ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN - ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ

–oo0oo–

BÁO CÁO ĐỒ ÁN 1

Design and Implementation of a Power-Efficient Viterbi Encoding and Decoding Architecture on FPGA: From Theory to Practical Application

Giảng viên hướng dẫn: Nguyễn Trung Hiếu
Sinh viên thực hiện: Nguyễn Đại Đồng
Mã số sinh viên: 2210780

Tp.HCM, .../.../20...

LỜI CẢM ƠN

Trước tiên, tôi xin gửi lời cảm ơn chân thành đến các thầy cô trong khoa Điện tử - Viễn thông, những người đã tận tình giảng dạy, truyền đạt kiến thức và kinh nghiệm quý báu trong suốt quá trình học tập, giúp tôi có nền tảng vững chắc để thực hiện đồ án này. Đặc biệt, tôi xin bày tỏ lòng biết ơn sâu sắc đến thầy Nguyễn Trung Hiếu, người đã trực tiếp hướng dẫn, hỗ trợ và định hướng tôi trong suốt quá trình nghiên cứu và hoàn thiện đồ án. Những góp ý quý giá và sự động viên của thầy đã giúp tôi vượt qua nhiều khó khăn và hoàn thành công việc một cách tốt nhất.

Tôi cũng xin gửi lời cảm ơn đến gia đình, bạn bè và các đồng nghiệp đã luôn ủng hộ, động viên và chia sẻ cùng tôi trong suốt quá trình thực hiện. Sự hỗ trợ về tinh thần lẫn vật chất từ mọi người là nguồn động lực lớn lao để tôi hoàn thành đồ án này.

Cuối cùng, tôi xin cảm ơn các tài liệu tham khảo, nguồn thông tin từ sách, báo, và các công cụ mô phỏng như ngôn ngữ lập trình C và SystemVerilog, đã cung cấp nền tảng lý thuyết và thực tiễn để tôi triển khai thành công nội dung đồ án. Mặc dù đã cố gắng hết sức, đồ án vẫn có thể còn những thiếu sót, tôi rất mong nhận được những ý kiến đóng góp để hoàn thiện hơn.

Trân trọng,

Tp. Hồ Chí Minh, ngày . . . tháng . . . năm . . .

Sinh viên

TÓM TẮT ĐỒ ÁN

Đồ án này trình bày về quá trình nghiên cứu, thiết kế và mô phỏng thuật toán giải mã Viterbi trong hệ thống truyền thông số, tập trung vào mã hóa và giải mã kênh sử dụng mã chập (Convolutional Codes). Nội dung bao gồm cơ sở lý thuyết về mã hóa kênh truyền, cấu trúc bộ mã hóa chập, và nguyên lý hoạt động của thuật toán Viterbi, một phương pháp giải mã tối ưu dựa trên quy hoạch động. Đồ án mô phỏng thuật toán Viterbi bằng ngôn ngữ lập trình C và SystemVerilog, với việc triển khai các khối chức năng như Branch Metric Unit (BMU), Add Compare Select Unit (ACSU) và Survivor Path Memory Unit (SPMU). Kết quả mô phỏng cho thấy khả năng khôi phục chính xác chuỗi bit gốc từ tín hiệu bị nhiễu, với ví dụ cụ thể sử dụng bộ mã hóa chập (3,1,2) và chuỗi đầu vào "110110". Đồ án không chỉ cung cấp cái nhìn sâu sắc về kỹ thuật mã hóa và giải mã mà còn chứng minh tính hiệu quả của thuật toán Viterbi trong việc cải thiện độ tin cậy của hệ thống truyền thông.

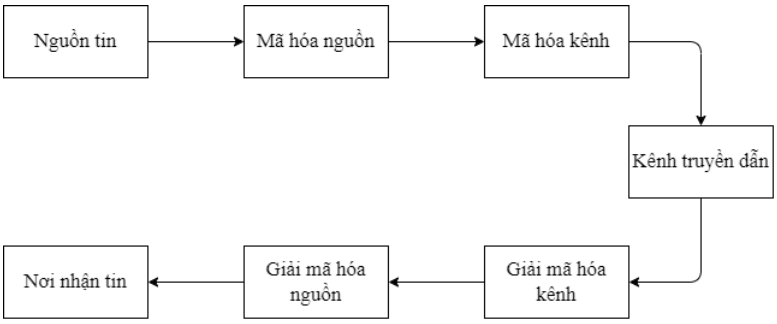
Mục lục

Danh sách hình vẽ

Danh sách bảng

CHƯƠNG 1. CỞ SỞ LÝ THUYẾT

1.1. Mã hóa kênh truyền trong lĩnh vực truyền thông thông tin



Hình 1: Mô hình hệ thống truyền thông số.

Mã hóa kênh truyền (Channel Coding) là một khâu quan trọng trong hệ thống truyền thông tin cùng với khối mã hóa nguồn. Mã hóa kênh là kỹ thuật thêm thông tin dư thừa (redundant information) vào tín hiệu gốc trước khi truyền để phát hiện và/hoặc sửa lỗi trong quá trình truyền qua kênh nhiễu.

Ở hình 1 thể hiện hệ thống truyền thông đơn giản trong đó,

- Nguồn tin: là nơi tạo ra thông tin cần truyền đi hay là thông tin mong muốn truyền.
- Mã hóa nguồn: là bộ mã hóa cho phép loại bỏ những thông tin dư thừa của chuỗi đầu vào.
- Mã hóa kênh: là bộ mã hóa cho phép ghép thêm các thông tin dư thừa vào chuỗi đầu vào.
- Kênh truyền dẫn: là các phương thức truyền thông tin có thể là không dây hoặc là hữu tuyến, vệ tinh.

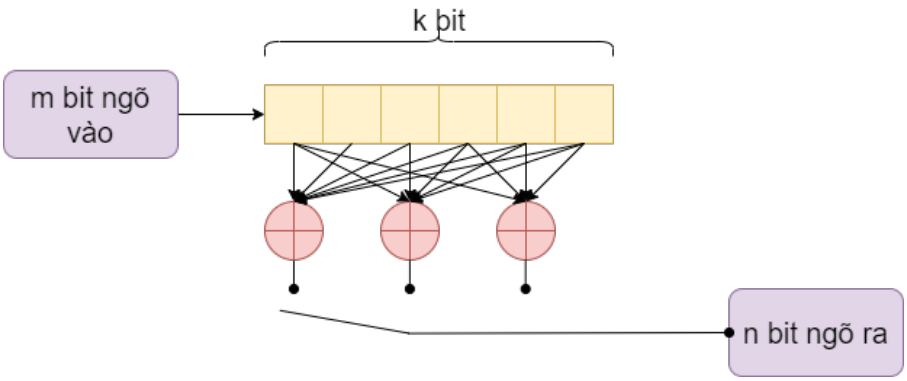
Việc thêm khối mã hóa kênh vào hệ thống truyền thông số là do thường tín hiệu trong truyền thông thực tế sẽ thường xảy ra các lỗi như: Lỗi ngẫu nhiên (Random Errors) do nhiễu trắng Gaussian (AWGN), Lỗi theo cụm (Burst Errors) do fading đa đường hoặc mất gói dữ liệu trong truyền thông không dây. Vì vậy, cần thêm khối mã hóa kênh vào để giúp cải thiện tỷ lệ lỗi bit (BER - Bit Error Rate) mà không cần tăng công suất truyền, còn giúp đảm bảo tính toàn vẹn của dữ liệu trong truyền thông không dây, hữu tuyến, và vệ tinh.

Có các loại mã hóa kênh phổ biến là:

- Mã khối (Block Codes): Dữ liệu được chia thành các khối có độ dài cố định, và các bit dư thừa thêm vào mỗi khối. Ví dụ: Mã Hamming, Mã Reed-Solomon.

- Mã chập (Convolutional Codes): Dữ liệu được mã hóa liên tục thông qua một bộ mã hóa dựa vào các thanh ghi dịch và phép toán XOR. Đặc trưng bởi các tham số: Độ dài ràng buộc (Constraint length), Tỷ lệ mã hóa (Code rate).
- Mã Turbo (Turbo Codes): Kết hợp nhiều bộ mã hóa và sử dụng kỹ thuật lặp (Iterative decoding) để đạt hiệu suất gần với giới hạn lý thuyết.
- Mã LDPC (Low-Density Parity-Check Codes): Sử dụng ma trận kiểm tra chẵn lẻ (Parity-check matrix) thưa thớt để đạt hiệu suất cao.

1.2. Mã hóa sử dụng Mã chập (Convolutional Encoder)



Hình 2: Cấu trúc tổng quát của một bộ mã hóa chập

Mã chập được tạo ra bằng cách cho chuỗi thông tin truyền qua hệ thống các thanh ghi dịch tuyến tính có số trạng thái hữu hạn. Số lượng thanh ghi là K, trong đó $L = K + 1$ là **chiều dài ràng buộc** (constraint length). Bộ mã hóa chập có m bit ngõ vào và n bit ngõ ra. Tốc độ mã là $r = m/n$, là tỉ số giữa số bit ngõ vào và số bit ngõ ra của tín hiệu. Tham số K thể hiện số lượng bit đầu vào (bao gồm bit đầu vào hiện tại và các bit đầu vào trước đó được lưu trong thanh ghi) ảnh hưởng đến đầu ra.

Một bộ mã hóa chập có cấu trúc gồm các thanh ghi dịch và các bộ cộng modulo-2. Các thanh ghi dịch lưu trữ các bit đầu vào trước đó, và các bộ cộng modulo-2 kết hợp các bit đầu vào hiện tại với các bit trong thanh ghi để tạo ra các bit đầu ra.

Giả sử, u là vector đầu vào, x là vector đầu ra tương ứng được mã hóa từ vector u, bây giờ chúng ta mô tả cách tạo ra x từ u. Để mô tả bộ mã hóa chúng ra cần phải kết nối vector u vào thanh ghi đầu vào và x vào thanh ghi đầu ra trong hình 2. Để có thể tính toán một cách dễ dàng, chúng ta tiến hành mô hình hóa bộ mã ở hình 2 thành một ma trận sinh G có dạng :

$$\mathbf{G} = \begin{bmatrix} g_{1,0} & g_{1,1} & g_{1,2} & \cdots & g_{1,K-1} \\ g_{2,0} & g_{2,1} & g_{2,2} & \cdots & g_{2,K-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_{n,0} & g_{n,1} & g_{n,2} & \cdots & g_{n,K-1} \end{bmatrix}$$

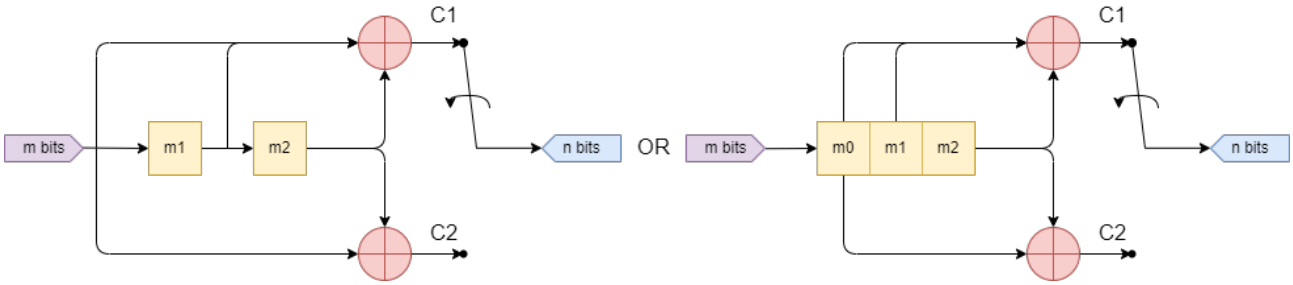
Trong đó:

- $g_{i,j}$: Hệ số kết hợp (0 hoặc 1) cho biết bit đầu vào thứ j có tham gia vào bit đầu ra thứ i hay không.
- $g_{i,j} = 1$: Bit đầu vào thứ j được sử dụng để tính bit đầu ra thứ i.
- $g_{i,j} = 0$: Bit đầu vào thứ j không được sử dụng để tính bit đầu ra thứ i.

