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

–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
2210780

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

Mục lục

Chương 1. Cơ sở lý thuyết **3**
1.1. Mã hóa kênh truyền trong lĩnh vực truyền thông thông tin 3
1.2. Mã hóa sử dụng Mã chập (Convolutional Encoder) 4
1.2.1. Ma trận sinh (Generator matrix) 5
1.2.2. Sơ đồ lưới (Trellis Diagram) 5
1.3. Giải mã kênh truyền trong lĩnh vực truyền thông thông tin 7
1.4. Giải mã kênh truyền sửa dụng thuật toán Viterbi 7

Chương 2. Mô phỏng thuật toán giải mã Viterbi bằng C **10**
2.1. Khối nhận chuỗi Bits đầu vào 11
2.2. Bộ mã tích chập 11
2.3. Bộ giải mã Viterbi 13
2.4. Kết quả 13

Chương 3. Mô phỏng thuật toán giải mã Viterbi bằng System Verilog **16**
3.1. Branch Metric Unit (BMU) 16
3.2. Add Compare Select Unit (ACSU) 20
3.3. Survivor Path Memory Unit (SPMU) 24

Danh sách hình vẽ

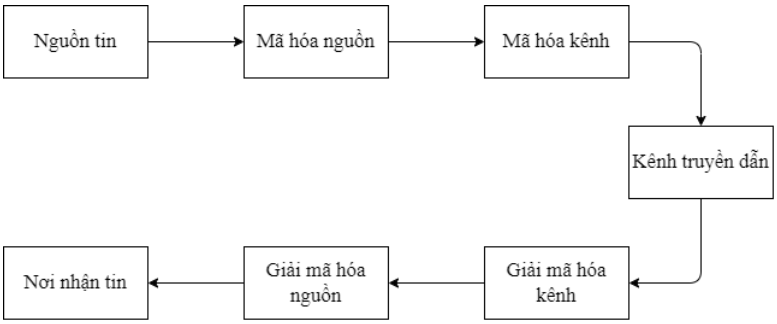
1 Mô hình hệ thống truyền thông số. 3
2 Cấu trúc tổng quát của một bộ mã hóa chập 4
3 Bộ mã hóa $(3, 1, 2)$. 5
4 Sơ đồ lưới giữa hai đơn vị thời gian của bộ mã hóa $(3, 1, 2)$. 6
5 Sơ đồ khối hệ thống. 10
6 Lưu đồ mô phỏng. 11
7 Lưu đồ giải thuật của Bộ mã Tích chập. 12
8 Lưu đồ giải thuật của Bộ giải mã Viterbi. 13
9 Bộ tính toán khoảng cách Hamming. 16
10 Bộ BMU. 17
11 Cách chuyển trạng thái trong bộ giải mã Viterbi $(3, 1, 2)$. 20
12 Bộ Add Compare Select. 21
13 Bộ ACSU hoàn chỉnh. 22
14 Máy trạng thái viết ở dạng Moore. 24
15 Bộ SPMU. 24

Danh sách bảng

1 Bảng chuyển trạng thái của mã chập $(3, 1, 2)$. 6

Chương 1. Cơ 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.

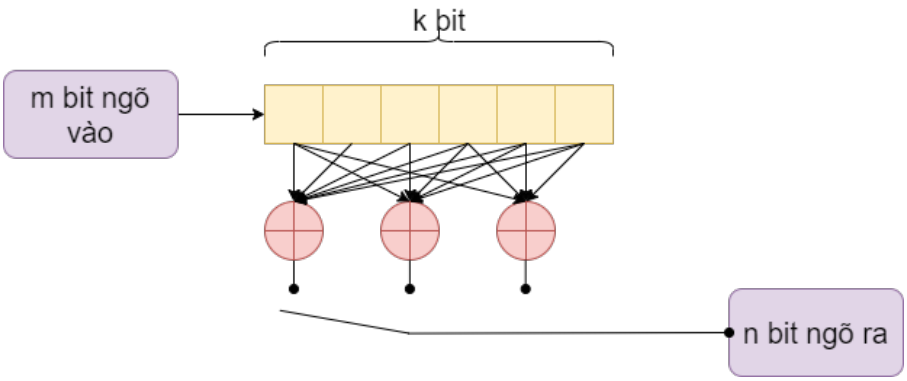
Việc thêm khối mã hóa kênh vào hệ thống truyền thông số là do thường tín hiệu trong truyền thông thực tế sẽ thường xảy ra các lỗi như: Lỗi ngẫu nhiên (Random Errors) do nhiễu trắng Gaussian (AWGN), Lỗi theo cụm (Burst Errors) do fading đa đường hoặc mất gói dữ liệu trong truyền thông không dây. Vì vậy, 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 :

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

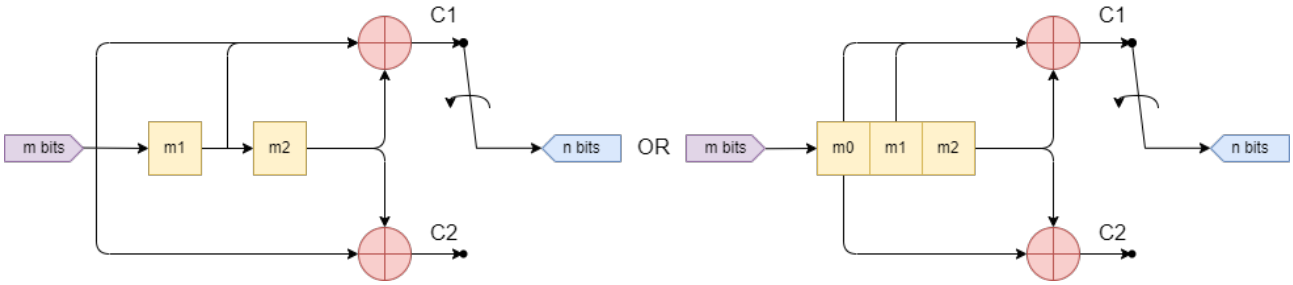
Trong đó:

- $g_{i,j}$: 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.

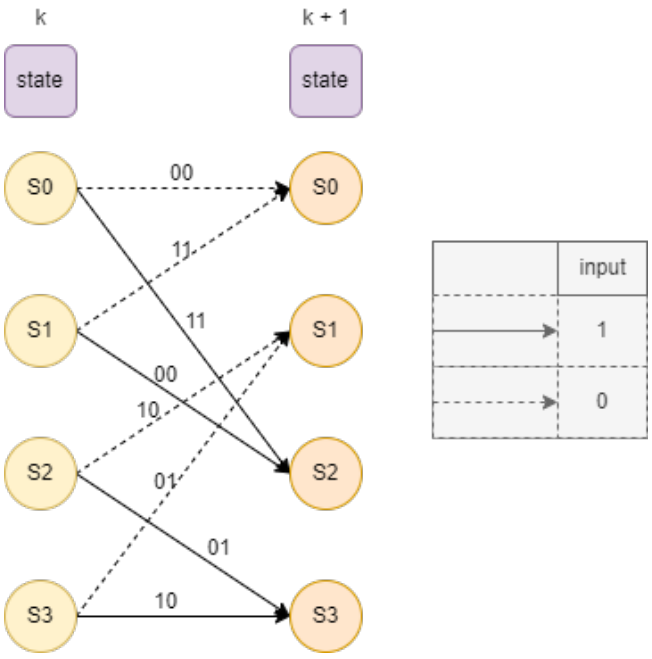
Để làm rõ hơn việc sử dụng sơ đồ trạng thái trong việc biểu diễn mã chập, ta sử dụng một bộ mã hóa chập ở hình 3. Với $K = 3$ ta có tương ứng với 4 trạng thái là: `S0(00)`, `S1(01)`, `S2(10)`, `S3(11)`, và trạng thái chuyển tiếp có thể được tạo từ trạng thái này sang trạng thái khác, quá trình chuyển 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 $(3, 1, 2)$.

