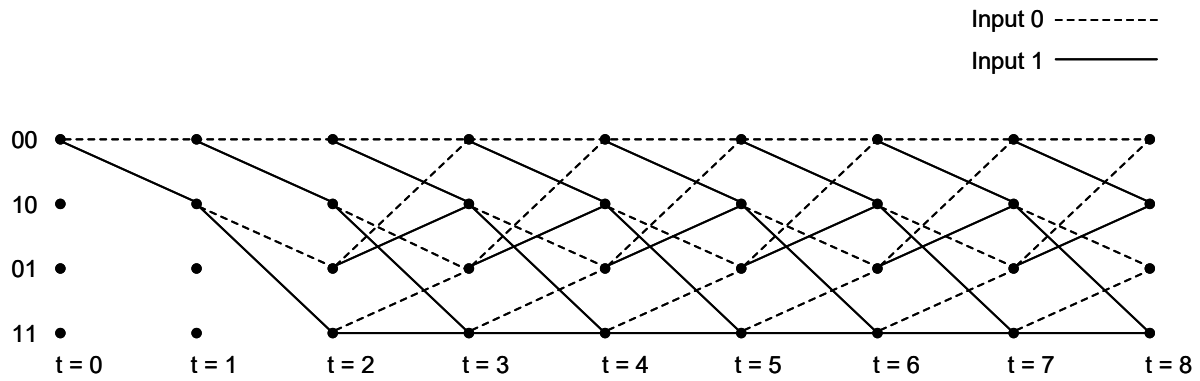
Ta xét ví dụ với đầu vào $m = \{1, 1, 0, 0, 1, 0, 1, 0\}$ với ma trận đa thức sinh là: $\mathbf{G}(x) = \begin{bmatrix} 1 + x^2 & 1 + x + x^2 \end{bmatrix}$. Khi đó từ mã thu được là $r = \{11, 10, 10, 11, 11, 01, 00, 01\}$.

Dựa theo sơ đồ FSM ở Hình 1.2, ta có bảng trạng thái tương ứng đầu vào như Bảng 1.2. Kết hợp với sơ đồ lưới như trên Hình 1.3, ta được một giản đồ lưới như trên Hình 1.4. Đây chính là giản đồ lưới mã hóa từ dãy bit đầu vào m . Thuật toán Viterbi sẽ giải mã ngược để tìm lại dãy bit ban đầu.

Xuất phát từ trạng thái 00, tại $t = 1$, khoảng cách Hamming giữa đầu ra (11) với nhánh 00-00 và 00-10 lần lượt là 2 và 0. Ta "đánh dấu" khoảng cách Hamming này trên từng nhánh như trên Hình 1.5.

Tiếp tục với trạng thái tại $t = 2$, tương tự như trạng thái trước, khoảng cách Hamming giữa đầu ra (10) với các nhánh 00-00, 00-10, 10-01 và 10-11 lần lượt là 3, 3, 2 và 0 (sở dĩ khoảng cách Hamming lớn hơn 2 là do ta cộng dồn metric của nhánh vào metric của nhánh trước đó). Ta "đánh dấu" khoảng cách Hamming này trên từng nhánh như

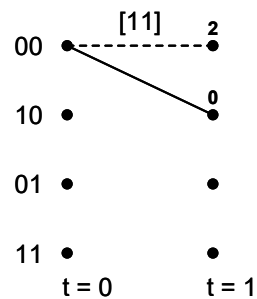
trên Hình 1.6.



Hình 1.4 Sơ đồ lưới của mã tích chập dựa theo đầu vào m

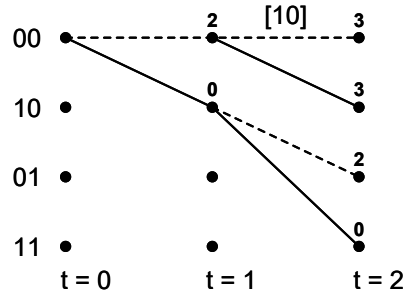
Bảng 1.2 Bảng trạng thái của mã tích chập dựa theo đầu vào m

t	Input	Output	State
0	1	11	10
1	1	10	11
2	0	10	01
3	0	11	00
4	1	11	10
5	0	01	01
6	1	00	10
7	0	01	01

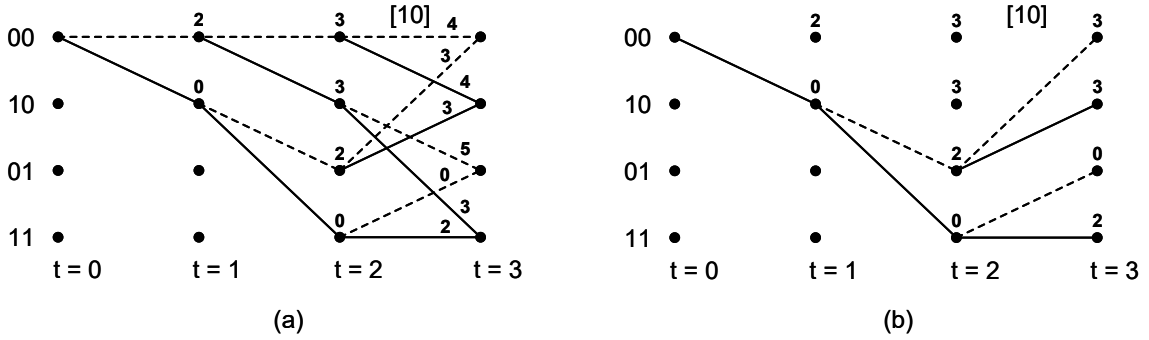


Hình 1.5 Minh họa giản đồ Trellis tại thời điểm $t = 1$

Tương tự trạng thái tại $t = 3$, khoảng cách Hamming giữa đầu ra (10) với các nhánh 00-00, 00-10, 10-01, 10-11, 01-00, 01-10, 11-01 và 11-11 lần lượt là 4, 4, 5, 3, 3, 4, 0 và 2. Ta "đánh dấu" khoảng cách Hamming này trên từng nhánh như trên Hình 1.7(a), tuy nhiên tại trạng thái này xuất hiện 2 tuyến đường cùng tới 1 trạng thái. Khi đó ta loại bỏ tuyến đường có khoảng cách Hamming lớn. Tuyến đường giữ lại được biểu diễn như Hình 1.7(b).