Vậy để được vector x thì ta chỉ cần thực hiện phép nhân ma trận G và vector u lại với nhau:

$$x = u \cdot G$$

Để hiểu và phân tích Mã chập, có nhiều phương pháp biểu diễn khác nhau. Mỗi phương pháp biểu diễn có mục đích và ứng dụng riêng, giúp thiết kế, phân tích và triển khai mã chập một cách hiệu quả. Tuy nhiên, có các phương pháp chính sau [1]:



Hình 3: Bộ mã hóa $(3, 1, 2)$.

Trong đó, $C1 = m + m_1 + m_2$ và $C2 = m + m_2$.

1.2.1. Ma trận sinh (Generator matrix)

Sử dụng phương pháp biểu diễn mã chập bằng ma trận sinh, ta có thể lấy bộ mã hóa ở hình 3 để thể hiện cách hoạt động của ma trận sinh. Từ hình 3 ta có thể suy ra ma trận sinh G là:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

Giả sử, $m_t = 1$, $m_{t-1} = 0$, $m_{t-2} = 1$, khi đó:

$$n = m \cdot G = \begin{bmatrix} m_t & m_{t-1} & m_{t-2} \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

1.2.2. Sơ đồ lưới (Trellis Diagram)

Sơ đồ trạng thái biểu diễn các trạng thái của thanh ghi và các chuyển tiếp giữa các trạng thái dựa trên bit đầu vào và bit đầu ra. Trong sơ đồ trạng thái, mỗi nút biểu diễn

[1] Để trực quan hơn, sau đây ta sẽ biểu diễn một bộ mã hóa chập đơn giản sau với $K = 3, n = 2, m = 1$.

cho một trạng thái của thanh ghi. Các nội dung biểu diễn chuyển tiếp giữa các trạng thái, được gán nhãn bởi bit vào và bit đầu ra tương ứng.

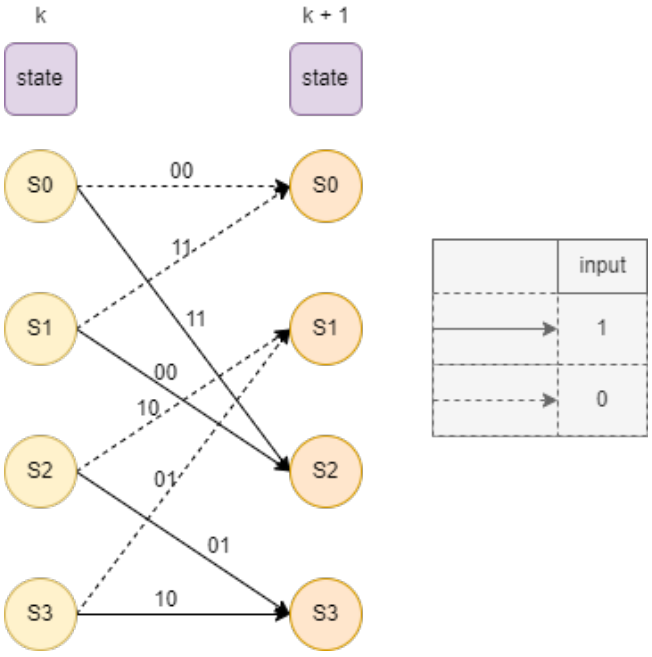
Để làm rõ hơn việc sử dụng sơ đồ trạng thái trong việc biểu diễn mã chập, ta sử dụng một bộ mã hóa chập ở hình 3. Với $K = 3$ ta có tương ứng với 4 trạng thái là: `S0(00)`, `S1(01)`, `S2(10)`, `S3(11)`, và trạng thái chuyển tiếp có thể được tạo từ trạng thái này sang trạng thái khác, quá trình chuyển tiếp có thể là:

	Next State, if		Output symbol, if	
Current State	Input = 0	Input = 1	Input = 0	Input = 1
S0	S0	S2	00	11
S1	S0	S2	11	00
S2	S1	S3	10	01
S3	S1	S3	01	10

Bảng 1: Bảng chuyển trạng thái của mã chập $(3, 1, 2)$.

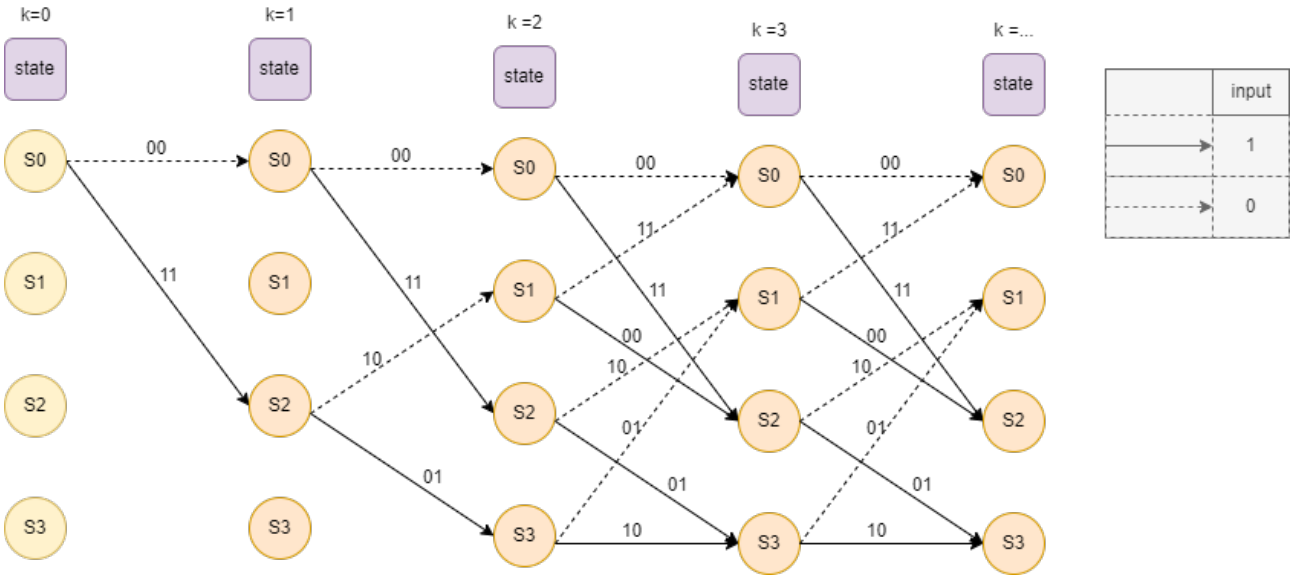
Sơ đồ lưới là một mở rộng của sơ đồ trạng thái theo thời gian, biểu diễn tất cả các đường đi có thể của mã chập qua các bước thời gian. Sơ đồ lưới bao gồm các nút biểu diễn trạng thái tại mỗi bước thời gian. Các cung biểu diễn chuyển tiếp giữa các trạng thái, được gán nhãn bởi bit đầu vào và bit đầu ra.

Ở hình 3, ta có thể biểu diễn một cung chuyển tiếp đơn giản giữa hai đơn vị thời gian như sau:



Hình 4: Sơ đồ lưới giữa hai đơn vị thời gian của bộ mã hóa $(3, 1, 2)$.

Từ đó, ta có thể biểu diễn hoàn chỉnh cách biểu diễn bộ mã hóa chập bằng sơ đồ lưới như sau:



Giả sử, chuỗi bit ngõ vào là `0101` thì ngõ ra tương ứng với chuỗi bit sau khi được mã hóa sẽ là `00 11 10 00`.

1.3. Giải mã kênh truyền trong lĩnh vực truyền thông thông tin

Giải mã kênh truyền là quá trình khôi phục thông tin gốc từ tín hiệu đã được mã hóa trên kênh truyền. Nó là bước ngược lại của quá trình mã hóa kênh, nhằm đưa dữ liệu về trạng thái ban đầu. Hiệu quả của giải mã kênh phụ thuộc vào nhiều yếu tố, bao gồm mô hình trạng thái và phương pháp giải mã được sử dụng. Mục tiêu chính là giảm thiểu lỗi trong quá trình truyền dữ liệu.

Ở hình 1 thể hiện hệ thống truyền thông đơn giản trong đó,

- Giải mã nguồn: khôi phục dữ liệu nguồn.
- Giải mã kênh: sửa lỗi do nhiễu trong quá trình truyền dẫn, đảm bảo dữ liệu nhận được gần nhất với dữ liệu gốc.

1.4. Giải mã kênh truyền sửa dụng thuật toán Viterbi

Thuật toán Viterbi là một giải pháp được sử dụng phổ biến để giải mã chuỗi bit được mã hóa bởi bộ mã hóa tích chập. Chi tiết của một bộ giải mã riêng phụ thuộc vào một bộ mã hóa tích chập tương ứng. Thuật toán Viterbi không phải là một thuật toán đơn lẻ có thể dùng để giải mã những chuỗi bit mà được mã hóa bởi bất cứ một bộ mã hóa chập nào.

Thuật toán Viterbi được khởi xướng bởi Andrew Viterbi vào năm 1967 như là một thuật toán giải mã cho mã chập qua các tuyến thông tin có nhiễu. Nó được sử dụng trong cả hai hệ thống CDMA và GSM, các modem số, vệ tinh, thông tin vũ trụ, và các hệ thống mạng cục bộ không dây. Hiện nay còn được sử dụng phổ biến trong kỹ thuật nhận dạng giọng nói, nhận dạng từ mã, ngôn ngữ học máy tính.

Thuật toán Giải mã Vitervi là một trong hai loại thuật toán giải mã được sử dụng với bộ mã hóa chập - là một loại giải mã tuần tự. Ưu điểm của giải mã tuần tự so với Viterbi là nó có thể hoạt động tốt với các mã chập có chiều dài ràng buộc lớn, nhưng nó lại có thời gian giải mã biến đổi.

Còn ưu điểm cảu thuật toán Giải mã Viterbi là nó có thời gian mã ổn định. Điều đó rất tốt cho việc thực thi bộ giải mã bằng phần cứng. Nhưng mà yêu cầu về sự tính toán của nó tăng theo hàm mũ như là một hàm của chiều dài rằng buộc. Vì vậy trong thực tế, người ta thường giới hạn chiều dài ràng buộc của nó là 9 ($k \leq 9$).

Thuật toán Viterbi là một giải pháp giải mã tối ưu cho các chuỗi bit được mã hóa bởi bộ mã hóa tích chập (convolutional encoder). Đây là thuật toán quyết định cứng (hard-decision) sử dụng nguyên lý quy hoạch động để tìm chuỗi trạng thái có xác suất cao nhất trong lưới trạng thái (trellis diagram).