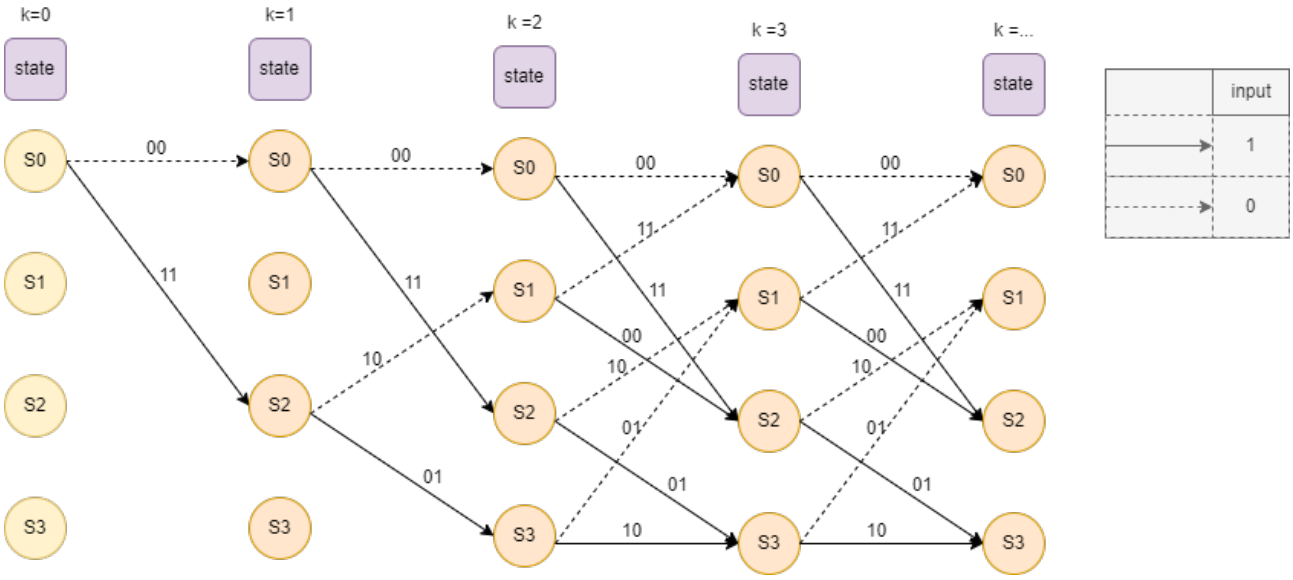
Sơ đồ lưới là một mở rộng của sơ đồ trạng thái theo thời gian, biểu diễn tất cả các đường đi có thể của mã chập qua các bước thời gian. Sơ đồ lưới bao gồm các nút biểu diễn trạng thái tại mỗi bước thời gian. Các cung biểu diễn chuyển tiếp giữa các trạng thái, được gán nhãn bởi bit đầu vào và bit đầu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

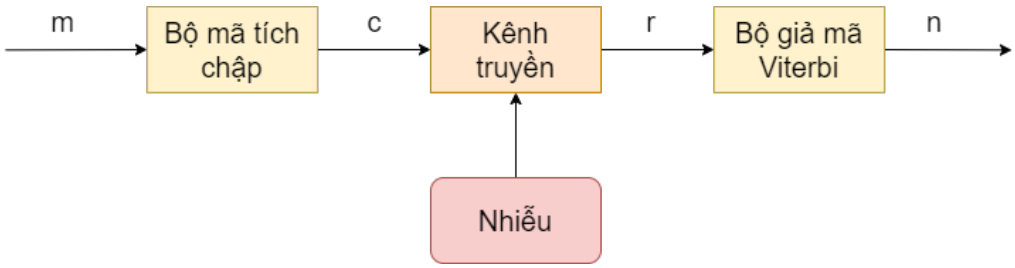
Phương trình cập nhật metric:

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

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

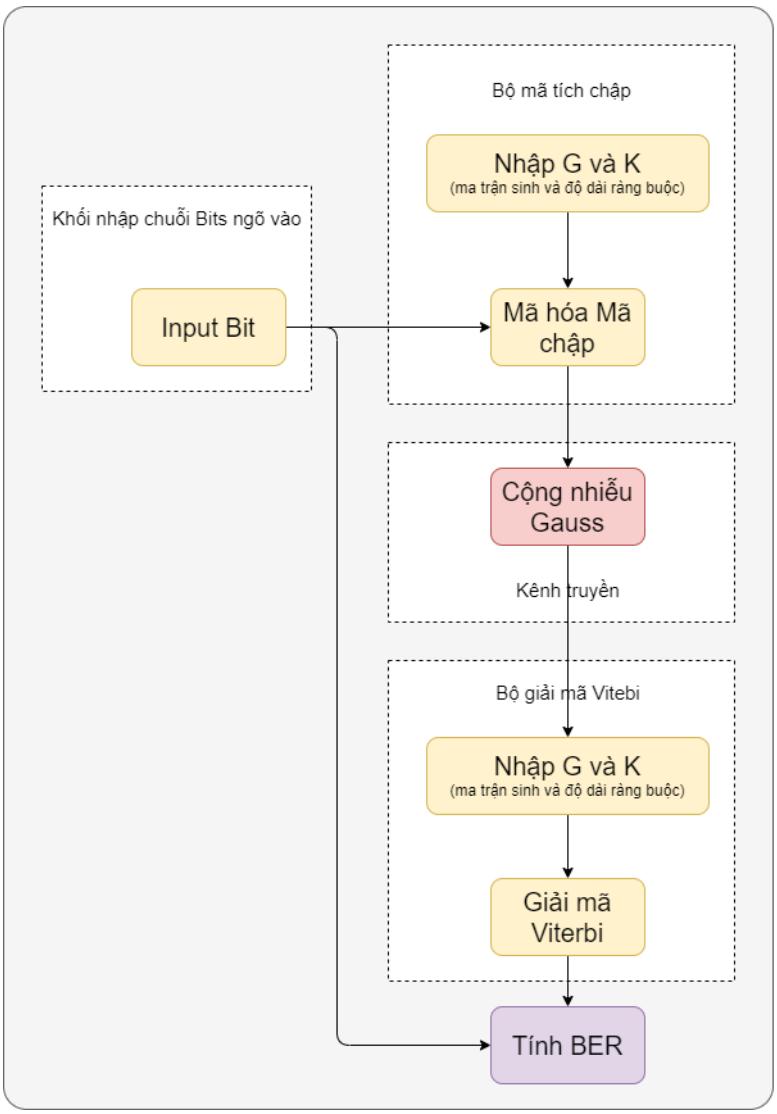
Chương 2. Mô phỏng thuật toán giải mã Viterbi bằng C