Hình 1.6 Minh họa giản đồ Trellis tại thời điểm $t = 2$



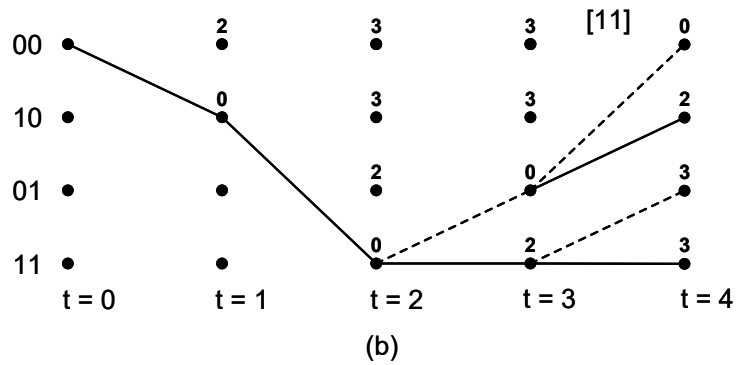
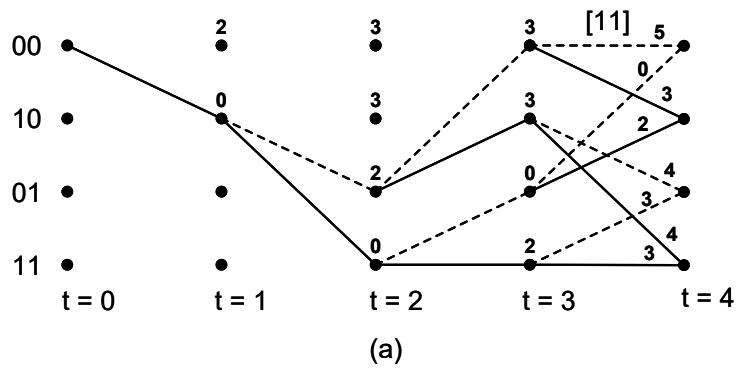
Hình 1.7 Minh họa giản đồ Trellis tại thời điểm $t = 3$

Tương tự trạng thái tại $t = 4$, khoảng cách Hamming giữa đầu ra (11) với các nhánh 00-00, 00-10, 10-01, 10-11, 01-00, 01-10, 11-01 và 11-11 lần lượt là 5, 3, 4, 4, 0, 2, 3 và 3. Ta "đánh dấu" khoảng cách Hamming này trên từng nhánh như trên Hình 1.8(a). Tuy nhiên tại trạng thái này, một nút xuất hiện 2 tuyến đường cùng đi tới. Khi đó ta loại bỏ tuyến đường có khoảng cách Hamming lớn. Tuyến đường giữ lại được biểu diễn như Hình 1.8(b).

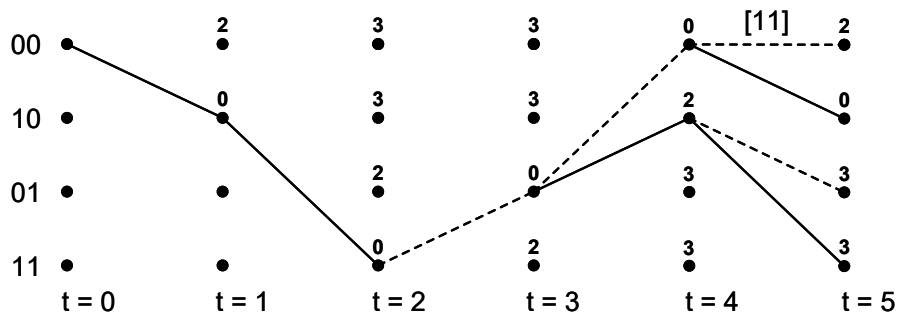
Lần lượt thực hiện tương tự đối với các trạng thái sau, đối với nút có 2 tuyến đi đến, ta chỉ lựa chọn tuyến có khoảng cách Hamming nhỏ hơn. Khi đó ta có giản đồ Trellis ở các thời điểm $t = 5$ (Hình 1.9), $t = 6$ (Hình 1.10), $t = 7$ (Hình 1.11) và $t = 8$ (Hình 1.12).

Sau thời điểm $t = 8$, giản đồ Trellis đã biểu diễn được hết khoảng cách Hamming của 8 cặp bit từ mã. Loại bỏ những nút có khoảng cách Hamming lớn, khi đó ta xác định được đường đi có khoảng cách Hamming nhỏ nhất được biểu diễn trên Hình 1.13. Với bit ban đầu là 1, đường đi sẽ biểu diễn bằng nét liền; với bit ban đầu là 0, đường đi sẽ biểu diễn bằng nét đứt, từ đó ta sẽ xác định được dãy 8 bit sau giải mã là $m_0 = \{1, 1, 0, 0, 1, 0, 1, 0\}$, trùng khớp với đầu vào bộ mã hóa ($m = \{1, 1, 0, 0, 1, 0, 1, 0\}$).

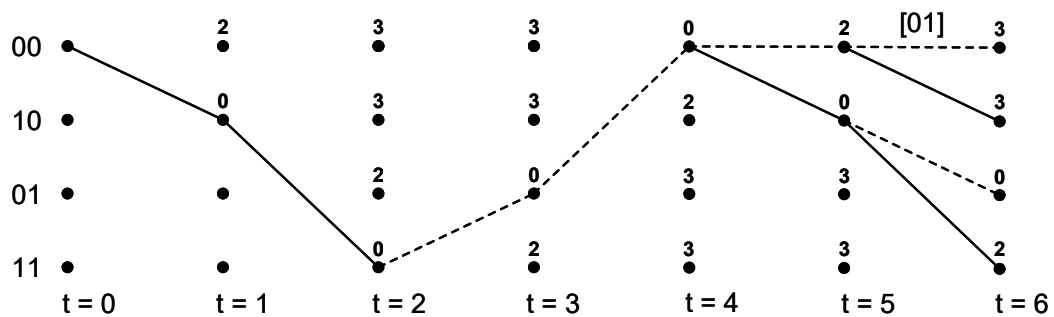
Thuật toán Viterbi đảm bảo chọn được đường có xác suất cao nhất trong trellis, do vậy tỷ lệ lỗi bit của bộ giải mã rất thấp. Vì vậy, Viterbi lý tưởng cho các hệ thống cần độ trễ thấp và tốc độ xử lý cao. Thuật toán Viterbi là thuật toán quy hoạch động, hoạt động bằng cách duy trì độ đo đường đi cho mỗi trạng thái trong đồ thị Trellis với độ phức tạp



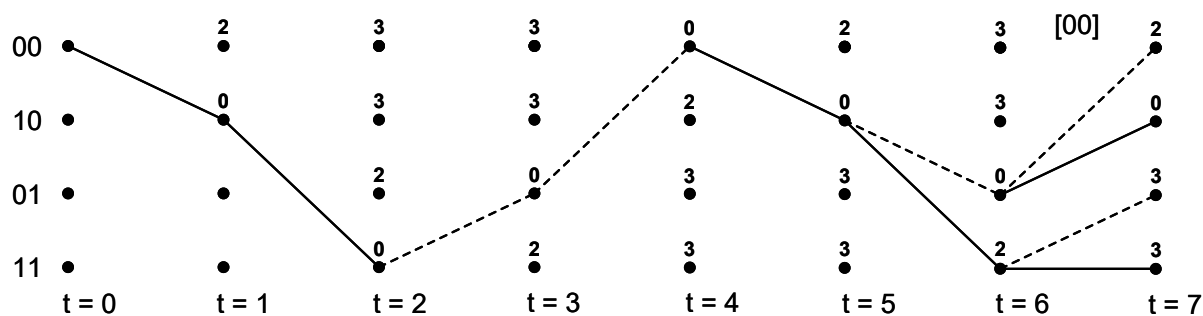
Hình 1.8 Minh họa giản đồ Trellis tại thời điểm $t = 4$