Đặc điểm chính

- **Tính tương thích**: Mỗi bộ giải mã Viterbi phải được thiết kế tương ứng với một bộ mã hóa tích chập cụ thể.
- **So sánh với giải mã tuần tự**:
  + Ưu điểm: Hiệu suất giải mã ổn định, độ phức tạp tính toán thấp
  + Nhược điểm: Khó áp dụng cho mã có độ dài ràng buộc (constraint length) lớn
- **Ứng dụng**:
  + Hệ thống thông tin di động (GSM, CDMA)
  + Truyền dẫn vệ tinh
  + Nhận dạng tiếng nói

Nguyên lý hoạt động

Thuật toán thực hiện các bước chính:

1. Khởi tạo metric cho các trạng thái
2. Tính toán độ tương đồng (branch metric)
3. Cập nhật path metric
4. Truy vết ngược (traceback) để tìm chuỗi giải mã tối ưu

Phương trình cập nhật metric:

$$\gamma_t(s', s) = \gamma_{t-1}(s') + \sum_{i=1}^{n} (r_i^{(t)} - c_i^{(t)})^2$$

trong đó:

- s', s: các trạng thái liên tiếp
- $r_i^{(t)}$: bit nhận được tại thời điểm t
- $c_i^{(t)}$: bit mã hóa tương ứng

1.5. Hard Decision và Soft Decision trong Giải Mã Mã Hóa Sửa Lỗi

Trong các hệ thống thông tin số, mã hóa sửa lỗi (error-correction coding) đóng vai trò quan trọng trong việc đảm bảo dữ liệu được truyền qua kênh nhiễu một cách đáng tin cậy. Hai phương pháp giải mã phổ biến là **hard decision** và **soft decision**, được áp dụng cho cả mã khối (block codes) và mã chập (convolutional codes). Hard decision đơn giản hóa quá trình giải mã bằng cách đưa ra quyết định nhị phân, trong khi soft decision tận dụng thông tin xác suất để cải thiện hiệu suất. Phần tiếp theo sẽ trình bày cơ sở lý thuyết của hai phương pháp này [1].

1.5.1. Hard decision (Quyết định cứng)

Hard decision là phương pháp giải mã trong đó bộ giải mã nhận đầu vào dưới dạng các giá trị nhị phân (0 hoặc 1). Tại bộ giải điều chế (demodulator), tín hiệu nhận được sẽ được so sánh với một ngưỡng (threshold) để quyết định giá trị bit:

- Nếu tín hiệu vượt ngưỡng, bit được gán giá trị 1.
- Nếu tín hiệu dưới ngưỡng, bit được gán giá trị 0.

Quá trình này bỏ qua thông tin về độ tin cậy của tín hiệu, chỉ dựa vào quyết định cứng (hard decision).

Trong hard decision, bộ giải mã sử dụng khoảng cách Hamming (Hamming distance) để xác định chuỗi mã (codeword) gần nhất với chuỗi nhận được. Khoảng cách Hamming được định nghĩa là số vị trí mà hai chuỗi bit khác nhau. Ví dụ, với mã khối, thuật toán giải mã sẽ tìm chuỗi mã hợp lệ sao cho:

$$d_H(r, c) = \min_{c \in C} d_H(r, c),$$

trong đó r là chuỗi nhận được, c là chuỗi mã hợp lệ trong tập mã C, và d_H là khoảng cách Hamming.

Ưu và nhược điểm

- Ưu điểm: Đơn giản, dễ triển khai trên phần cứng, yêu cầu tính toán thấp.
- Nhược điểm: Mất thông tin về độ tin cậy của tín hiệu, dẫn đến hiệu suất kém hơn trong kênh nhiễu mạnh, đặc biệt với các mã chập khi sử dụng thuật toán Viterbi.

1.5.2. Soft decision (Quyết định mềm)

Soft decision là phương pháp giải mã sử dụng thông tin xác suất hoặc giá trị liên tục từ bộ giải điều chế, thay vì quyết định nhị phân. Thay vì gán bit 0 hoặc 1, bộ giải mã nhận các giá trị thực (real-valued) đại diện cho độ tin cậy của mỗi bit. Phương pháp này tận dụng thông tin bổ sung để cải thiện khả năng sửa lỗi.

Trong soft decision, bộ giải mã thường sử dụng khoảng cách Euclidean (Euclidean distance) thay vì khoảng cách Hamming. Với chuỗi nhận được $r = (r_1, r_2, \ldots, r_n)$ và chuỗi mã $c = (c_1, c_2, \ldots, c_n)$, khoảng cách Euclidean được tính như sau:

$$d_E(r, c) = \sqrt{\sum_{i=1}^{n} (r_i - c_i)^2}.$$

Bộ giải mã tìm chuỗi mã c sao cho $d_E(r, c)$ là nhỏ nhất. Trong trường hợp mã chập, thuật toán Viterbi sử dụng soft decision bằng cách tính path metric dựa trên giá trị xác suất hoặc độ tin cậy của mỗi bit, thay vì chỉ đếm lỗi nhị phân.

Ưu và nhược điểm

- Ưu điểm: Cải thiện hiệu suất giải mã, đặc biệt trong kênh nhiễu Gaussian, nhờ sử dụng thông tin độ tin cậy. Hiệu suất có thể cải thiện 2–3 dB so với hard decision.
- Nhược điểm: Phức tạp hơn, yêu cầu tính toán cao hơn và bộ nhớ lớn hơn, đặc biệt khi triển khai trên phần cứng.

1.5.3. So sánh Hard Decision và Soft Decision

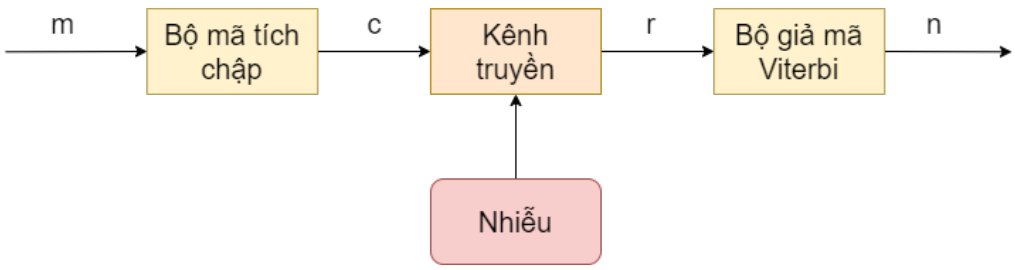
Hard decision và soft decision khác nhau ở cách xử lý đầu vào và độ phức tạp:

- Đầu vào: Hard decision chỉ nhận giá trị nhị phân, trong khi soft decision sử dụng giá trị liên tục hoặc xác suất.
- Hiệu suất: Soft decision thường vượt trội hơn trong môi trường nhiễu mạnh, đặc biệt với mã chập khi sử dụng thuật toán Viterbi.
- Độ phức tạp: Hard decision đơn giản hơn, phù hợp với các hệ thống có tài nguyên hạn chế, trong khi soft decision yêu cầu phần cứng mạnh hơn.

Cả hai phương pháp đều được sử dụng rộng rãi trong các hệ thống thông tin:

- Hard decision: Thường được dùng trong các hệ thống đơn giản như truyền dữ liệu tốc độ thấp hoặc khi tài nguyên phần cứng hạn chế.
- Soft decision: Được áp dụng trong các hệ thống hiện đại như truyền hình vệ tinh (DVB-S2), mạng không dây (Wi-Fi, LTE), và các hệ thống yêu cầu hiệu suất cao.

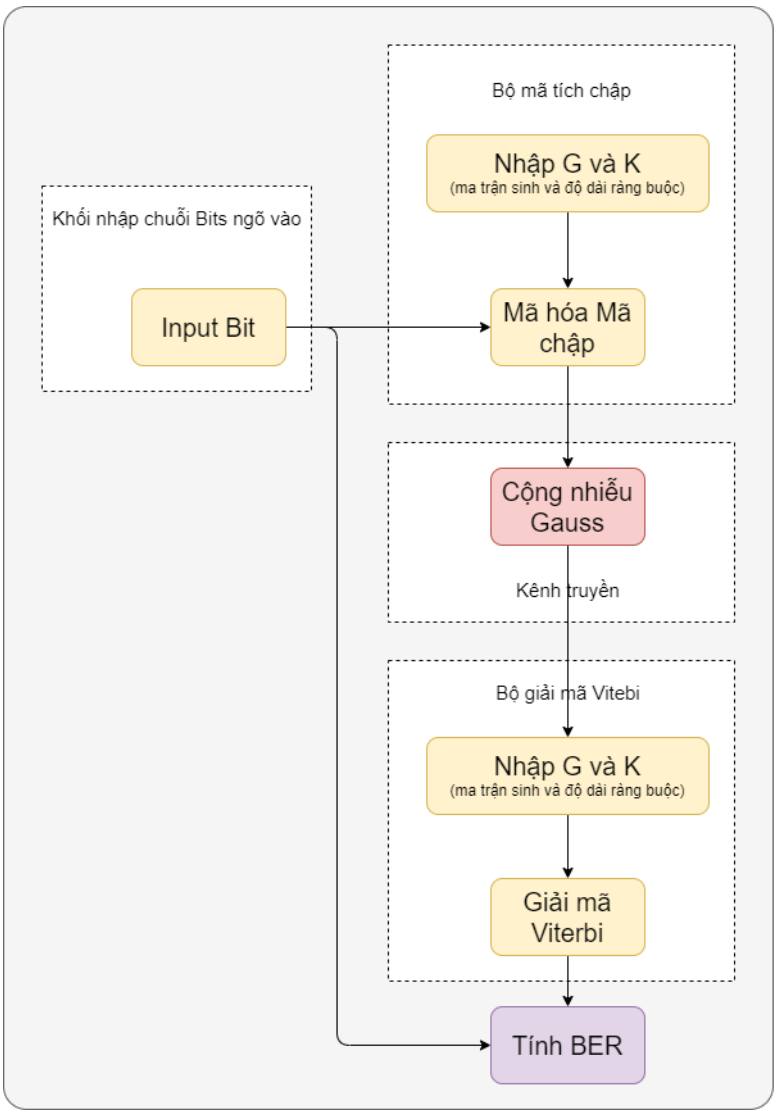
CHƯƠNG 2. MÔ PHỎNG THUẬT TOÁN BẰNG C



Hình 5: Sơ đồ khối hệ thống.

Tín hiệu sau khi được số hóa thành các bit, các bit này sẽ được đưa đến bộ mã hóa Mã chập. Sau khi được mã hóa, tín hiệu (các bit) đươc truyền qua kênh truyền có nhiễu, ở đây có thể coi nhiễu là nhiễu Gauss trắng. Tín hiệu đã bị thay đổi bởi nhiễu được thu vào và giải mã bởi bộ giải mã Viterbi. Ngờ thuật toán Viterbi, tín hiệu được giải mã sẽ gần giống nhất với tín hiệu ban đầu.

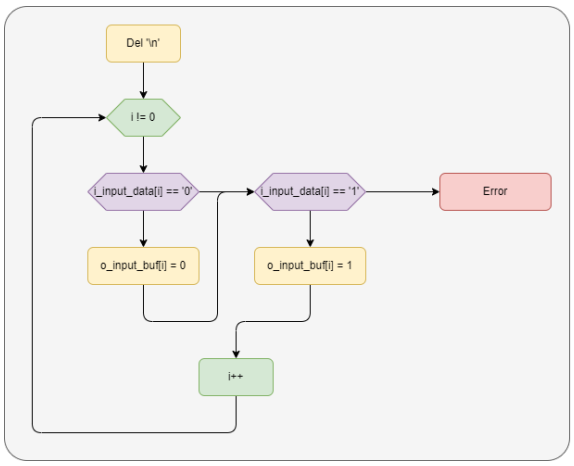
Dựa vào hình 5, ta thực hiện theo sơ đồ sau:



Hình 6: Lưu đồ mô phỏng.

2.1. Khối nhận chuỗi Bits đầu vào

Cho phép người dùng nhập giá trị chuỗi bit đầu vào ở dạng chuỗi (`char`) và chuỗi chuỗi vừa nhận thành mảng một chuỗi chứa các giá trị số thực (`int`).