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

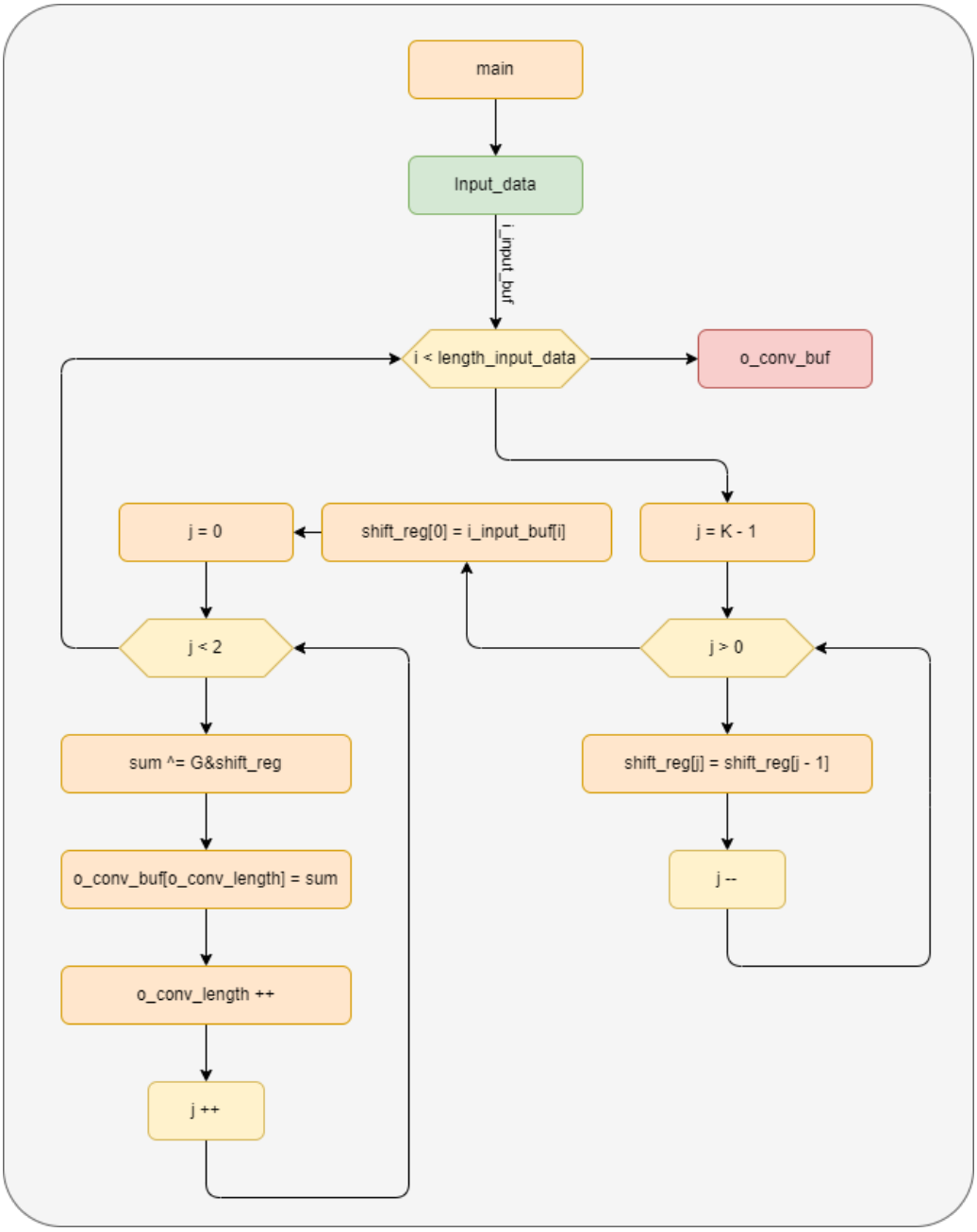
Cho phép người dùng nhập giá trị chuỗi bit đầu vào ở dạng chuỗi (`char`) và chuỗi chuỗi vừa nhận thành mảng một chuỗi chứa các giá trị số thực (`int`).

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:

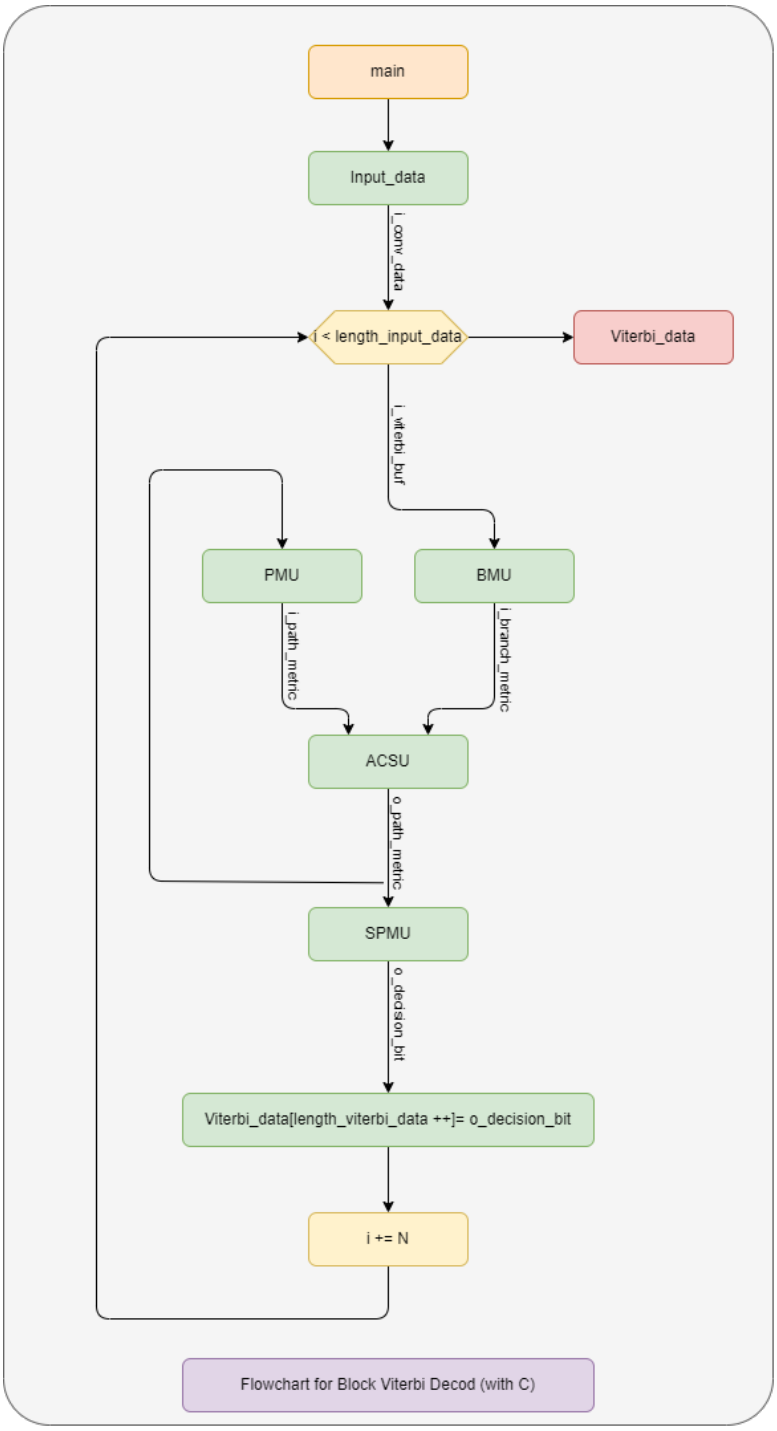
$$G = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}, K = 3$$

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



Hình 7: Lưu đồ giải thuật của Bộ mã Tích chập.

2.3. Bộ giải mã Viterbi



Hình 8: Lưu đồ giải thuật của Bộ giải mã Viterbi.