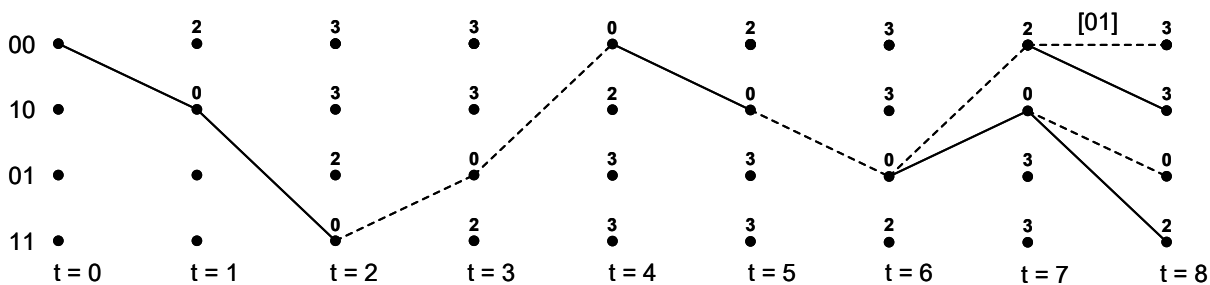
Hình 1.9 Minh họa giản đồ Trellis tại thời điểm $t = 5$



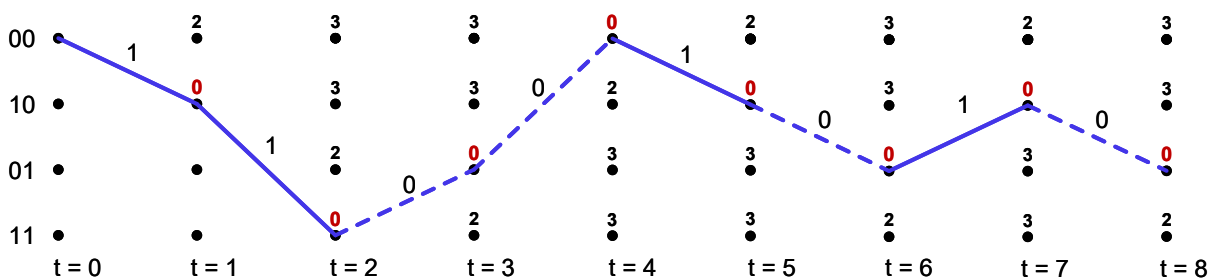
Hình 1.10 Minh họa giản đồ Trellis tại thời điểm $t = 6$



Hình 1.11 Minh họa giản đồ Trellis tại thời điểm $t = 7$



Hình 1.12 Minh họa giản đồ Trellis tại thời điểm $t = 8$



Hình 1.13 Xác định đường có khoảng cách Hamming nhỏ nhất trên giản đồ Trellis

$O(N.2^{K-1})$, do vậy độ phức tạp sẽ tăng nhanh nếu K tăng. Hơn nữa bộ giải mã Viterbi cần có một khối memory để lưu lại đường đi sống sót với dung lượng $N.2^{K-1}$ bit, đồng nghĩa với việc bộ nhớ tăng khi K tăng. Do vậy trong thiết kế mã tích chập và bộ giải mã Viterbi, K bị giới hạn từ 3 đến 7.

CHƯƠNG 2. YÊU CẦU KỸ THUẬT

2.1 Yêu cầu thông số kỹ thuật

Bộ giải mã Viterbi được thiết kế với các thông số cơ bản sau:

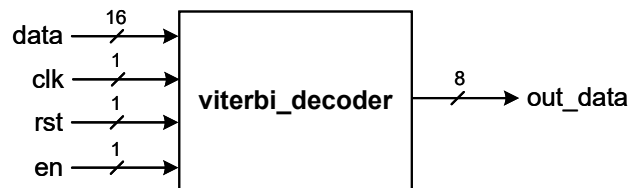
- Độ dài ràng buộc (Constrain Length): $K = 3$.
- Tốc độ mã hóa (Coding Rate): $R = \frac{1}{2}$.
- Đa thức sinh (Generator Polynomials): $g_1(x) = x^2 + x + 1$ và $g_2(x) = x^2 + 1$.
- Dữ liệu đầu vào: chuỗi 16 bit đã được mã hóa.
- Dữ liệu đầu ra: chuỗi 8 bit (thông tin ban đầu trước khi mã hóa).

2.2 Sơ đồ khối các module

2.2.1 Module *viterbi_decoder*

2.2.1.1 Sơ đồ khối

Sơ đồ khối của module *viterbi_decoder* được thể hiện trong Hình 2.1.



Hình 2.1 Sơ đồ khối của module *viterbi_decoder*

2.2.1.2 Chức năng

Module *viterbi_decoder* đóng vai trò là top module trong khối thiết kế, có chức năng khởi tạo 5 module nhỏ hơn (bao gồm các module *ctrl*, *ham_d*, *add_comp*, *mem* và *tbck_dec*) và thực hiện kết nối tín hiệu giữa các khối này với nhau. Cụ thể:

- Khối *ctrl*: khối điều khiển trung tâm.
- Khối *ham_d*: khối chia 16 bit đầu vào thành 8 bộ 2 bit một và tính toán khoảng cách Hamming của chúng.
- Khối *add_comp*: khối cộng và so sánh metric.
- Khối *mem*: bộ nhớ lưu trữ trạng thái.
- Khối *tbck_dec*: khối trace-back dùng để suy ra chuỗi đầu ra.

Module *viterbi_decoder* có nhiệm vụ giải mã chuỗi dữ liệu mã hóa 16 bit ở

đầu vào (data) thành chuỗi dữ liệu gốc 8 bit ở đầu ra (output) thông qua quá trình xử lý của các khối con bên trong.

- Nếu $rst = 1$, module sẽ thực hiện khởi tạo lại toàn bộ hệ thống.

- Nếu $en = 1$ và tại sườn dương của tín hiệu clk , mạch bắt đầu thực hiện quá trình giải mã theo thuật toán Viterbi.

- Ngược lại, nếu $en = 0$, toàn bộ mạch sẽ giữ nguyên trạng thái hiện tại và không thực hiện bất kỳ phép tính nào.

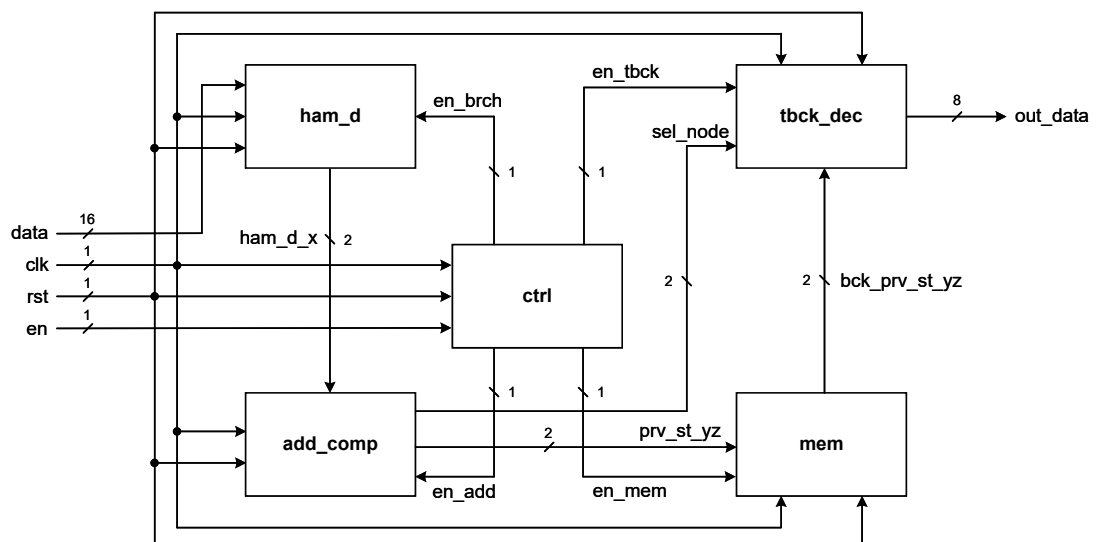
2.2.1.3 Đầu vào, đầu ra

Bảng mô tả chân I/O của module `viterbi_decoder` được thể hiện ở Bảng 2.1.