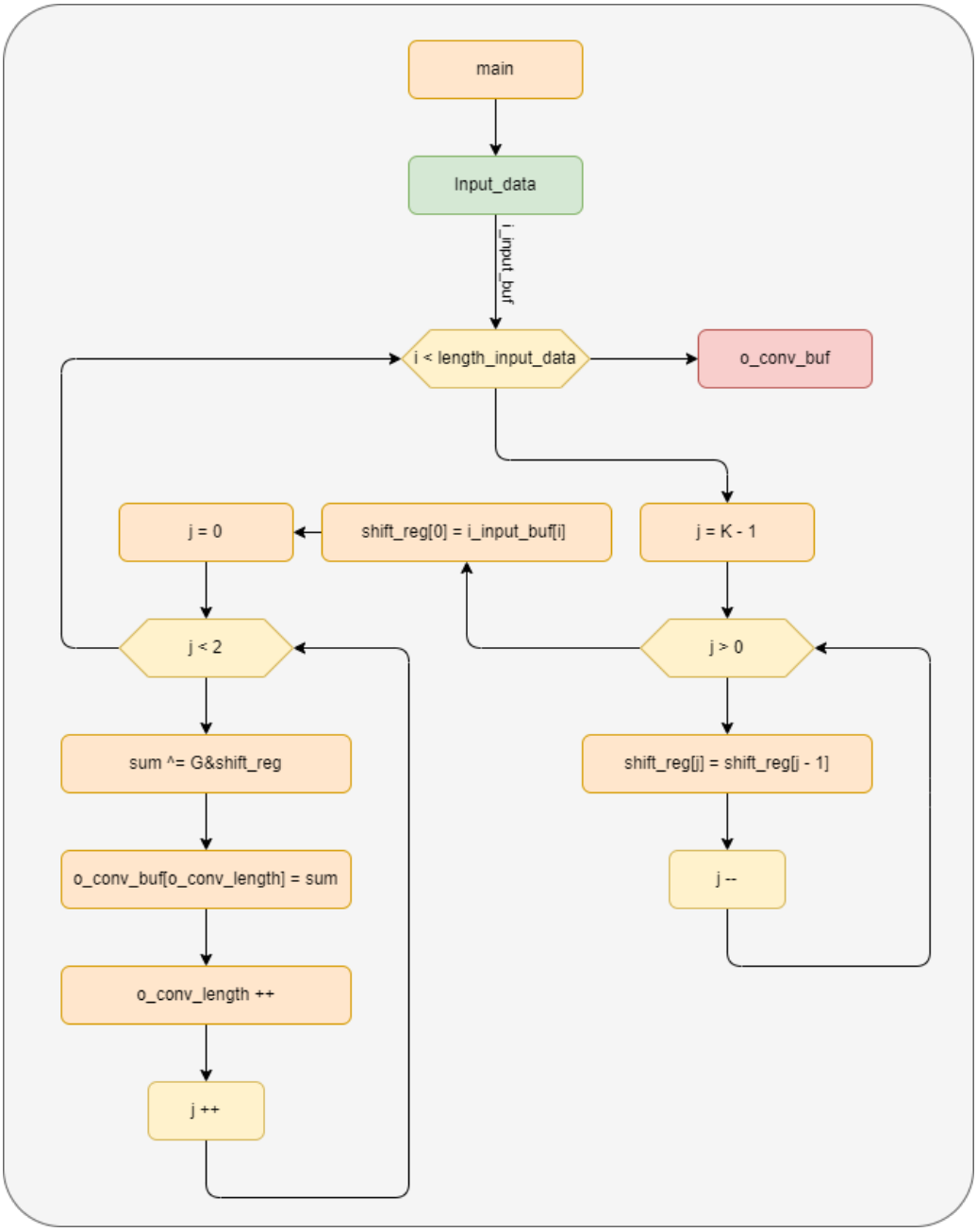
Hình 7: Lưu đồ của Khối nhận chuỗi bits đầu vào.

2.2. Bộ mã tích chập

Với việc mô phỏng bộ Viterbi $(3, 1, 2)$, thì giá trị G và K được cho trước là:

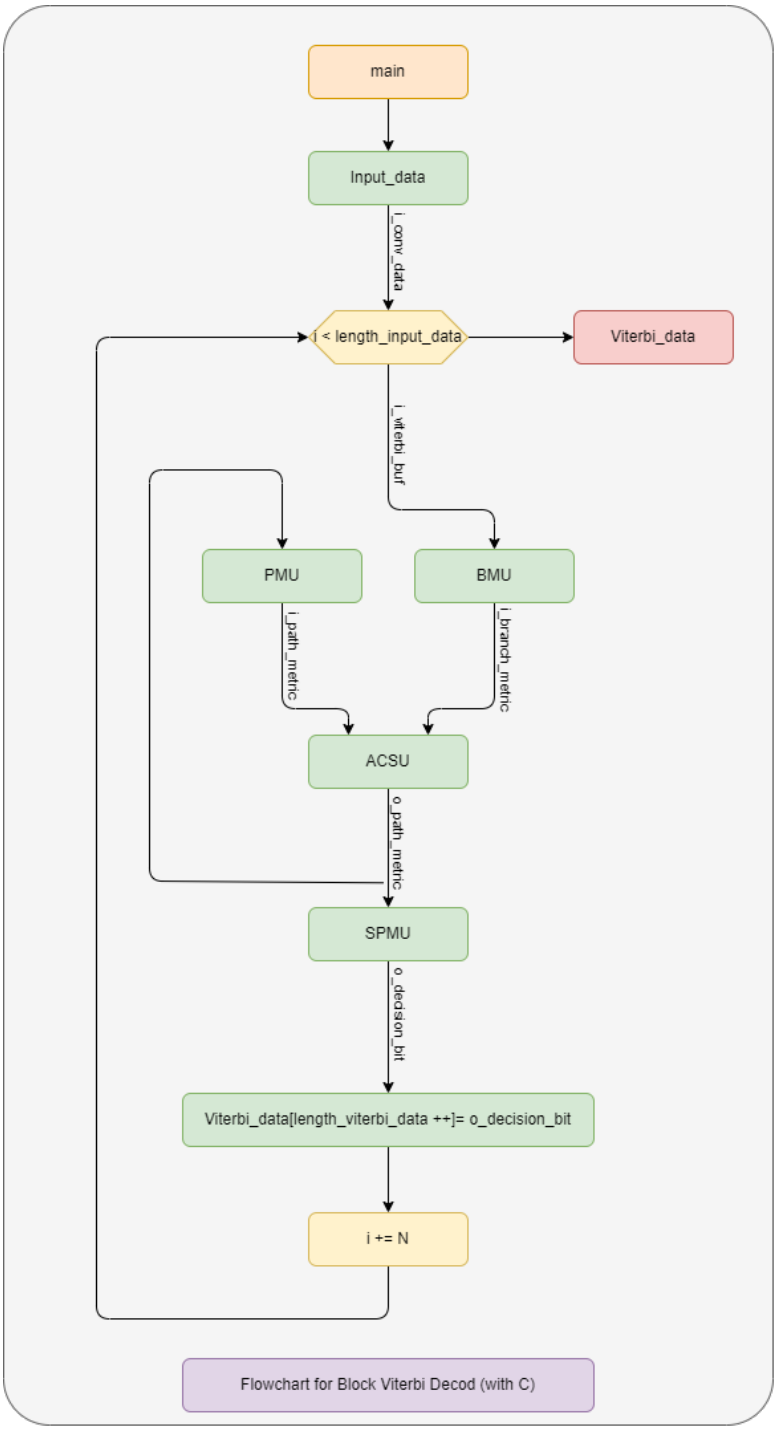
$$G = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}, K = 3$$

Ta có lưu đồ giải thuật của Bộ mã Tích chập,



Hình 8: Lưu đồ giải thuật của Bộ mã Tích chập.

2.3. Bộ giải mã Viterbi



Hình 9: Lưu đồ giải thuật của Bộ giải mã Viterbi.

2.4. Kết quả

```
------------------- Input Data -------------------
Input data: 110110
The constraint length K = 3
The number of input M = 1
The number of output N = 2
```

```
Enter the Generator Polynomial Matrix (G[n][k]): = [(1 1 1) (1 0 1)]

Input data: 110110

------------- Ouput Conv Data -------------

Conv Data: 110101000101


------------------- Input Data -------------------
Input data: 110110
The constraint length K = 3
The number of input M = 1
The number of output N = 2
Enter the Generator Polynomial Matrix (G[n][k]): = [(1 1 1) (1 0 1)]

Input data: 110110

------------- Ouput Conv Data -------------

Conv Data: 110101000101

------------- Viterbi Decoder -------------
Viterbi Data: 110101000101 - BER = 0
Viterbi Data: 110110
```

Listing 1: Mô phỏng việc mã hóa và giải mã với 0bit sai.

```
------------------- Input Data -------------------
Input data: 110110
The constraint length K = 3
The number of input M = 1
The number of output N = 2
Enter the Generator Polynomial Matrix (G[n][k]): = [(1 1 1) (1 0 1)]

Input data: 110110

------------- Ouput Conv Data -------------

Conv Data: 110101000101

------------- Viterbi Decoder -------------
Viterbi data: 110111000101 - BER = 1
Viterbi Data: 110110
```

Listing 2: Mô phỏng việc mã hóa và giải mã với 1bit sai.

```
------------------- Input Data -------------------
Input data: 110110
The constraint length K = 3
```

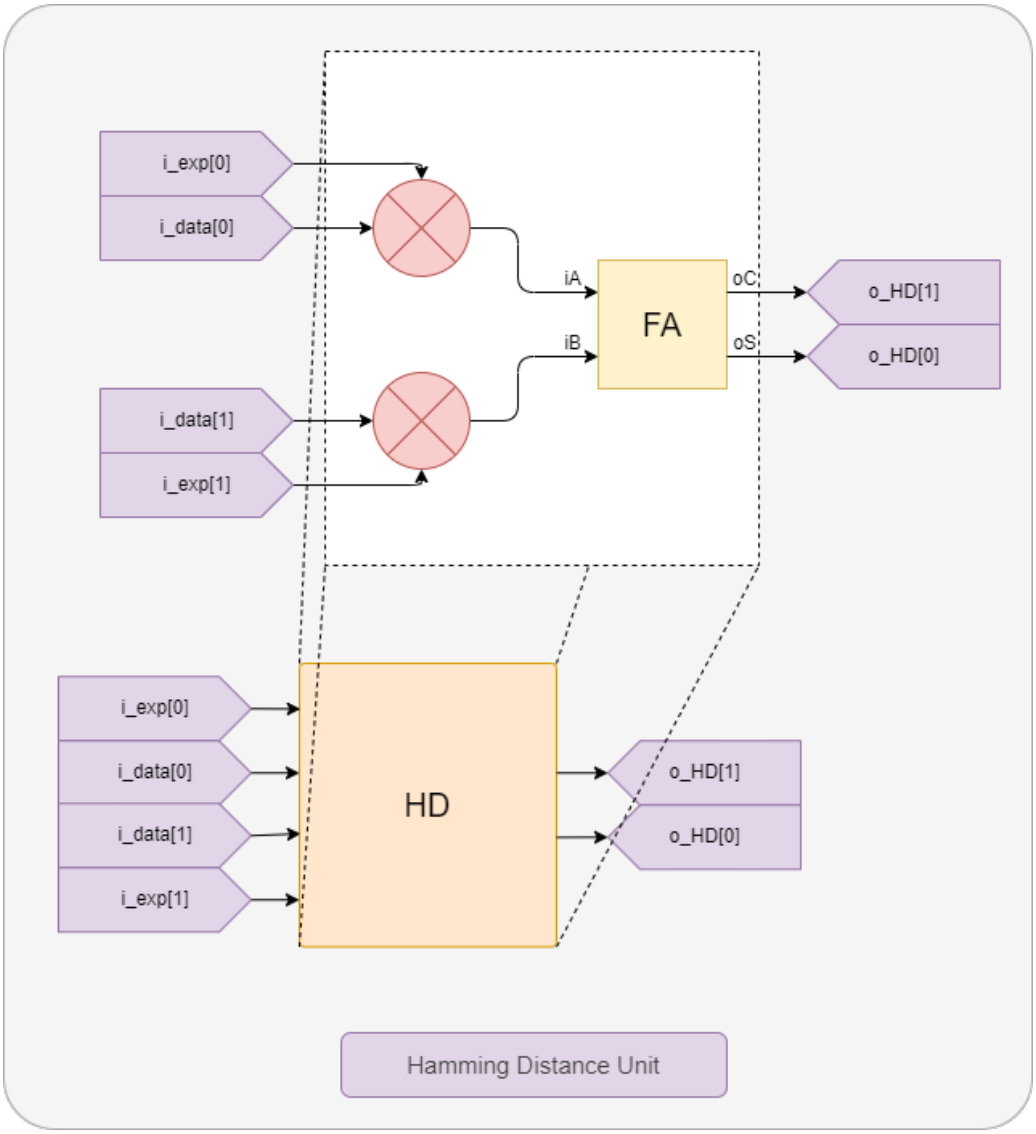
```
The number of input M = 1
The number of output N = 2
Enter the Generator Polynomial Matrix (G[n][k]): = [(1 1 1) (1 0 1)]

Input data: 110110

------------- Ouput Conv Data -------------

Conv Data: 110101000101

------------- Viterbi Decoder -------------
Viterbi data: 110111000111 - BER = 2
Viterbi Data: 110110
```

Listing 3: Mô phỏng việc mã hóa và giải mã với 0bit sai.