2.4. Kết quả

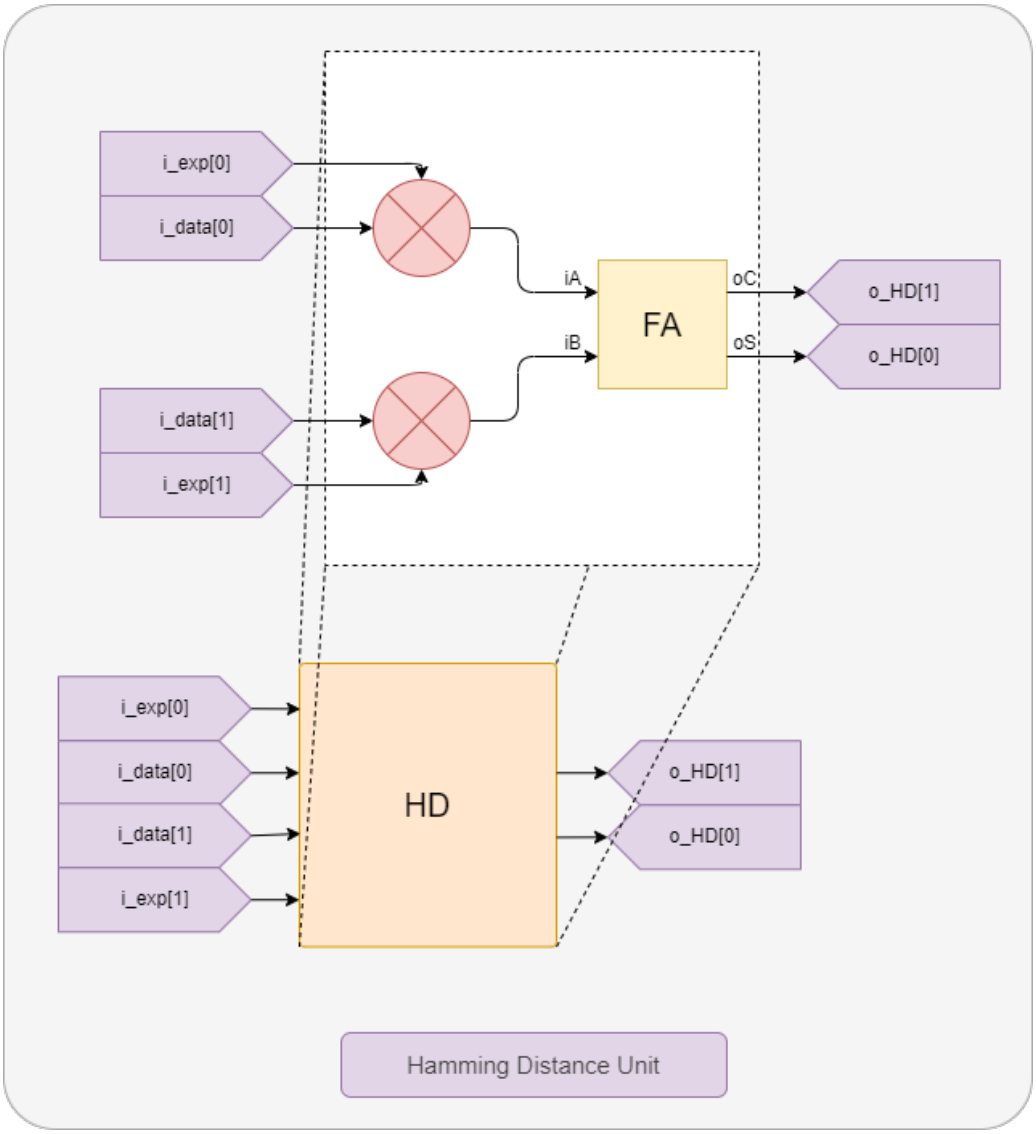
```
------------------- Input Data -------------------
Input data: 110110
The constraint length K = 3
The number of input M = 1
The number of output N = 2
```

```
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

Input Viterbi: 11
BM0 = 2, BM1 = 1, BM2 = 1, BM3 = 0
iPM0 = 0, iPM1 = 3, iPM2 = 3, iPM3 = 3
oPM0 = 2, oPM1 = 4, oPM2 = 0, oPM3 = 4
State = 1
Nex State = 3
Decision Bit = 1
Input Viterbi: 10
BM0 = 1, BM1 = 0, BM2 = 2, BM3 = 1
iPM0 = 2, iPM1 = 4, iPM2 = 0, iPM3 = 4
oPM0 = 3, oPM1 = 2, oPM2 = 3, oPM3 = 0
State = 3
Nex State = 4
Decision Bit = 1
Input Viterbi: 10
BM0 = 1, BM1 = 0, BM2 = 2, BM3 = 1
iPM0 = 3, iPM1 = 2, iPM2 = 3, iPM3 = 0
oPM0 = 3, oPM1 = 0, oPM2 = 3, oPM3 = 2
State = 4
Nex State = 2
Decision Bit = 0
Input Viterbi: 00
BM0 = 0, BM1 = 1, BM2 = 1, BM3 = 2
iPM0 = 3, iPM1 = 0, iPM2 = 3, iPM3 = 2
```

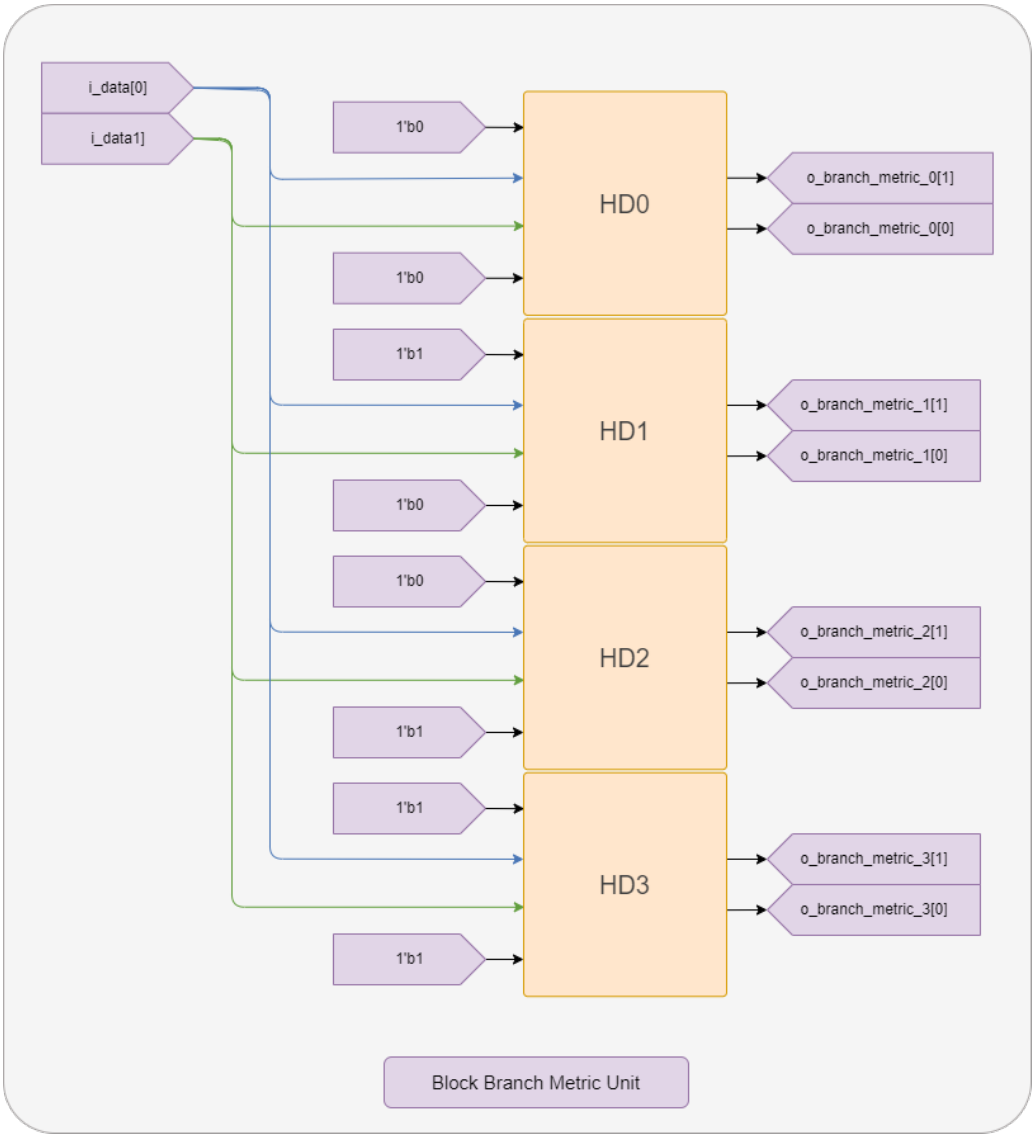
```
oPM0 = 2, oPM1 = 3, oPM2 = 0, oPM3 = 3
State = 2
Nex State = 3
Decision Bit = 1
Input Viterbi: 10
BM0 = 1, BM1 = 0, BM2 = 2, BM3 = 1
iPM0 = 2, iPM1 = 3, iPM2 = 0, iPM3 = 3
oPM0 = 3, oPM1 = 2, oPM2 = 3, oPM3 = 0
State = 3
Nex State = 4
Decision Bit = 1
Input Viterbi: 10
BM0 = 1, BM1 = 0, BM2 = 2, BM3 = 1
iPM0 = 3, iPM1 = 2, iPM2 = 3, iPM3 = 0
oPM0 = 3, oPM1 = 0, oPM2 = 3, oPM3 = 2
State = 4
Nex State = 2
Decision Bit = 0
Viterbi Data: 110110
```

Chương 3. Mô phỏng thuật toán giải mã Viterbi bằng System Verilog

3.1. Branch Metric Unit (BMU)



Hình 9: Bộ tính toán khoảng cách Hamming.