Bảng 2.1 Bảng mô tả các chân I/O của module `viterbi_decoder`

Tên chân	I/O	Số bit	Chức năng
clk	I	1	Tạo xung clock, sườn dương của xung là tín hiệu điều khiển lệnh trong mạch.
rst	I	1	Thiết lập module về trạng thái ban đầu.
en	I	1	Cho phép module hoạt động.
data	I	16	Dữ liệu đầu vào cần giải mã.
out_data	O	8	Dữ liệu đầu ra sau khi được giải mã.

2.2.1.4 Sơ đồ nối dây bên trong

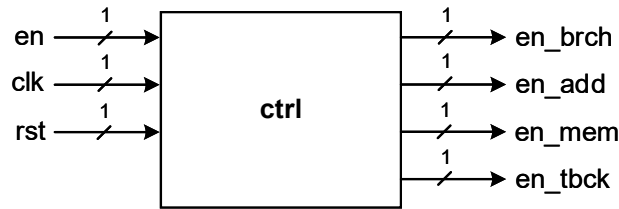


Hình 2.2 Sơ đồ nối dây bên trong của module `viterbi_decoder`

2.2.2 Module `ctrl`

2.2.2.1 Sơ đồ khối

Sơ đồ khối của module `ctrl` được thể hiện trong Hình 2.3.



Hình 2.3 Sơ đồ khối của module `ctrl`

2.2.2.2 Chức năng

Module `ctrl` có nhiệm vụ tạo ra các tín hiệu điều khiển enable để đồng bộ hoạt động giữa các khối trong mạch. Module hoạt động như một máy trạng thái hữu hạn (FSM). Khi tín hiệu `en` được kích hoạt, `ctrl` sẽ lần lượt cho phép các khối `ham_d`, `add_comp`, `mem` và `tbck_dec` hoạt động theo đúng thứ tự, cụ thể:

- Trạng thái S_0 : Khi `rst` = 1, FSM quay về trạng thái này, đặt biến `count` = 0. Các tín hiệu `en_brch`, `en_add`, `en_mem`, `en_tbck` đều bằng 0. Chờ tín hiệu `en` = 1 để chuyển sang trạng thái tiếp theo.

- Trạng thái S_1 : Khi `en` = 1, FSM chuyển sang trạng thái này, `en_brch` = 1, khối `ham_d` hoạt động, tách 8 cặp bit và tính khoảng cách Hamming cho từng cặp.

- Trạng thái S_2 : Sau khi khối `ham_d` tính toán xong khoảng cách Hamming của các cặp bit, trạng thái S_1 sẽ chuyển sang trạng thái này, khi đó `en_add` = 1, cho phép khối `add_comp` hoạt động.

- Trạng thái S_3 : Sau khi khối `add_comp` hoạt động xong, trạng thái S_2 sẽ chuyển sang trạng thái này, khi đó `en_mem` = 1, cho phép khối `add_mem` hoạt động. Ở trạng thái này, sau mỗi chu kỳ `clk`, biến `count` tăng thêm 1, khi `count` > 10, FSM sẽ chuyển sang trạng thái S_4 .

- Trạng thái S_4 : Khi `count` > 10, FSM chuyển sang trạng thái S_4 , khi đó `en_tbck` = 1, cho phép khối `tbck_dec` hoạt động. Khối này sẽ truy ngược lại `trellis` để lấy chuỗi bit đầu ra.

2.2.2.3 Đầu vào, đầu ra

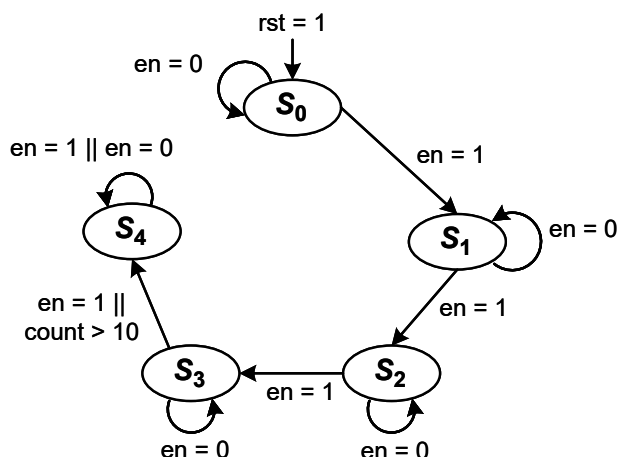
Bảng mô tả các chân I/O của module `ctrl` được thể hiện ở Bảng 2.2.

2.2.2.4 Sơ đồ trạng thái FSM

Bảng 2.2 Bảng mô tả các chân I/O của module **ctrl**

Tên chân	I/O	Số bit	Reg	Chức năng
clk	I	1		Tạo xung clock, sườn dương của xung là tín hiệu điều khiển lệnh trong mạch.
rst	I	1		Thiết lập module về trạng thái ban đầu.
en	I	1		Thiết lập module hoạt động.
en_brch	O	1	Reg	Tín hiệu điều khiển khối ham_d hoạt động.
en_add	O	1	Reg	Tín hiệu điều khiển khối add_comp hoạt động.
en_mem	O	1	Reg	Tín hiệu điều khiển khối mem hoạt động.
en_tbck	O	1	Reg	Tín hiệu điều khiển khối tbck_dec hoạt động.

Như đã nhắc đến trong Mục 2.2.2.2, module **ctrl** hoạt động như một FSM với 5 trạng thái. Mô hình FSM được biểu diễn trên Hình 2.4.



Hình 2.4 Sơ đồ trạng thái FSM của module **ctrl**

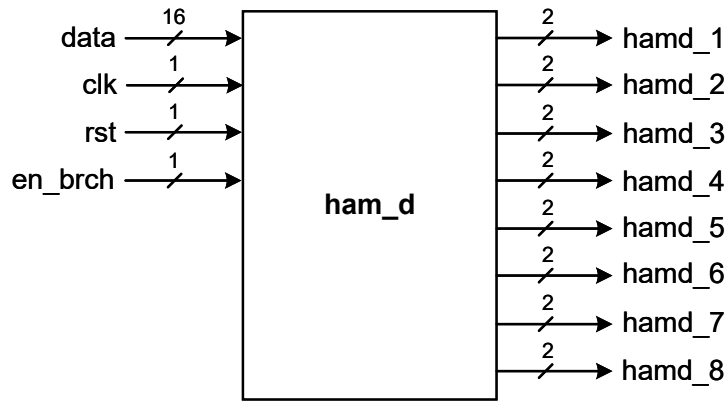
2.2.3 Module **ham_d**

2.2.3.1 Sơ đồ khối

Sơ đồ khối của module **ham_d** được thể hiện trong Hình 2.5.