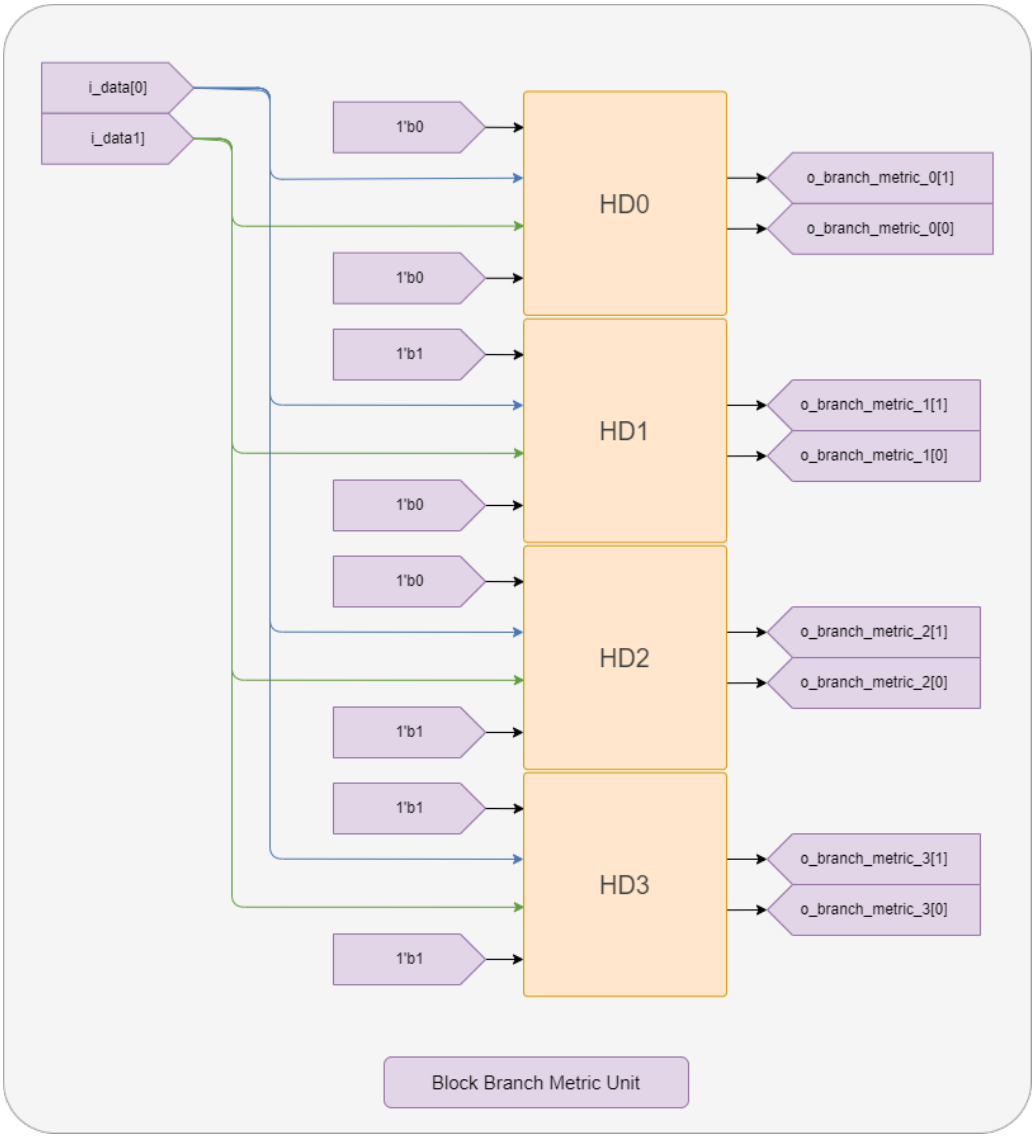
CHƯƠNG 3. MÔ PHỎNG THUẬT TOÁN GIẢI MÃ VITERBI BẰNG SYSTEM VERILOG

3.1. Branch Metric Unit (BMU)

3.1.1. Block Diagram for BMU



Hình 10: Bộ tính toán khoảng cách Hamming.



Hình 11: Bộ BMU.

3.1.2. IO for BMU

Port	Size	Function
i_data	2	Data ngõ vào bộ VD.
o_BM_0	2	Giá trị Branch Metric sau khi tính toán với expected là 0x00
o_BM_1	2	Giá trị Branch Metric sau khi tính toán với expected là 0x01
o_BM_2	2	Giá trị Branch Metric sau khi tính toán với expected là 0x10
o_BM_3	2	Giá trị Branch Metric sau khi tính toán với expected là 0x11

Bảng 2: Bảng sơ đồ chân của bộ BMU.

3.1.3. Chức năng của BMU

Bộ BMU có chức tính toán Khoảng cách Hamming giữa giá trị đã ngõ vào (i_data) và các giá trị expected. Với bộ VD $(3, 1, 2)$ thì expected word sẽ là 0x00, 0x01, 0x10, 0x11.

i_data	o_BM_0	o_BM_1	o_BM_2	o_BM_3
00	00	01	01	10
01	01	00	10	01
10	00	10	00	01
11	10	01	01	00

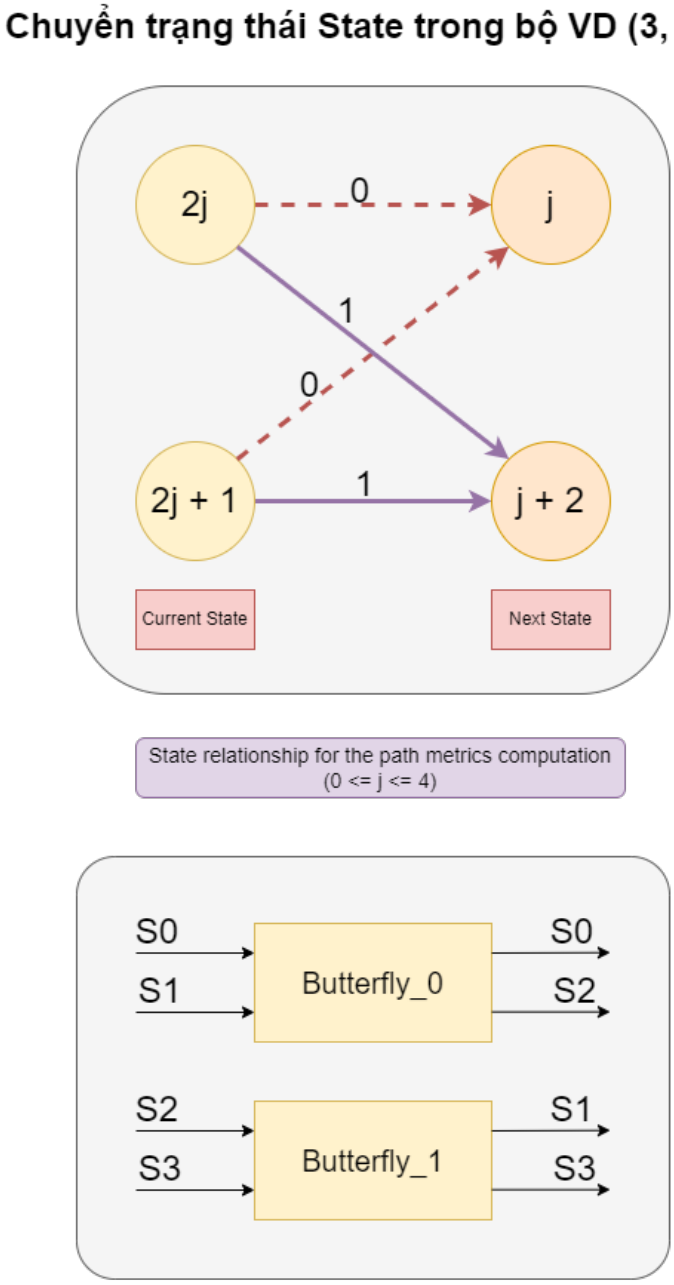
Bảng 3: Bẳng sự thật của bộ BMU.

```
TestCase 1: 0x00
| Time =                 10000   |
| w_idata = 00  |
| w_BM_0 = 00   | w_BM_1 = 01   | w_BM_2 = 01   | w_BM_3 = 10   |
-> PASS
--------------------------------------------------
TestCase 2: 0x01
| Time =                 20000   |
| w_idata = 01  |
| w_BM_0 = 01   | w_BM_1 = 00   | w_BM_2 = 10   | w_BM_3 = 01   |
-> PASS
--------------------------------------------------
TestCase 3: 0x10
| Time =                 30000   |
| w_idata = 10  |
| w_BM_0 = 01   | w_BM_1 = 10   | w_BM_2 = 00   | w_BM_3 = 01   |
-> PASS
--------------------------------------------------
TestCase 4: 0x11
| Time =                 40000   |
| w_idata = 11  |
| w_BM_0 = 10   | w_BM_1 = 01   | w_BM_2 = 01   | w_BM_3 = 00   |
-> PASS
--------------------------------------------------
```