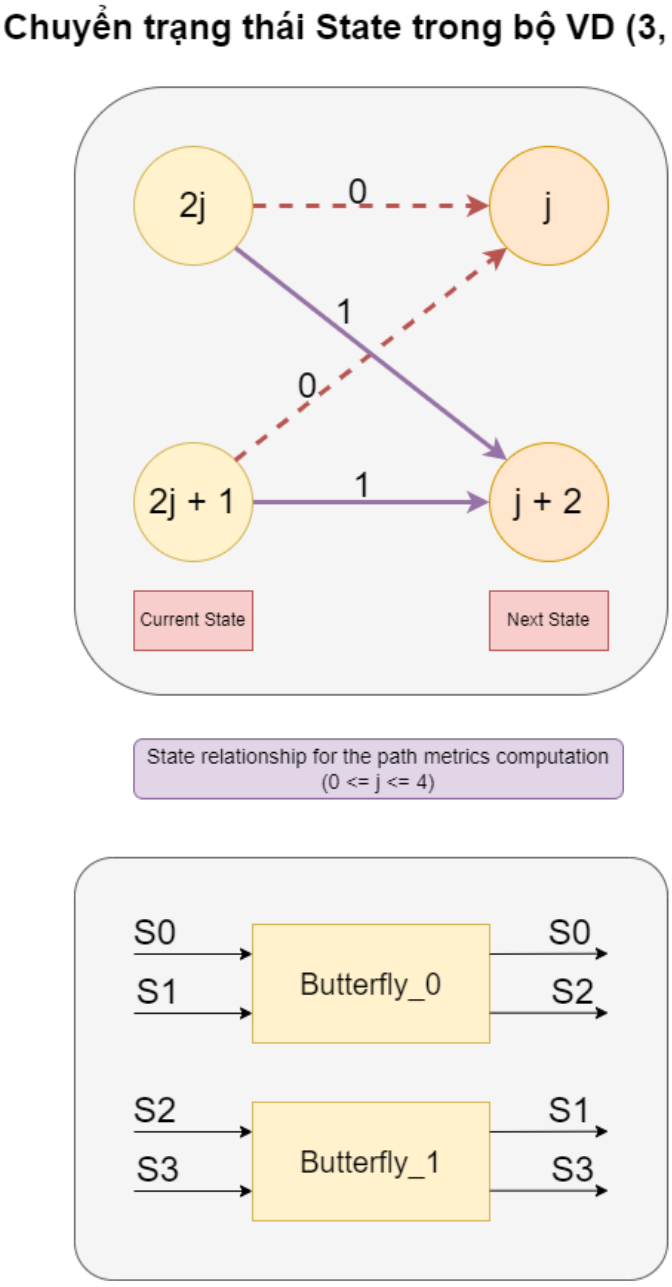
Hình 10: Bộ BMU.

```
Time: 0 | i_rst_n = 0 | i_data: 00 | o_BM_0: 00 | o_BM_1: 00 | o_BM_2: 00
 | o_BM_3: 00 | o_valid: 0
Time: 5000 | i_rst_n = 0 | i_data: 00 | o_BM_0: 00 | o_BM_1: 00 | o_BM_2:
 00 | o_BM_3: 00 | o_valid: 0
Time: 5000 | i_rst_n = 0 | i_data: 00 | o_BM_0: 00 | o_BM_1: 00 | o_BM_2:
 00 | o_BM_3: 00 | o_valid: 0
Time: 5000 | i_rst_n = 0 | i_data: 00 | o_BM_0: 00 | o_BM_1: 00 | o_BM_2:
 00 | o_BM_3: 00 | o_valid: 0
Time: 10000 | i_rst_n = 1 | i_data: 00 | o_BM_0: 00 | o_BM_1: 00 | o_BM_2:
  00 | o_BM_3: 00 | o_valid: 0
Time: 10000 | i_rst_n = 1 | i_data: 00 | o_BM_0: 00 | o_BM_1: 00 | o_BM_2:
  00 | o_BM_3: 00 | o_valid: 0
Time: 15000 | i_rst_n = 1 | i_data: 00 | o_BM_0: 00 | o_BM_1: 00 | o_BM_2:
  00 | o_BM_3: 00 | o_valid: 0
Time: 15000 | i_rst_n = 1 | i_data: 00 | o_BM_0: 00 | o_BM_1: 00 | o_BM_2:
  00 | o_BM_3: 00 | o_valid: 0
Time: 15000 | i_rst_n = 1 | i_data: 00 | o_BM_0: 00 | o_BM_1: 01 | o_BM_2:
```