2.2.3.2 Chức năng

Khối **ham_d** có nhiệm vụ trích xuất và xử lý từng cặp 2 bit liên tiếp từ một dữ liệu đầu vào 16 bit, đồng thời tính toán khoảng cách Hamming giữa các cặp bit này với các mẫu tham chiếu. Nó đảm bảo rằng tại mỗi chu kỳ xử lý, bộ giải mã Viterbi nhận được đúng cặp bit cần thiết cùng các giá trị khoảng cách Hamming tương ứng, phục vụ cho quá trình lựa chọn đường đi tối ưu trong trellis. Chức năng của khối được đặc tả như sau:



Hình 2.5 Sơ đồ khối của module `ham_d`

- Khi `rst = 1`, module được đưa về trạng thái khởi tạo, một bộ đếm nội `count` được đặt tại giá trị 15, tương ứng với bit cao nhất (MSB) trong chuỗi 16 bit đầu vào. Các cặp bit sau tách được gán bằng 00 và khoảng cách Hamming cũng được gán bằng 00.

- Khi `en_brch = 1`, module bắt đầu hoạt động:

+ Hai bit liên tiếp được lấy ra từ vị trí `count` và `count - 1` của dữ liệu đầu vào `data[15:0]`, bắt đầu từ MSB. Sau mỗi lần trích xuất, giá trị `count` giảm đi 2, tiến dần về LSB.

+ Cặp bit vừa được trích xuất được đưa vào mạch tính toán nội bộ để so sánh với bốn mẫu mã hóa tham chiếu (00, 01, 10, 11). Mỗi phép so sánh xác định số bit khác biệt (Hamming distance) giữa cặp bit và mẫu tham chiếu. Kết quả của tám phép tính (ứng với tám trạng thái trong trellis Viterbi) được xuất song song ra các đầu ra `hamd_1` đến `hamd_8` (mỗi đầu ra là giá trị 2 bit biểu diễn khoảng cách Hamming).

+ Khi `count` giảm tới 2, quá trình trích xuất sẽ dừng lại để tránh truy cập ngoài phạm vi dữ liệu.

2.2.3.3 Đầu vào, đầu ra

Bảng mô tả các chân I/O của module `ham_d` được mô tả trên Bảng 2.3.

2.2.3.4 Lưu đồ thuật toán

Lưu đồ thuật toán của module `ham_d` được biểu diễn trên Hình 2.6.

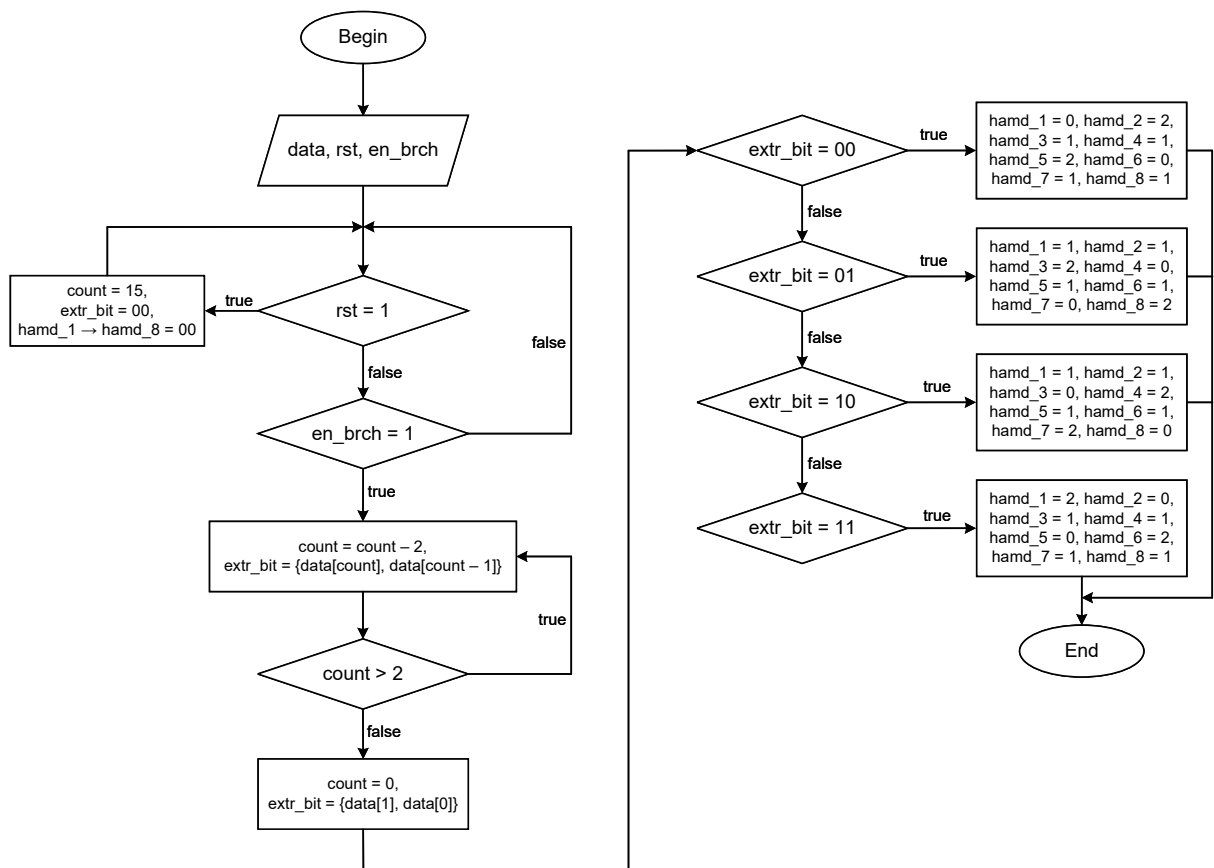
2.2.4 Module `add_comp`

2.2.4.1 Sơ đồ khối

Sơ đồ khối của module `add_comp` được thể hiện trong Hình 2.7.

Bảng 2.3 Bảng mô tả các chân I/O của module **ham_d**

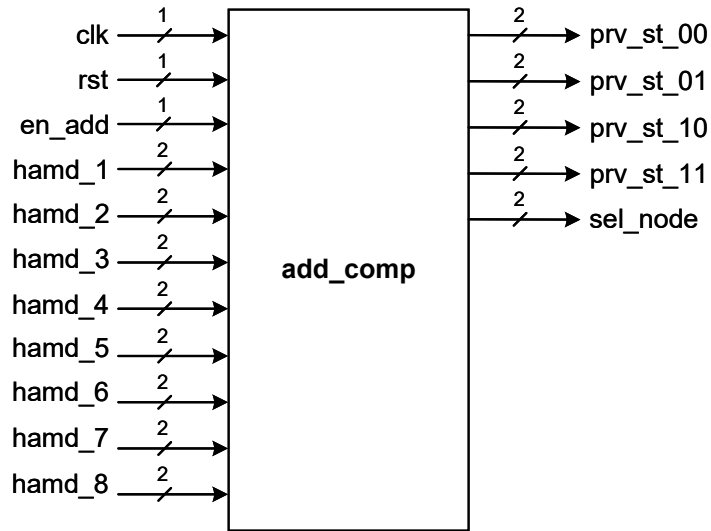
Tên chân	I/O	Số bit	Reg	Chức năng
clk	I	1		Tạo xung clock, sườn dương của xung là tín hiệu điều khiển lệnh trong mạch.
rst	I	1		Thiết lập module về trạng thái ban đầu.
en_brch	I	1		Thiết lập module hoạt động.
hamd_1, hamd_5	O	2	Reg	Khoảng cách hamming giữa đầu vào và đầu ra tại nút 00.
hamd_2, hamd_6	O	2	Reg	Khoảng cách hamming giữa đầu vào và đầu ra tại nút 01.
hamd_3, hamd_7	O	2	Reg	Khoảng cách hamming giữa đầu vào và đầu ra tại nút 10.
hamd_4, hamd_8	O	2	Reg	Khoảng cách hamming giữa đầu vào và đầu ra tại nút 11.