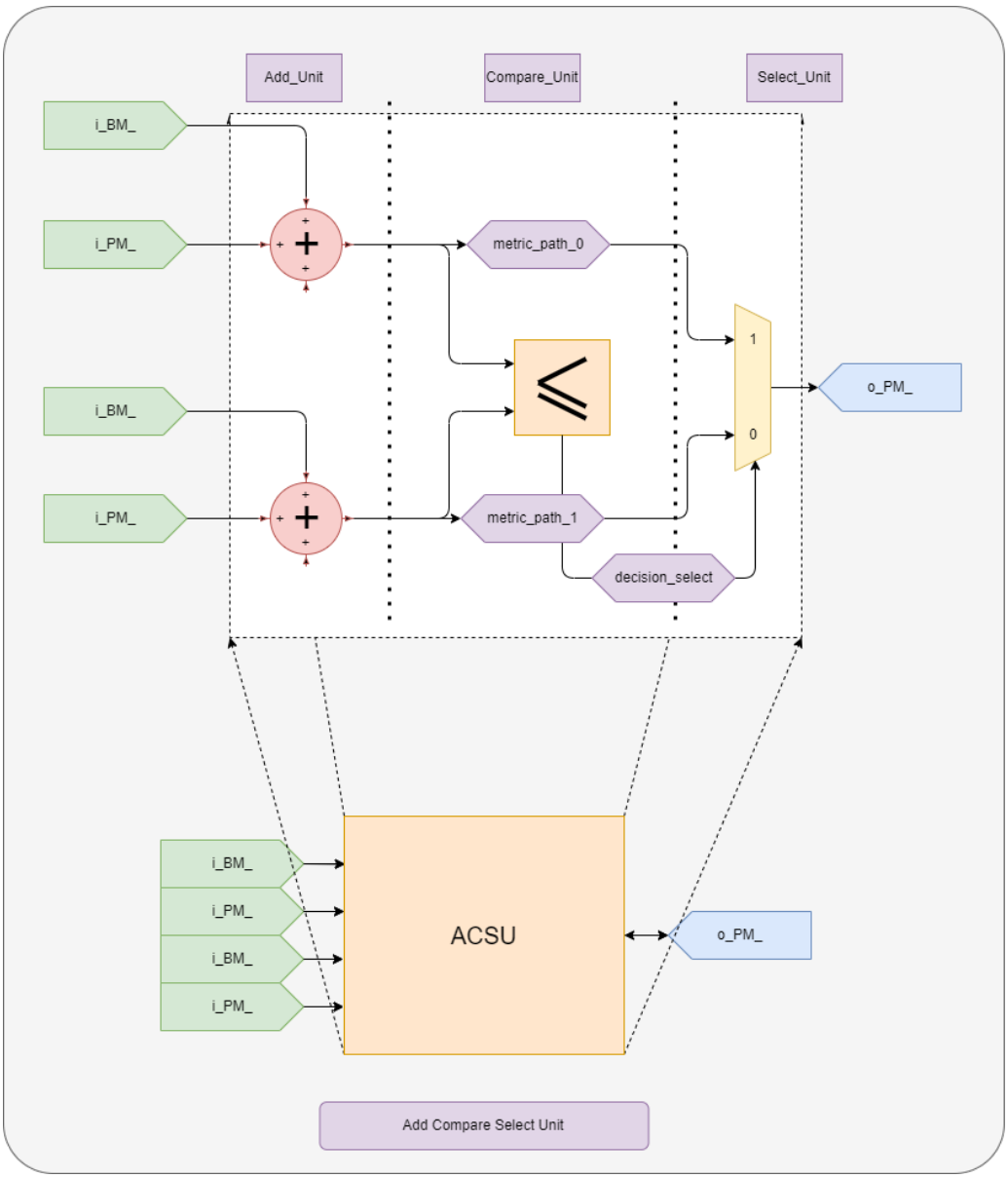
Listing 4: The Result of testing BMU.

3.2. Add Compare Select Unit (ACSU)

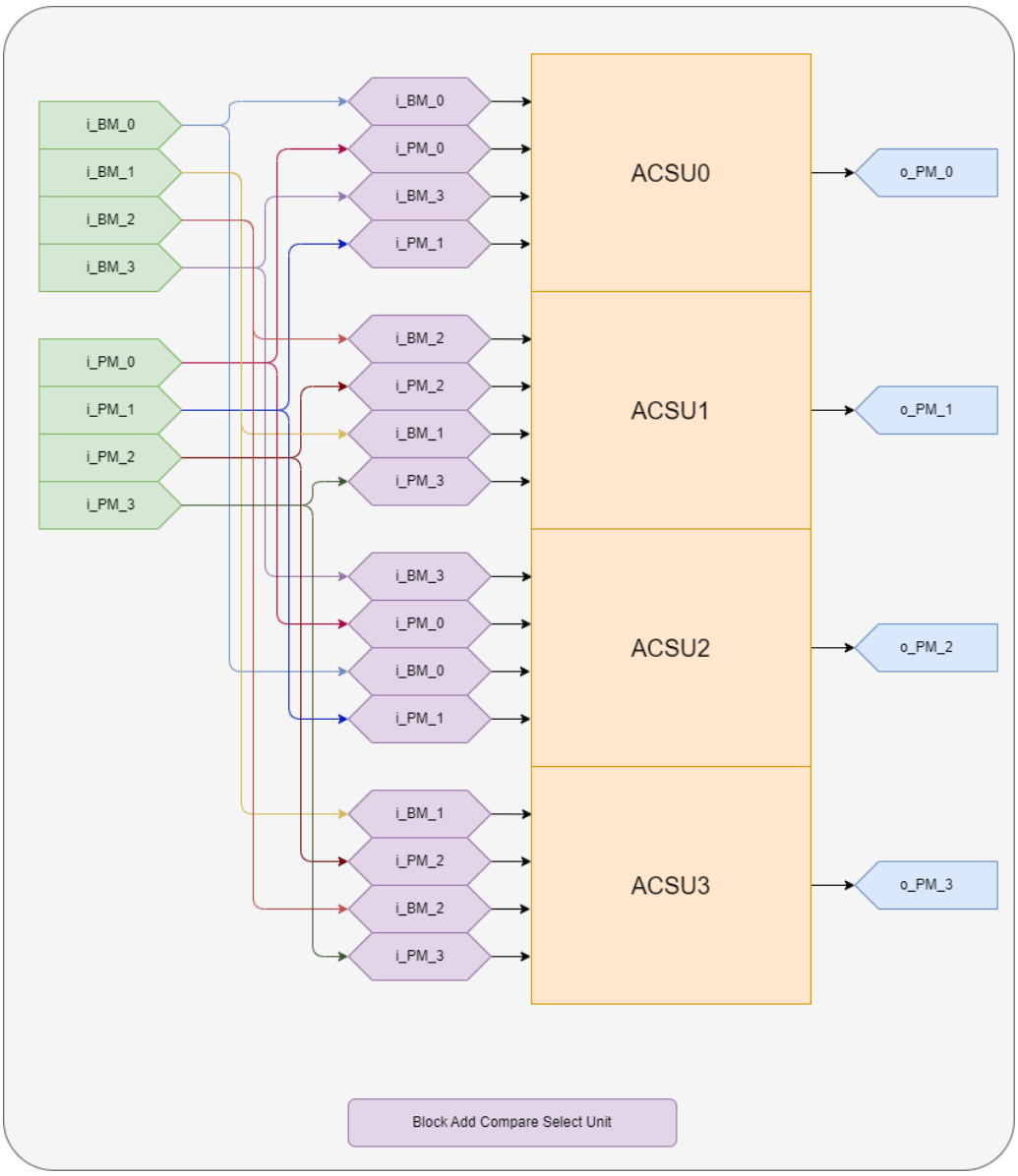
3.2.1. Block Diagram for ACSU



Hình 12: Cách chuyển trạng thái trong bộ giải mã Viterbi $(3, 1, 2)$.



Hình 13: Bộ Add Compare Select.



Hình 14: Bộ ACSU hoàn chỉnh.

3.2.2. IO for ACSU

Port	Size	Function
i_BM_0	2	Ngõ vào số liệu nhánh 0 (Branch Metric)
i_BM_1	2	Ngõ vào số liệu nhánh 1 (Branch Metric)
i_BM_2	2	Ngõ vào số liệu nhánh 2 (Branch Metric)
i_BM_3	2	Ngõ vào số liệu nhánh 3 (Branch Metric)
i_PM_0	2	Ngõ vào số liệu đường 0 (Path Metric)
i_PM_1	2	Ngõ vào số liệu đường 1 (Path Metric)
i_PM_2	2	Ngõ vào số liệu đường 2 (Path Metric)
i_PM_3	2	Ngõ vào số liệu đường 3 (Path Metric)
o_PM_0	2	Ngõ ra số liệu đường 0 (Path Metric)
o_PM_1	2	Ngõ ra số liệu đường 1 (Path Metric)
o_PM_2	2	Ngõ ra số liệu đường 2 (Path Metric)
o_PM_3	2	Ngõ ra số liệu đường 3 (Path Metric)

Bảng 4: Bảng sơ đồ chân của bộ ACSU.

3.2.3. Chức năng của ACSU

Bộ ACSU thực các chức năng chính sau đây:

- Add: thực hiện chức năng cộng các giá trị Branch Metric ở trạng thái trước đó và giá trị Path Metric hiện tại ở state đó.

  Bộ Add Unit trên được thực theo bảng sự thật sau:

i_BM	i_PM	o_PM
00	00	00
00	01	01
00	10	10
00	11	11
01	00	01
01	01	10
01	10	11
01	11	11
10	00	10
10	01	11
10	10	11
10	11	11
11	00	11
11	01	11
11	10	11
11	11	11

Bảng 5: Bảng sự thật của Add Unit.

```
Starting Add_unit testbench...
============================
```

```
TestCase 1  | Inputs: 00 + 00   | Output: 00  | Expected: 00  |
-> PASS
TestCase 2  | Inputs: 00 + 01   | Output: 01  | Expected: 01  |
-> PASS
TestCase 3  | Inputs: 00 + 10   | Output: 10  | Expected: 10  |
-> PASS
TestCase 4  | Inputs: 00 + 11   | Output: 11  | Expected: 11  |
-> PASS
TestCase 5  | Inputs: 01 + 00   | Output: 01  | Expected: 01  |
-> PASS
TestCase 6  | Inputs: 01 + 01   | Output: 10  | Expected: 10  |
-> PASS
TestCase 7  | Inputs: 01 + 10   | Output: 11  | Expected: 11  |
-> PASS
TestCase 8  | Inputs: 01 + 11   | Output: 11  | Expected: 11  |
-> PASS
TestCase 9  | Inputs: 10 + 00   | Output: 10  | Expected: 10  |
-> PASS
TestCase 10   | Inputs: 10 + 01   | Output: 11  | Expected: 11  |
-> PASS
TestCase 11   | Inputs: 10 + 10   | Output: 11  | Expected: 11  |
-> PASS
TestCase 12   | Inputs: 10 + 11   | Output: 11  | Expected: 11  |
-> PASS
TestCase 13   | Inputs: 11 + 00   | Output: 11  | Expected: 11  |
-> PASS
TestCase 14   | Inputs: 11 + 01   | Output: 11  | Expected: 11  |
-> PASS
TestCase 15   | Inputs: 11 + 10   | Output: 11  | Expected: 11  |
-> PASS
TestCase 16   | Inputs: 11 + 11   | Output: 11  | Expected: 11  |
-> PASS

Test Summary:
=============
Total tests : 16
Passed      : 16
Failed      : 0
Pass rate   : 100.00%
- tb_addunit.sv:173: Verilog $finish
```

Listing 5: The Result of testing Add Unit.

- Compare: thực hiện việc so sánh hai giá trị Path Metric hiện tại để kiểm tra xem có giá trị nào hơn để đến bước quyết định.

  Bộ Compare Unit trên được thực hiện theo bảng sự thật sau:

i_metric_path_0	i_metric_path_1	o_compare_less
00	00	1
00	01	1
00	10	1
00	11	1
01	00	0
01	01	1
01	10	1
01	11	1
10	00	0
10	01	0
10	10	1
10	11	1
11	00	0
11	01	0
11	10	0
11	11	1

Bảng 6: Bảng sự thật của bộ Compare Unit.

```
Starting Compare_unit testbench...
================================
TestCase 1  | Inputs: A=00   B=00   | Output: 1   | Expected: 1   |
-> PASS
TestCase 2  | Inputs: A=00   B=01   | Output: 1   | Expected: 1   |
-> PASS
TestCase 3  | Inputs: A=00   B=10   | Output: 1   | Expected: 1   |
-> PASS
TestCase 4  | Inputs: A=00   B=11   | Output: 1   | Expected: 1   |
-> PASS
TestCase 5  | Inputs: A=01   B=00   | Output: 0   | Expected: 0   |
-> PASS
TestCase 6  | Inputs: A=01   B=01   | Output: 1   | Expected: 1   |
-> PASS
TestCase 7  | Inputs: A=01   B=10   | Output: 1   | Expected: 1   |
-> PASS
TestCase 8  | Inputs: A=01   B=11   | Output: 1   | Expected: 1   |
-> PASS
TestCase 9  | Inputs: A=10   B=00   | Output: 0   | Expected: 0   |
-> PASS
TestCase 10   | Inputs: A=10   B=01   | Output: 0   | Expected: 0
|
-> PASS
TestCase 11   | Inputs: A=10   B=10   | Output: 1   | Expected: 1
|
-> PASS
TestCase 12   | Inputs: A=10   B=11   | Output: 1   | Expected: 1
|
-> PASS
```

```
 TestCase 13   | Inputs: A=11   B=00   | Output: 0   | Expected: 0
|
 -> PASS
 TestCase 14   | Inputs: A=11   B=01   | Output: 0   | Expected: 0
|
 -> PASS
 TestCase 15   | Inputs: A=11   B=10   | Output: 0   | Expected: 0
|
 -> PASS
 TestCase 16   | Inputs: A=11   B=11   | Output: 1   | Expected: 1
|
 -> PASS

 Test Summary:
 =============
 Total tests : 16
 Passed      : 16
 Failed      : 0
 Pass rate   : 100.00%
 - tb_compare.sv:151: Verilog $finish
```

Listing 6: The Result of testing Compare Unit.