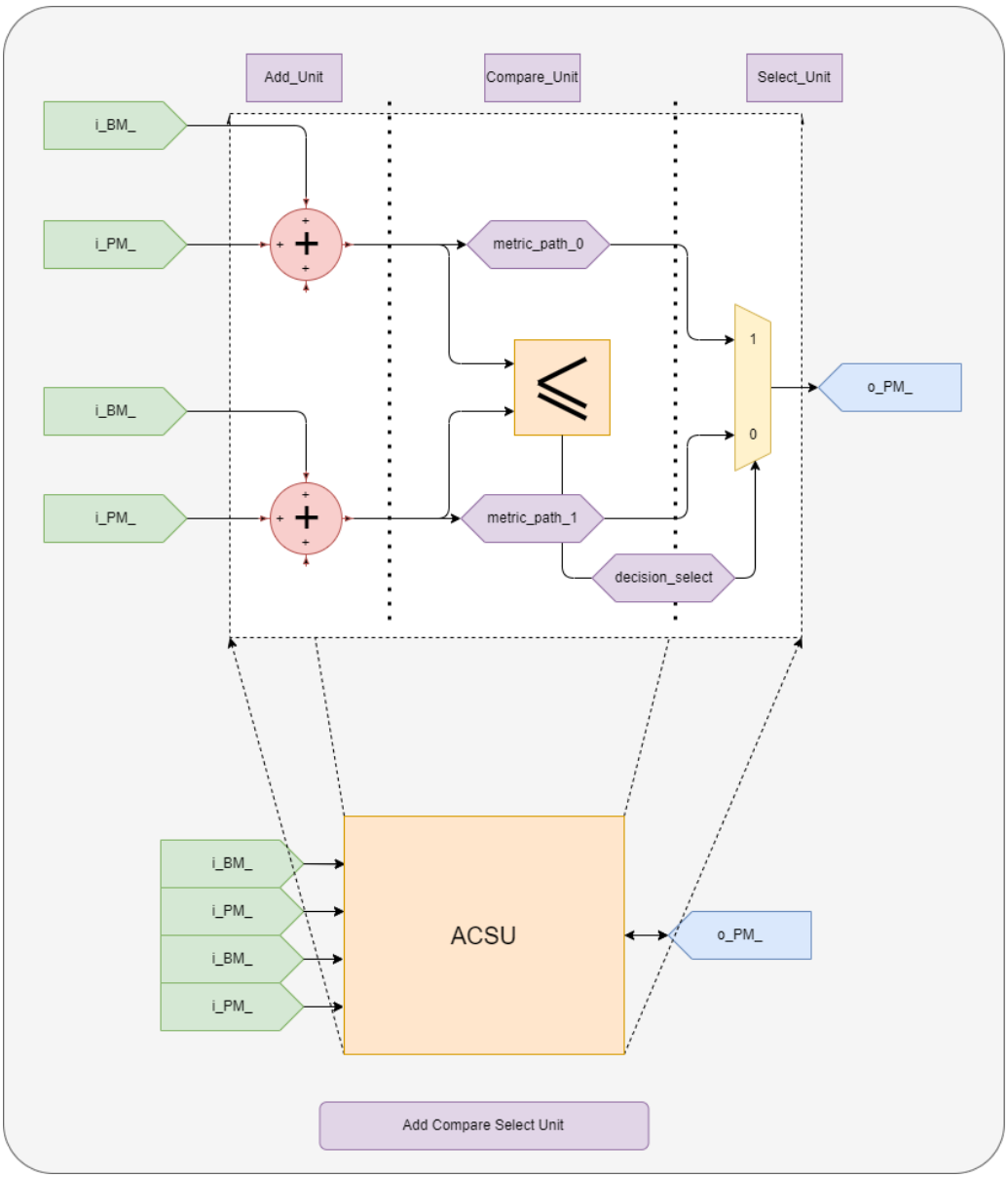
```
  01 | o_BM_3: 10 | o_valid: 1
Time: 20000 | i_rst_n = 1 | i_data: 00 | o_BM_0: 00 | o_BM_1: 01 | o_BM_2:
  01 | o_BM_3: 10 | o_valid: 1
Time: 20000 | i_rst_n = 1 | i_data: 00 | o_BM_0: 00 | o_BM_1: 01 | o_BM_2:
  01 | o_BM_3: 10 | o_valid: 1
Time: 25000 | i_rst_n = 1 | i_data: 00 | o_BM_0: 00 | o_BM_1: 01 | o_BM_2:
  01 | o_BM_3: 10 | o_valid: 1
Time: 25000 | i_rst_n = 1 | i_data: 00 | o_BM_0: 00 | o_BM_1: 01 | o_BM_2:
  01 | o_BM_3: 10 | o_valid: 1
Time: 25000 | i_rst_n = 1 | i_data: 00 | o_BM_0: 00 | o_BM_1: 01 | o_BM_2:
  01 | o_BM_3: 10 | o_valid: 1
Time: 30000 | i_rst_n = 1 | i_data: 00 | o_BM_0: 00 | o_BM_1: 01 | o_BM_2:
  01 | o_BM_3: 10 | o_valid: 1
Time: 30000 | i_rst_n = 1 | i_data: 00 | o_BM_0: 00 | o_BM_1: 01 | o_BM_2:
  01 | o_BM_3: 10 | o_valid: 1
Time: 35000 | i_rst_n = 1 | i_data: 00 | o_BM_0: 00 | o_BM_1: 01 | o_BM_2:
  01 | o_BM_3: 10 | o_valid: 1
Time: 35000 | i_rst_n = 1 | i_data: 01 | o_BM_0: 00 | o_BM_1: 01 | o_BM_2:
  01 | o_BM_3: 10 | o_valid: 1
Time: 35000 | i_rst_n = 1 | i_data: 01 | o_BM_0: 01 | o_BM_1: 00 | o_BM_2:
  10 | o_BM_3: 01 | o_valid: 1
Time: 40000 | i_rst_n = 1 | i_data: 01 | o_BM_0: 01 | o_BM_1: 00 | o_BM_2:
  10 | o_BM_3: 01 | o_valid: 1
Time: 40000 | i_rst_n = 1 | i_data: 01 | o_BM_0: 01 | o_BM_1: 00 | o_BM_2:
  10 | o_BM_3: 01 | o_valid: 1
Time: 45000 | i_rst_n = 1 | i_data: 01 | o_BM_0: 01 | o_BM_1: 00 | o_BM_2:
  10 | o_BM_3: 01 | o_valid: 1
Time: 45000 | i_rst_n = 1 | i_data: 01 | o_BM_0: 01 | o_BM_1: 00 | o_BM_2:
  10 | o_BM_3: 01 | o_valid: 1
Time: 45000 | i_rst_n = 1 | i_data: 01 | o_BM_0: 01 | o_BM_1: 00 | o_BM_2:
  10 | o_BM_3: 01 | o_valid: 1
Time: 50000 | i_rst_n = 1 | i_data: 01 | o_BM_0: 01 | o_BM_1: 00 | o_BM_2:
  10 | o_BM_3: 01 | o_valid: 1
Time: 50000 | i_rst_n = 1 | i_data: 01 | o_BM_0: 01 | o_BM_1: 00 | o_BM_2:
  10 | o_BM_3: 01 | o_valid: 1
Time: 55000 | i_rst_n = 1 | i_data: 01 | o_BM_0: 01 | o_BM_1: 00 | o_BM_2:
  10 | o_BM_3: 01 | o_valid: 1
Time: 55000 | i_rst_n = 1 | i_data: 10 | o_BM_0: 01 | o_BM_1: 00 | o_BM_2:
  10 | o_BM_3: 01 | o_valid: 1
Time: 55000 | i_rst_n = 1 | i_data: 10 | o_BM_0: 01 | o_BM_1: 10 | o_BM_2:
  00 | o_BM_3: 01 | o_valid: 1
Time: 60000 | i_rst_n = 1 | i_data: 10 | o_BM_0: 01 | o_BM_1: 10 | o_BM_2:
  00 | o_BM_3: 01 | o_valid: 1
Time: 60000 | i_rst_n = 1 | i_data: 10 | o_BM_0: 01 | o_BM_1: 10 | o_BM_2:
  00 | o_BM_3: 01 | o_valid: 1
Time: 65000 | i_rst_n = 1 | i_data: 10 | o_BM_0: 01 | o_BM_1: 10 | o_BM_2:
  00 | o_BM_3: 01 | o_valid: 1
Time: 65000 | i_rst_n = 1 | i_data: 10 | o_BM_0: 01 | o_BM_1: 10 | o_BM_2:
  00 | o_BM_3: 01 | o_valid: 1
```

```
Time: 65000 | i_rst_n = 1 | i_data: 10 | o_BM_0: 01 | o_BM_1: 10 | o_BM_2:
  00 | o_BM_3: 01 | o_valid: 1
Time: 70000 | i_rst_n = 1 | i_data: 10 | o_BM_0: 01 | o_BM_1: 10 | o_BM_2:
  00 | o_BM_3: 01 | o_valid: 1
Time: 70000 | i_rst_n = 1 | i_data: 10 | o_BM_0: 01 | o_BM_1: 10 | o_BM_2:
  00 | o_BM_3: 01 | o_valid: 1
Time: 75000 | i_rst_n = 1 | i_data: 10 | o_BM_0: 01 | o_BM_1: 10 | o_BM_2:
  00 | o_BM_3: 01 | o_valid: 1
Time: 75000 | i_rst_n = 1 | i_data: 11 | o_BM_0: 01 | o_BM_1: 10 | o_BM_2:
  00 | o_BM_3: 01 | o_valid: 1
Time: 75000 | i_rst_n = 1 | i_data: 11 | o_BM_0: 10 | o_BM_1: 01 | o_BM_2:
  01 | o_BM_3: 00 | o_valid: 1
Time: 80000 | i_rst_n = 1 | i_data: 11 | o_BM_0: 10 | o_BM_1: 01 | o_BM_2:
  01 | o_BM_3: 00 | o_valid: 1
Time: 80000 | i_rst_n = 1 | i_data: 11 | o_BM_0: 10 | o_BM_1: 01 | o_BM_2:
  01 | o_BM_3: 00 | o_valid: 1
Time: 85000 | i_rst_n = 1 | i_data: 11 | o_BM_0: 10 | o_BM_1: 01 | o_BM_2:
  01 | o_BM_3: 00 | o_valid: 1
Time: 85000 | i_rst_n = 1 | i_data: 11 | o_BM_0: 10 | o_BM_1: 01 | o_BM_2:
  01 | o_BM_3: 00 | o_valid: 1
Time: 85000 | i_rst_n = 1 | i_data: 11 | o_BM_0: 10 | o_BM_1: 01 | o_BM_2:
  01 | o_BM_3: 00 | o_valid: 1
Time: 90000 | i_rst_n = 1 | i_data: 11 | o_BM_0: 10 | o_BM_1: 01 | o_BM_2:
  01 | o_BM_3: 00 | o_valid: 1
Time: 90000 | i_rst_n = 1 | i_data: 11 | o_BM_0: 10 | o_BM_1: 01 | o_BM_2:
  01 | o_BM_3: 00 | o_valid: 1
Time: 95000 | i_rst_n = 1 | i_data: 11 | o_BM_0: 10 | o_BM_1: 01 | o_BM_2:
  01 | o_BM_3: 00 | o_valid: 1
Time: 95000 | i_rst_n = 1 | i_data: 11 | o_BM_0: 10 | o_BM_1: 01 | o_BM_2:
  01 | o_BM_3: 00 | o_valid: 1
Time: 95000 | i_rst_n = 1 | i_data: 11 | o_BM_0: 10 | o_BM_1: 01 | o_BM_2:
  01 | o_BM_3: 00 | o_valid: 1
- tb_bmu.sv:53: Verilog $finish
Time: 100000 | i_rst_n = 1 | i_data: 11 | o_BM_0: 10 | o_BM_1: 01 | o_BM_2
 : 01 | o_BM_3: 00 | o_valid: 1
Time: 100000 | i_rst_n = 1 | i_data: 11 | o_BM_0: 10 | o_BM_1: 01 | o_BM_2
 : 01 | o_BM_3: 00 | o_valid:
```