Hình 2.6 Lưu đồ thuật toán của module **ham_d**

2.2.4.2 Chức năng

Khối **add_comp** thực hiện các phép cộng, so sánh và chọn đường đi sống sót trong



Hình 2.7 Sơ đồ khối của module `add_comp`

thuật toán giải mã Viterbi. Mục tiêu của khối là cập nhật tổng tích lũy cho bốn trạng thái hiện tại của trellis: 00, 01, 10, 11, lựa chọn trạng thái trước đó tương ứng với mỗi trạng thái hiện tại dựa trên tổng tích lũy nhỏ hơn, đồng thời xác định nút có giá trị tích lũy nhỏ nhất toàn cục, để phục vụ cho việc truy vết ngược (traceback). Chức năng của khối được đặc tả như sau:

- Khi `rst = 1`, tất cả các thanh ghi tổng tích lũy và đầu ra (`sum_xx`, `prv_st_xx`) được đặt về 0. Khi `en_add = 1`, tại mỗi cạnh lên của `clk`, khối thực hiện phép cộng và so sánh cho từng trạng thái:

+ `sum_00_new = min(hamd_1 + sum_00_prev, hamd_5 + sum_01_prev).`

+ `sum_01_new = min(hamd_3 + sum_10_prev, hamd_7 + sum_11_prev).`

+ `sum_10_new = min(hamd_2 + sum_00_prev, hamd_6 + sum_01_prev).`

+ `sum_11_new = min(hamd_4 + sum_10_prev, hamd_8 + sum_11_prev).`

- Với mỗi phép so sánh, trạng thái trước tương ứng (`prv_st_xx`) được cập nhật theo nhánh có giá trị nhỏ hơn.

- Sau khi cập nhật bốn tổng tích lũy, khối thực hiện phép so sánh giữa `sum_00_new`, `sum_01_new`, `sum_10_new`, `sum_11_new` để tìm giá trị nhỏ nhất. `sel_node` được gán bằng mã trạng thái (2 bit) của node có tổng tích lũy nhỏ nhất. Nếu các tổng bằng nhau, thứ tự ưu tiên là: `00 > 10 > 01 > 11`.

2.2.4.3 Đầu vào, đầu ra

Bảng mô tả các chân I/O của module `add_comp` được mô tả trên Bảng 2.4.

Bảng 2.4 Bảng mô tả các chân I/O của module **add_comp**

Tên chân	I/O	Số bit	Reg	Chức năng
clk	I	1		Tạo xung clock, sườn dương của xung là tín hiệu điều khiển lệnh trong mạch.
rst	I	1		Thiết lập module về trạng thái ban đầu.
en_add	I	1		Thiết lập module hoạt động.
hamd_1, hamd_5	I	2		Khoảng cách hamming giữa đầu vào và đầu ra tại nút 00.
hamd_2, hamd_6	I	2		Khoảng cách hamming giữa đầu vào và đầu ra tại nút 01.
hamd_3, hamd_7	I	2		Khoảng cách hamming giữa đầu vào và đầu ra tại nút 10.
hamd_4, hamd_8	I	2		Khoảng cách hamming giữa đầu vào và đầu ra tại nút 11.
prv_st_00	O	2	Reg	Nút trước khi chuyển sang nút 00.
prv_st_01	O	2	Reg	Nút trước khi chuyển sang nút 01.
prv_st_10	O	2	Reg	Nút trước khi chuyển sang nút 10.
prv_st_11	O	2	Reg	Nút trước khi chuyển sang nút 11.
sel_node	O	2	Reg	Nút có tổng khoảng cách hamming nhỏ nhất.

2.2.4.4 Lưu đồ thuật toán

Lưu đồ thuật toán của module **add_comp** được biểu diễn trên Hình 2.8.