- Select: thực hiện việc quyết định quá trị ngõ ra của giá trị Path Metric với tín hiệu o_compare_less từ bộ Compare Unit, việc chọn dữ liệu ngõ ra là thực hiện phép tính nhỏ hơn hoặc bằng ($\leq$).

i_metric_path_0	**i_metric_path_1**	**i_compare_less**	**o_PM**
Data_0	x	0	Data_0
x	Data_1	1	Data_1

Bảng 7: Bảng sự thật của bộ Select Unit.

Bộ ACSU cần kết hợp thêm bộ PMU (Path Metric Unit) có chức năng lưu giá trị Path Metric của các state tại thời gian trước đó, để có thể giúp.

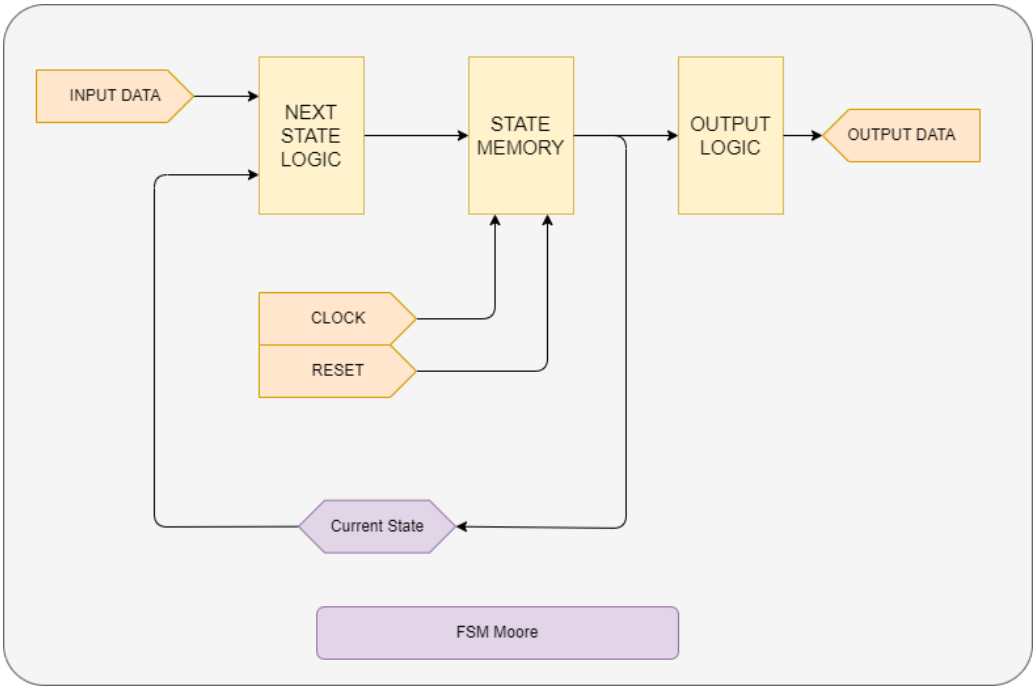
```
Time: 110000   | i_data = 00    | i_valid = 0    |
| i_BM_0: 00   | i_BM_1: 01   | i_BM_2: 01   | i_BM_3: 10   |
| i_PM_0: 00   | i_PM_1: 11   | i_PM_2: 11   | i_PM_3: 11   |
| o_PM_0: 00   | o_PM_1: 11   | o_PM_2: 10   | o_PM_3: 11   |
| t_PM_0: 00   | t_PM_1: 11   | t_PM_2: 10   | t_PM_3: 11   |
-> PASS
====================================
Starting ACSU and PMU testbench...
Time: 130000   | i_data = 00    | i_valid = 1    |
| i_BM_0: 00   | i_BM_1: 01   | i_BM_2: 01   | i_BM_3: 10   |
| i_PM_0: 00   | i_PM_1: 11   | i_PM_2: 11   | i_PM_3: 11   |
| o_PM_0: 00   | o_PM_1: 11   | o_PM_2: 10   | o_PM_3: 11   |
```

```
| t_PM_0: 00  | t_PM_1: 11  | t_PM_2: 10  | t_PM_3: 11  |
-> PASS
====================================
====================================
Time: 150000  | i_data = 00   | i_valid = 1   |
| i_BM_0: 00  | i_BM_1: 01  | i_BM_2: 01  | i_BM_3: 10  |
| i_PM_0: 00  | i_PM_1: 11  | i_PM_2: 10  | i_PM_3: 11  |
| o_PM_0: 00  | o_PM_1: 11  | o_PM_2: 10  | o_PM_3: 11  |
| t_PM_0: 00  | t_PM_1: 11  | t_PM_2: 10  | t_PM_3: 11  |
-> PASS
====================================
====================================
Time: 170000  | i_data = 01   | i_valid = 1   |
| i_BM_0: 01  | i_BM_1: 00  | i_BM_2: 10  | i_BM_3: 01  |
| i_PM_0: 00  | i_PM_1: 11  | i_PM_2: 10  | i_PM_3: 11  |
| o_PM_0: 10  | o_PM_1: 10  | o_PM_2: 10  | o_PM_3: 01  |
| t_PM_0: 10  | t_PM_1: 10  | t_PM_2: 10  | t_PM_3: 01  |
-> PASS
====================================
====================================
Time: 190000  | i_data = 10   | i_valid = 1   |
| i_BM_0: 01  | i_BM_1: 10  | i_BM_2: 00  | i_BM_3: 01  |
| i_PM_0: 01  | i_PM_1: 11  | i_PM_2: 01  | i_PM_3: 10  |
| o_PM_0: 10  | o_PM_1: 10  | o_PM_2: 10  | o_PM_3: 10  |
| t_PM_0: 10  | t_PM_1: 10  | t_PM_2: 10  | t_PM_3: 10  |
-> PASS
====================================
====================================
Time: 210000  | i_data = 11   | i_valid = 1   |
| i_BM_0: 10  | i_BM_1: 01  | i_BM_2: 01  | i_BM_3: 00  |
| i_PM_0: 10  | i_PM_1: 01  | i_PM_2: 10  | i_PM_3: 10  |
| o_PM_0: 11  | o_PM_1: 11  | o_PM_2: 01  | o_PM_3: 11  |
| t_PM_0: 11  | t_PM_1: 11  | t_PM_2: 01  | t_PM_3: 11  |
-> PASS
====================================
====================================
Simulation finished.
====================================
- tb_acsu.sv:359: Verilog $finish
```

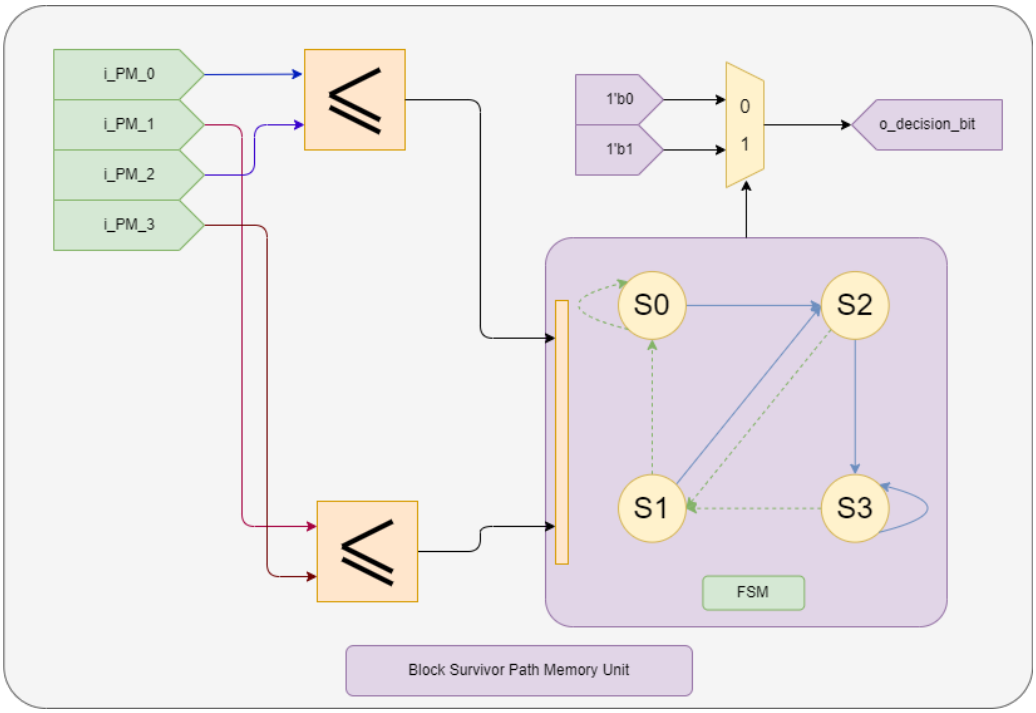
Listing 7: The Result of testing ACSU.

3.3. Survivor Path Memory Unit (SPMU)

3.3.1. Block Diagram for SPMU



Hình 15: Máy trạng thái viết ở dạng Moore.



Hình 16: Bộ SPMU.