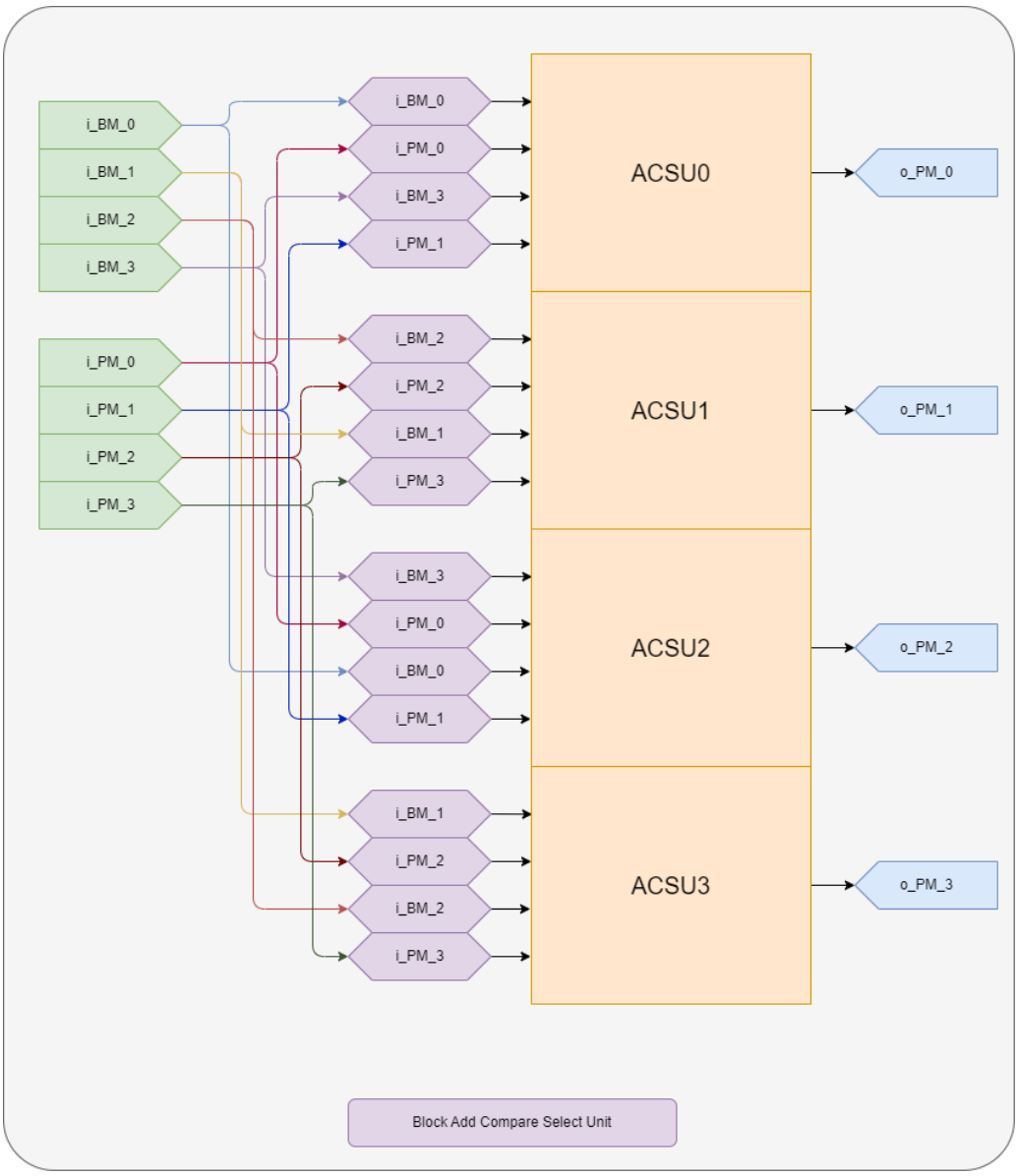
3.2. Add Compare Select Unit (ACSU)



Hình 11: Cách chuyển trạng thái trong bộ giải mã Viterbi $(3, 1, 2)$.



Hình 12: Bộ Add Compare Select.

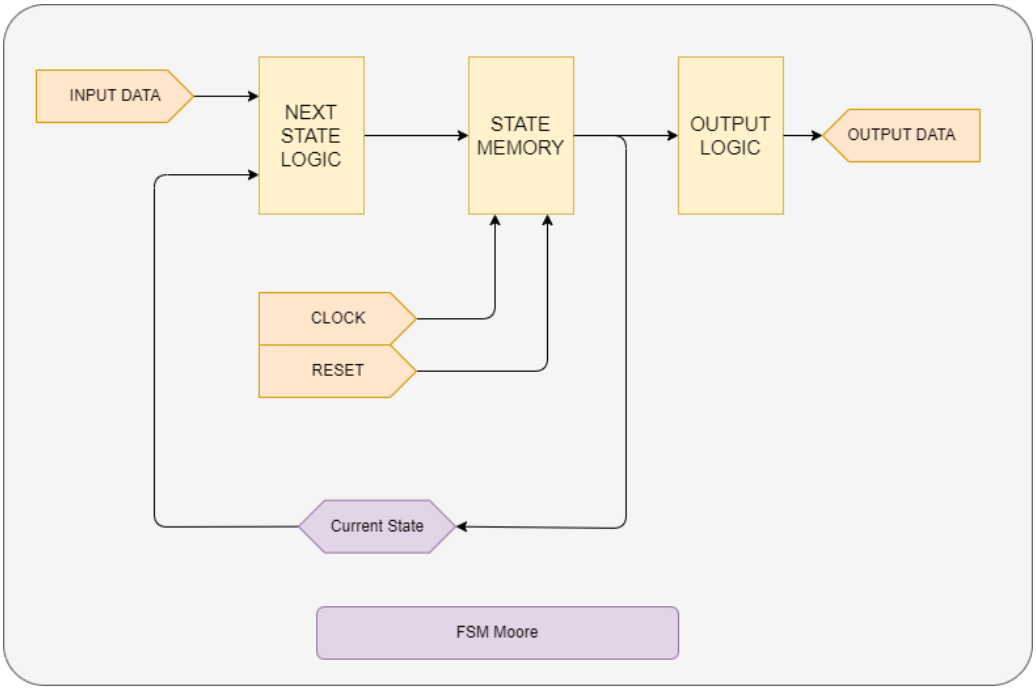


Hình 13: Bộ ACSU hoàn chỉnh.