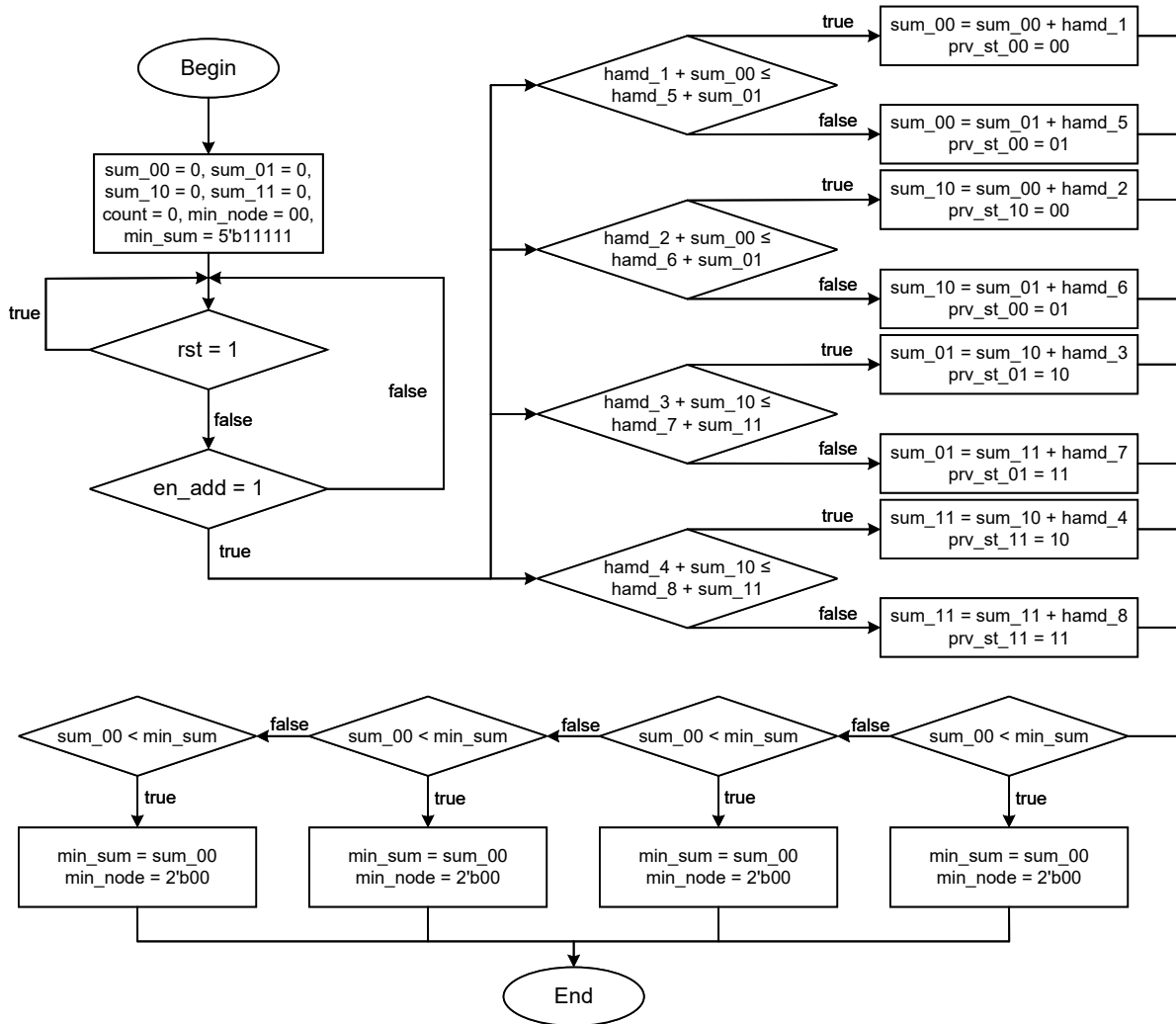
2.2.5 Module **mem**

2.2.5.1 Sơ đồ khối

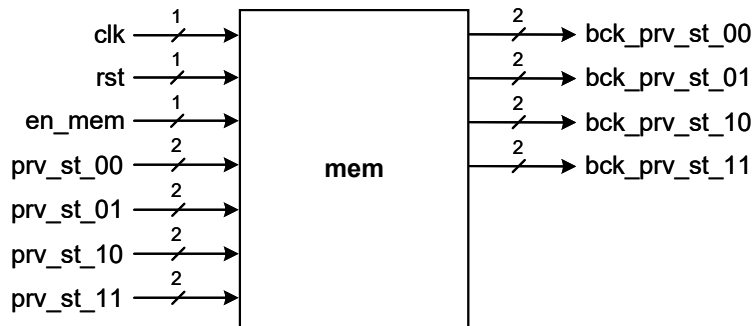
Sơ đồ khối của module **mem** được thể hiện trong Hình 2.9.

2.2.5.2 Chức năng

Khối **mem** là bộ nhớ lưu trữ và truy xuất các trạng thái trước đó trong thuật toán Viterbi. Nó hỗ trợ quá trình traceback để tái tạo lại chuỗi bit đầu vào đã được giải mã. Khối nhận các tín hiệu trạng thái trước (**prv_st_00**, **prv_st_01**, **prv_st_10**, **prv_st_11**) từ khối **add_comp** và lưu vào một mảng hai chiều **trellis_diagr[0:3][0:7]** nhằm biểu diễn sơ đồ Trellis gồm 4 nút và 8 chu kỳ thời gian. Khi đến giai đoạn truy vết ngược (traceback), khối xuất ra các trạng thái trước tương ứng (**bck_prv_st_00**, **bck_prv_st_01**, **bck_prv_st_10**, **bck_prv_st_11**) để xác định chuỗi bit tối ưu. Chức năng của khối được đặc tả như sau:



Hình 2.8 Lưu đồ thuật toán của module add_comp



Hình 2.9 Sơ đồ khối của module mem

- Khi $rst = 1$, khởi tạo $count = 0$, $trace = 7$, gán tất cả các phần tử của $trellis_diagr$ bằng $2'b00$.

- Khi $en_mem = 1$, nếu $count < 8$ thì ghi các giá trị prv_st_xx vào $trellis_diagr[*][count]$, sau đó tăng $count$ lên 1. Khi $count = 8$, đọc các giá trị tại $trellis_diagr[*][count]$ và xuất ra các cổng $bck_prv_st_xx$, giảm

trace từ 7 xuống 0.

2.2.5.3 Đầu vào, đầu ra

Bảng mô tả các chân I/O của module mem được mô tả trên Bảng 2.5.

Bảng 2.5 Bảng mô tả các chân I/O của module mem

Tên chân	I/O	Số bit	Reg	Chức năng
clk	I	1		Tạo xung clock, sườn dương của xung là tín hiệu điều khiển lệnh trong mạch.
rst	I	1		Thiết lập module về trạng thái ban đầu.
en_mem	I	1		Thiết lập module hoạt động.
prv_st_00	I	2		Nút trước khi chuyển sang nút 00.
prv_st_01	I	2		Nút trước khi chuyển sang nút 01.
prv_st_10	I	2		Nút trước khi chuyển sang nút 10.
prv_st_11	I	2		Nút trước khi chuyển sang nút 11.
bck_prv_st_00	O	2	Reg	Nút đi đến nút 00.
bck_prv_st_01	O	2	Reg	Nút đi đến nút 01.
bck_prv_st_10	O	2	Reg	Nút đi đến nút 10.
bck_prv_st_11	O	2	Reg	Nút đi đến nút 11.

2.2.5.4 Lưu đồ thuật toán

Lưu đồ thuật toán của module mem được biểu diễn trên Hình 2.10.

2.2.6 Module tbck_dec

2.2.6.1 Sơ đồ khối

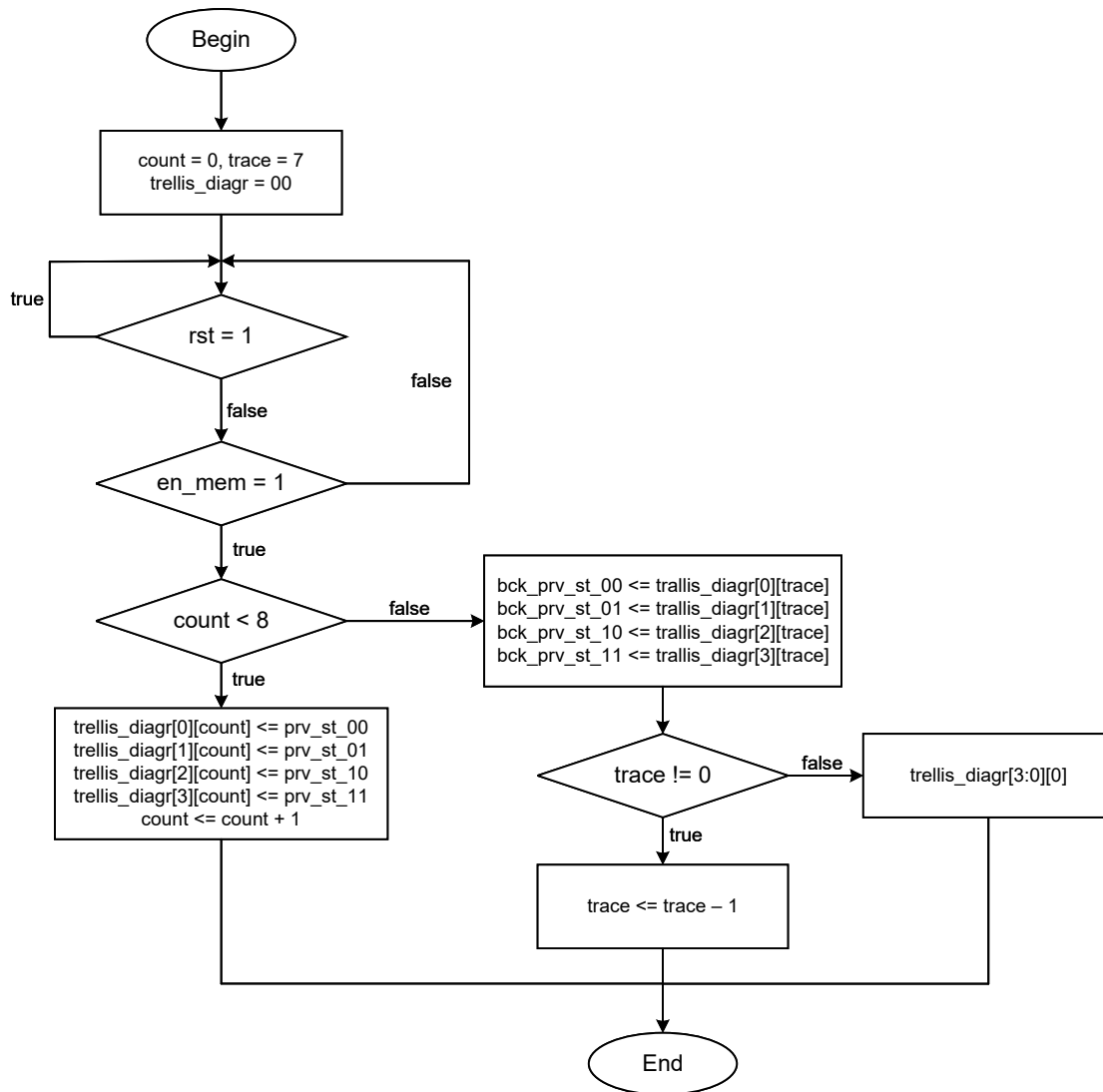
Sơ đồ khối của module tbcb_dec được thể hiện trong Hình 2.11.