3.3.2. IO for SPMU

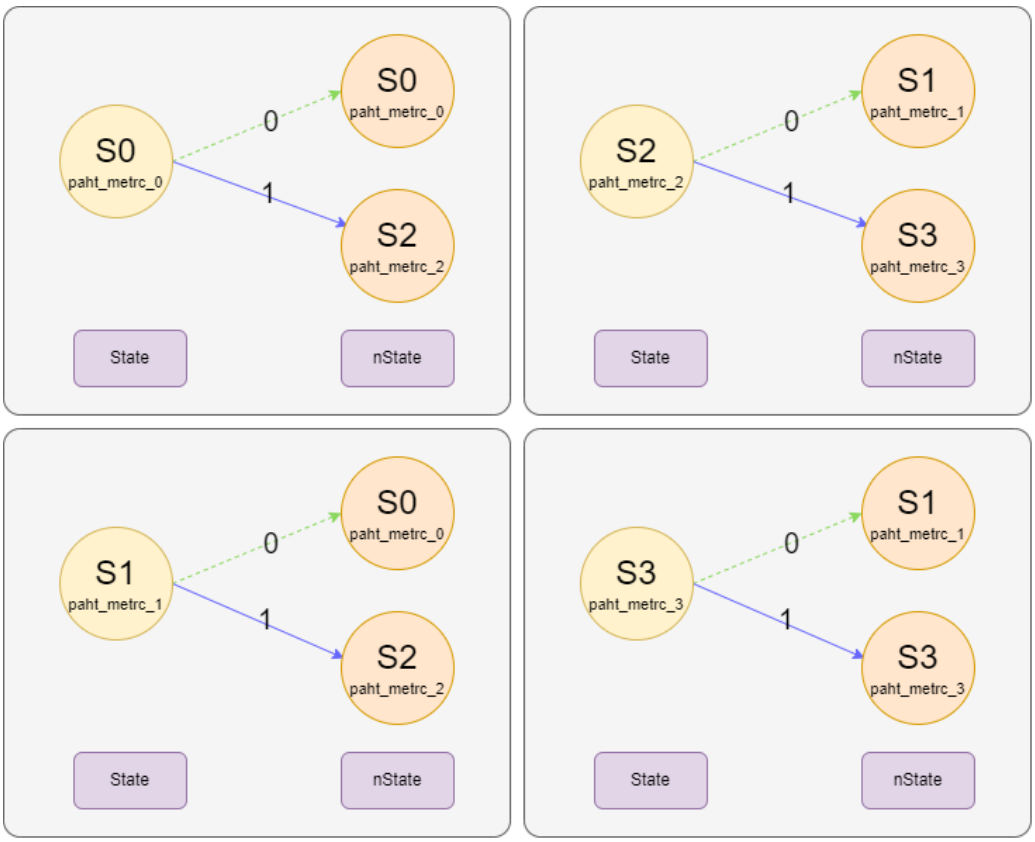
Port	Size	Function
i_clk	1	Xung clock hệ thống
i_rst_n	1	Tín hiệu reset tích cực mức thấp
i_valid	1	Tín hiệu valid cho dữ liệu vào
i_PM_0	2	Giá trị Path Metric 0 đầu vào
i_PM_1	2	Giá trị Path Metric 1 đầu vào
i_PM_2	2	Giá trị Path Metric 2 đầu vào
i_PM_3	2	Giá trị Path Metric 3 đầu vào
o_decision	1	Quyết định đường đi tối ưu (0-3)
o_valid	1	Tín hiệu valid cho dữ liệu ra

Bảng 8: Bảng sơ đồ chân của bộ SPMU (Survivor Path Memory Unit).

3.3.3. Chức năng của bộ SMPU

Bộ SPMU có nhiệm vụ quyết định bit ngõ ra dữ vào bit đầu vào. Có thể coi bộ SPMU là một nhân xử lý chính trong bộ VD giúp quyết định bit ngõ ra là gì. Bộ SPMU được viết dự vào máy trạng thái với các trạng chính trong trellis thể hiện ở hình 4. Như vậy, ta có thể thấy được ngõ ra được quyết định bởi trạng thái hiện tại và trạng thái tiếp chứ không phụ thuộc vào điều kiện ngõ vào, nên ta sử dụng mô hình moore như hình 15, giúp cho dữ liệu ngõ được ổn định và chính xác hơn.

Sơ đồ quyết định bit ngõ ra dữ vào trạng thái hiện và trạng ngõ ra như sau:



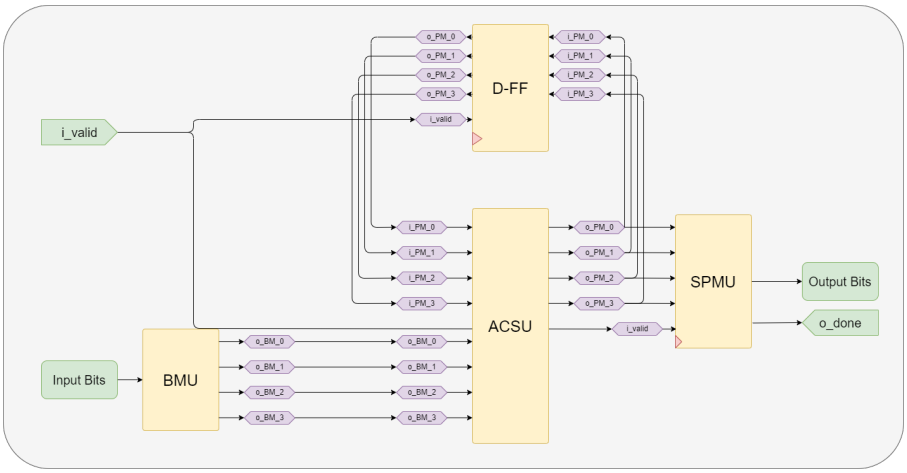
Hình 17: Sơ đồ quyết định bit ngõ ra của bộ SPMU.

```
Time =  5000 | i_rst_n = 1  | i_valid = 0   | o_decision = 0   |
PM_0 = 00    | PM_1 = 00   | PM_2 = 00    | PM_3 = 00         |
========================================
TestCase 1: S0 -> S0
Time =  16000 | i_rst_n = 1   | i_valid = 1   | o_decision = 0 |
PM_0 = 00     | PM_1 = 01    | PM_2 = 10    | PM_3 = 11         |
-> PASS
========================================
TestCase 2: S0 -> S2
Time = 26000 | i_rst_n = 1  | i_valid = 1   | o_decision = 1   |
PM_0 = 11    | PM_1 = 10   | PM_2 = 01    | PM_3 = 00         |
-> PASS
========================================
TestCase 3: S2 -> S1
Time = 36000 | i_rst_n = 1  | i_valid = 1   | o_decision = 0     |
PM_0 = 10    | PM_1 = 00   | PM_2 = 00    | PM_3 = 01         |
-> PASS
========================================
TestCase 4: S1 -> S2
Time = 46000 | i_rst_n = 1  | i_valid = 1   | o_decision = 1   |
PM_0 = 11    | PM_1 = 00   | PM_2 = 01    | PM_3 = 10         |
-> PASS
========================================
TestCase 5: S2 -> S3
Time = 56000 | i_rst_n = 1  | i_valid = 1   | o_decision = 1   |
PM_0 = 01    | PM_1 = 10   | PM_2 = 11    | PM_3 = 00         |
-> PASS
========================================
TestCase 6: S3 -> S3
Time = 66000 | i_rst_n = 1  | i_valid = 1   | o_decision = 1   |
PM_0 = 11    | PM_1 = 10   | PM_2 = 01    | PM_3 = 00         |
-> PASS
========================================
TestCase 7: S3 -> S1
Time = 76000 | i_rst_n = 1  | i_valid = 1   | o_decision = 0   |
PM_0 = 11    | PM_1 = 00   | PM_2 = 01    | PM_3 = 10         |
-> PASS
========================================
TestCase 8: S1 -> S0
Time = 86000 | i_rst_n = 1  | i_valid = 1   | o_decision = 0  |
PM_0 = 00    | PM_1 = 01   | PM_2 = 01    | PM_3 = 10        |
-> PASS
========================================
Testbench completed successfully
==================================
- tb_spmu.sv:168: Verilog $finish
```

Listing 8: The result of testing SPMU

3.4. Viterbi Decoder Block

3.4.1. Block Diagram for Viterbi Decoder Block



Hình 18: Bộ Viterbi Decoder.

3.4.2. IO for Viterbi Decoder Block

Port	Size	Function
i_clk	1	Xung nhịp hệ thống
i_rst_n	1	Tín hiệu reset tích cực mức thấp (Active-low reset)
i_valid	1	Tín hiệu báo dữ liệu vào hợp lệ
i_data	2	Dữ liệu đầu vào cần giải mã
o_decision	1	Bit dữ liệu đã được giải mã
o_valid	1	Tín hiệu báo dữ liệu ra hợp lệ

Bảng 9: Bảng sơ đồ chân của bộ Viterbi Decoder Block.

```
Time: 0, o_decision: 0, o_valid: 0
TestCase 1:
Data input: 10101010
Data conv : 1110001000100010
Time: 50000, o_decision: 1, o_valid: 1
Time: 51000  | i_valid = 1  | i_data: 11 | o_decision = 1 | o_valid = 1 |
Time: 71000  | i_valid = 1  | i_data: 10 | o_decision = 0 | o_valid = 1 |
Time: 91000  | i_valid = 1  | i_data: 00 | o_decision = 1 | o_valid = 1 |
Time: 111000 | i_valid = 1  | i_data: 10 | o_decision = 0 | o_valid = 1 |
Time: 131000 | i_valid = 1  | i_data: 00 | o_decision = 1 | o_valid = 1 |
Time: 151000 | i_valid = 1  | i_data: 10 | o_decision = 0 | o_valid = 1 |
Time: 171000 | i_valid = 1  | i_data: 00 | o_decision = 1 | o_valid = 1 |
Time: 191000 | i_valid = 1  | i_data: 10 | o_decision = 0 | o_valid = 1 |
Time: 210000 | i_valid = 1  | i_data: 10 | o_decision = 0 | o_valid = 1 |
============================================================
Time: 210000, o_decision: 0, o_valid: 0
```

```
TestCase 2:
Data input: 00101001
Data conv : 0000111000101111
Time: 250000, o_decision: 0, o_valid: 1
Time: 271000 | i_valid = 1  | i_data: 00 | o_decision = 0 | o_valid = 1 |
Time: 291000 | i_valid = 1  | i_data: 00 | o_decision = 0 | o_valid = 1 |
Time: 311000 | i_valid = 1  | i_data: 11 | o_decision = 1 | o_valid = 1 |
Time: 331000 | i_valid = 1  | i_data: 10 | o_decision = 0 | o_valid = 1 |
Time: 351000 | i_valid = 1  | i_data: 00 | o_decision = 1 | o_valid = 1 |
Time: 371000 | i_valid = 1  | i_data: 10 | o_decision = 0 | o_valid = 1 |
Time: 391000 | i_valid = 1  | i_data: 11 | o_decision = 0 | o_valid = 1 |
Time: 411000 | i_valid = 1  | i_data: 11 | o_decision = 1 | o_valid = 1 |
Time: 430000 | i_valid = 1  | i_data: 11 | o_decision = 1 | o_valid = 1 |
============================================================
- tb_Viterbi_decoding.sv:119: Verilog $finish
```

CHƯƠNG 4. THỰC HIỆN TRÊN KIT DE2

CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1. Kết luận

Trong đồ án này, chúng tôi đã thiết kế thành công bộ giải mã Viterbi với thông số kỹ thuật:

- Độ dài ràng buộc K=3
- Tỷ lệ mã hóa (rate) 1/2
- Sử dụng thuật toán giải mã tối ưu dựa trên độ đo khoảng cách Hamming

Các kết quả đạt được bao gồm:

- Thiết kế hoàn chỉnh kiến trúc bộ giải mã gồm 3 khối chính: Khối tính toán BMU (Branch Metric Unit), ACSU (Add-Compare-Select Unit) và SPMU (Survivor Path Memory Unit)
- Mô phỏng kiểm chứng hoạt động chính xác với BER (Bit Error Rate) thấp hơn so với giải mã hard-decision thông thường
- Tối ưu hóa tài nguyên phần cứng khi triển khai trên FPGA

5.2. Hướng phát triển

Để nâng cao hiệu năng và ứng dụng thực tế của bộ giải mã, các hướng phát triển trong tương lai bao gồm:

- **Tích hợp giải mã soft-decision**: Sử dụng 3-4 bit lượng tử hóa để cải thiện khoảng 2dB hiệu năng so với giải mã hard-decision
- **Tăng độ dài ràng buộc (K=7 hoặc K=9)**: Đạt hiệu năng giải mã tốt hơn nhưng đòi hỏi tài nguyên phần cứng lớn hơn
- **Ứng dụng cho chuẩn không dây**: Triển khai cho các chuẩn thông tin di động (4G/5G) và vệ tinh
- **Tối ưu kiến trúc pipeline**: Nâng cao tốc độ xử lý phù hợp cho các ứng dụng tốc độ cao
- **Tích hợp với các hệ thống mã hóa khác**: Kết hợp với mã hóa Reed-Solomon trong hệ thống mã chập xếp chồng

A Phụ lục A: Dữ liệu bổ sung

B Phụ lục B: Mã nguồn

This my project.

Tài liệu

[1] G. C. Clark and J. B. Cain, *Error-Correction Coding for Digital Communications*. Applications of Communications Theory, Springer New York, 1981.