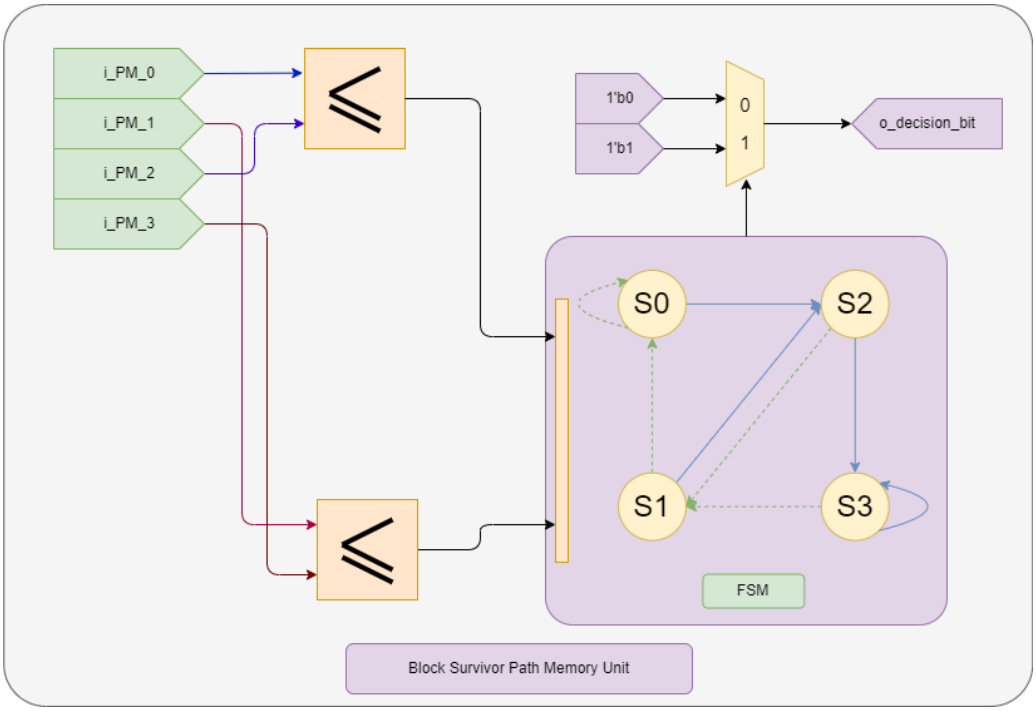
```
Starting Add_unit testbench...
============================
Time: 0 | i_rst_n = 0 |
| i_BM_0: 00 | i_BM_1: 00 | i_BM_2: 00 | i_BM_3: 00 |
| i_iPM_0: 00 | i_iPM_1: 00 | i_iPM_2: 00 | i_iPM_3: 00 |
| o_oPM_0: 00 | o_oPM_1: 00 | o_oPM_2: 00 | o_oPM_3: 00
Time: 10000 | i_rst_n = 1 |
| i_BM_0: 00 | i_BM_1: 00 | i_BM_2: 00 | i_BM_3: 00 |
| i_iPM_0: 00 | i_iPM_1: 11 | i_iPM_2: 11 | i_iPM_3: 11 |
| o_oPM_0: 00 | o_oPM_1: 00 | o_oPM_2: 00 | o_oPM_3: 00
Time: 25000 | i_rst_n = 1 |
| i_BM_0: 00 | i_BM_1: 00 | i_BM_2: 10 | i_BM_3: 00 |
| i_iPM_0: 00 | i_iPM_1: 00 | i_iPM_2: 00 | i_iPM_3: 00 |
| o_oPM_0: 00 | o_oPM_1: 00 | o_oPM_2: 00 | o_oPM_3: 00
Time: 35000 | i_rst_n = 1 |
| i_BM_0: 00 | i_BM_1: 00 | i_BM_2: 10 | i_BM_3: 00 |
```

```
| i_iPM_0: 00 | i_iPM_1: 00 | i_iPM_2: 00 | i_iPM_3: 00 |
| o_oPM_0: 00 | o_oPM_1: 00 | o_oPM_2: 00 | o_oPM_3: 00
Time: 45000 | i_rst_n = 1 |
| i_BM_0: 11 | i_BM_1: 11 | i_BM_2: 01 | i_BM_3: 01 |
| i_iPM_0: 00 | i_iPM_1: 00 | i_iPM_2: 00 | i_iPM_3: 00 |
| o_oPM_0: 00 | o_oPM_1: 00 | o_oPM_2: 00 | o_oPM_3: 00
Time: 55000 | i_rst_n = 1 |
| i_BM_0: 01 | i_BM_1: 10 | i_BM_2: 10 | i_BM_3: 01 |
| i_iPM_0: 00 | i_iPM_1: 00 | i_iPM_2: 00 | i_iPM_3: 00 |
| o_oPM_0: 01 | o_oPM_1: 01 | o_oPM_2: 01 | o_oPM_3: 01
Time: 65000 | i_rst_n = 1 |
| i_BM_0: 10 | i_BM_1: 10 | i_BM_2: 01 | i_BM_3: 00 |
| i_iPM_0: 01 | i_iPM_1: 01 | i_iPM_2: 01 | i_iPM_3: 01 |
| o_oPM_0: 01 | o_oPM_1: 10 | o_oPM_2: 01 | o_oPM_3: 10
Time: 75000 | i_rst_n = 1 |
| i_BM_0: 11 | i_BM_1: 00 | i_BM_2: 10 | i_BM_3: 00 |
| i_iPM_0: 01 | i_iPM_1: 10 | i_iPM_2: 01 | i_iPM_3: 10 |
| o_oPM_0: 00 | o_oPM_1: 01 | o_oPM_2: 00 | o_oPM_3: 01
Time: 85000 | i_rst_n = 1 |
| i_BM_0: 10 | i_BM_1: 10 | i_BM_2: 10 | i_BM_3: 10 |
| i_iPM_0: 00 | i_iPM_1: 01 | i_iPM_2: 00 | i_iPM_3: 01 |
| o_oPM_0: 00 | o_oPM_1: 00 | o_oPM_2: 00 | o_oPM_3: 00
Time: 95000 | i_rst_n = 1 |
| i_BM_0: 00 | i_BM_1: 10 | i_BM_2: 11 | i_BM_3: 00 |
| i_iPM_0: 00 | i_iPM_1: 00 | i_iPM_2: 00 | i_iPM_3: 00 |
| o_oPM_0: 10 | o_oPM_1: 10 | o_oPM_2: 10 | o_oPM_3: 10
Time: 105000 | i_rst_n = 1 |
| i_BM_0: 01 | i_BM_1: 10 | i_BM_2: 10 | i_BM_3: 01 |
| i_iPM_0: 10 | i_iPM_1: 10 | i_iPM_2: 10 | i_iPM_3: 10 |
| o_oPM_0: 00 | o_oPM_1: 10 | o_oPM_2: 00 | o_oPM_3: 10
Time: 115000 | i_rst_n = 1 |
| i_BM_0: 11 | i_BM_1: 00 | i_BM_2: 11 | i_BM_3: 10 |
| i_iPM_0: 00 | i_iPM_1: 10 | i_iPM_2: 00 | i_iPM_3: 10 |
| o_oPM_0: 01 | o_oPM_1: 10 | o_oPM_2: 01 | o_oPM_3: 10
Time: 125000 | i_rst_n = 1 |
| i_BM_0: 11 | i_BM_1: 01 | i_BM_2: 10 | i_BM_3: 11 |
| i_iPM_0: 01 | i_iPM_1: 10 | i_iPM_2: 01 | i_iPM_3: 10 |
| o_oPM_0: 10 | o_oPM_1: 00 | o_oPM_2: 10 | o_oPM_3: 00
Simulation finished.
==================================
- tb_acsu.sv:80: Verilog $finish
```

3.3. Survivor Path Memory Unit (SPMU)



Hình 14: Máy trạng thái viết ở dạng Moore.



Hình 15: Bộ SPMU.

```
Time =                  5000    | i_rst_n = 1    | o_decision = 0   | o_valid
  = 0    |
PM_0 = 0   | PM_1 = 0   | PM_2 = 0   | PM_3 = 0   |
========================================
```

```
case 1: S0 -> S0
Time = 15000  | i_rst_n = 1   | o_decision = 0  | o_valid = 1   |
PM_0 = 0  | PM_1 = 1  | PM_2 = 2  | PM_3 = 3  |
=======================================
case 2: S0 -> S2
Time = 25000  | i_rst_n = 1   | o_decision = 0  | o_valid = 1   |
PM_0 = 3  | PM_1 = 2  | PM_2 = 1  | PM_3 = 0  |
=======================================
case 3: S2 -> S1
Time = 35000  | i_rst_n = 1   | o_decision = 1  | o_valid = 1   |
PM_0 = 2  | PM_1 = 0  | PM_2 = 0  | PM_3 = 1  |
=======================================
case 4: S1 -> S2
Time = 45000  | i_rst_n = 1   | o_decision = 1  | o_valid = 1   |
PM_0 = 3  | PM_1 = 0  | PM_2 = 1  | PM_3 = 2  |
=======================================
case 5: S2 -> S3
Time = 55000  | i_rst_n = 1   | o_decision = 0  | o_valid = 1   |
PM_0 = 1  | PM_1 = 2  | PM_2 = 3  | PM_3 = 0  |
=======================================
case 6: S3 -> S3
Time = 65000  | i_rst_n = 1   | o_decision = 1  | o_valid = 1   |
PM_0 = 3  | PM_1 = 2  | PM_2 = 1  | PM_3 = 0  |
=======================================
Testbench completed successfully
=================================
- tb_spmu.sv:113: Verilog $finish
```