2.2.6.2 Chức năng

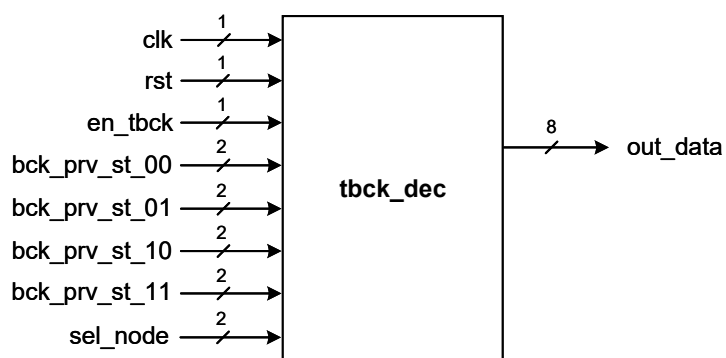
Khối tbck_dec thực hiện truy vết ngược (traceback) trên sơ đồ Trellis nhằm tái tạo lại chuỗi dữ liệu gốc sau khi thuật toán Viterbi hoàn tất quá trình tính toán. Kết quả là chuỗi dữ liệu 8 bit được xuất ra qua cổng out_data. Khối sử dụng FSM gồm 4 trạng thái tương ứng với các nút trong Trellis: 00 (S_0), 01 (S_1), 10 (S_2), 11 (S_3). Chức năng của khối được đặc tả như sau:

- Khi rst = 1, khởi tạo toàn bộ các thanh ghi và tín hiệu về 0: out_data, count, sel_node, sel_bit_out.

- Khi en_tbck = 1, FSM bắt đầu truy vết ngược trên trellis. Tại mỗi chu kỳ clk:



Hình 2.10 Lưu đồ thuật toán của module mem



Hình 2.11 Sơ đồ khối của module tbck_dec

- + Xác định trạng thái trước dựa trên sel_node hiện tại.
- + Ghi lại bit đầu vào tương ứng vào sel_bit_out[count].
- + Cập nhật sel_node bằng giá trị trạng thái trước (bck_prev_st_xx).

- + Tăng biến `count` để theo dõi số bit đã truy vết.
- Khi `count = 8` (đã truy ngược đủ 8 bước):
- + Gán `out_data = sel_bit_out` (chuỗi dữ liệu đã giải mã).
- + Reset `count` để sẵn sàng cho chu kỳ kế tiếp.

2.2.6.3 Đầu vào, đầu ra

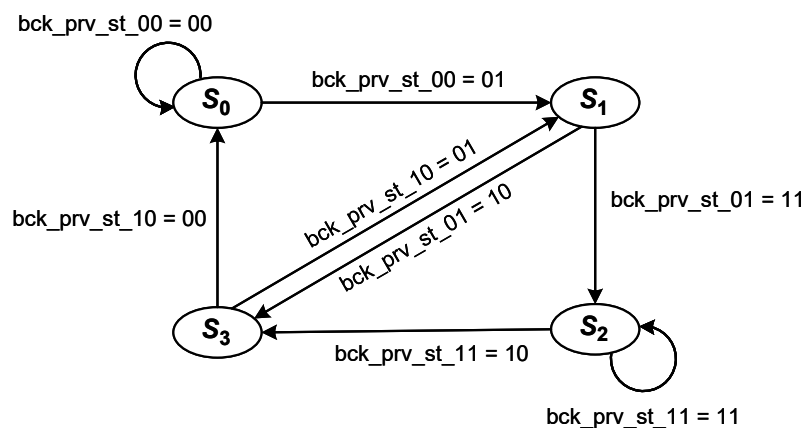
Bảng mô tả các chân I/O của module `tbck_dec` được mô tả trên Bảng 2.6.

Bảng 2.6 Bảng mô tả các chân I/O của module `tbck_dec`

Tên chân	I/O	Số bit	Reg	Chức năng
<code>clk</code>	I	1		Tạo xung clock, sườn dương của xung là tín hiệu điều khiển lệnh trong mạch.
<code>rst</code>	I	1		Thiết lập module về trạng thái ban đầu.
<code>en_tbck</code>	I	1		Thiết lập module hoạt động.
<code>bck_prv_st_00</code>	I	2		Nút đi đến nút 00.
<code>bck_prv_st_01</code>	I	2		Nút đi đến nút 01.
<code>bck_prv_st_10</code>	I	2		Nút đi đến nút 10.
<code>bck_prv_st_11</code>	I	2		Nút đi đến nút 11.
<code>out_data</code>	O	8	Reg	8 bit data sau giải mã.

2.2.6.4 Sơ đồ trạng thái FSM

Như đã nhắc đến trong Mục 2.2.6.2, module `tbck_dec` hoạt động như một FSM với 4 trạng thái. Mô hình FSM được biểu diễn trên Hình 2.12.



Hình 2.12 Sơ đồ trạng thái FSM của module `tbck_dec`

TÀI LIỆU THAM KHẢO

- [1] G. R. Ryan, M. S. và Nudd, “The viterbi algorithm,” February 1993.
- [2] N. Bình, *Bài giảng Lý thuyết thông tin*. Học viện Công nghệ Bưu chính Viễn thông, 2006.
- [3] P. H. Đăng, *Cơ sở lý thuyết thông tin. Chương 5: Mã tích chập, Thuật toán giải mã Viterbi*. Đại học Bách khoa Hà Nội, 2013.
- [4] MIT Department of Electrical Engineering and Computer Science, “Mit 6.02: Viterbi decoding of convolutional codes (lecture 9),” Lecture Notes, MIT OpenCourseWare, 2010, draft version, last update: October 6, 2010. [Online]. Available: <https://web.mit.edu/6.02/www/f2010/handouts/lectures/L9.pdf>