TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM

KHOA ĐIỆN - ĐIỆN TỬ

NGÀNH CÔNG NGHỆ KỸ THUẬT MÁY TÍNH

HCMUTE

BÁO CÁO MÔN HỌC

KIẾN TRÚC VÀ TỔ CHỨC MÁY TÍNH

ĐỀ TÀI: THIẾT KẾ MẠCH CHIA 8 BIT ĐƠN GIẢN SỬ DỤNG NGÔN NGỮ MÔ TẢ PHẦN CỨNG VERILOG

Sinh viên thực hiện:

1. **Huỳnh Minh An 22119039**
2. **Võ Quang Huy 22119082**
3. **Hồ Gia Huyên 22119083**
4. **Kiều Chí Hưng 22119085**
5. **Huỳnh Minh Quý 22119125**
6. **Trần Hoàng Tấn 22119132**

Giáo viên hướng dẫn:

TS. Phạm Văn Khoa

Thành phố Hồ Chí Minh, tháng 06/2024

Mục lục

1. GIỚI THIỆU

Trong bộ môn Kiến trúc và Tổ chức máy tính, chúng em đã được tìm hiểu về cấu trúc của một máy tính gồm 3 thành phần chính: Bộ xử lý trung tâm (CPU); Bộ nhớ (Main Memory); Bộ ngoại vi (I/O). Nhóm em đặt ra câu hỏi, làm thế nào mà máy tính có thể thực hiện được những nhiệm vụ đơn giản như là thực hiện các phép tính toán số học và logic. Tìm hiểu sâu hơn về bộ xử lý trung tâm (CPU), trong cấu tạo CPU có thành phần bộ xử lý số học ALU có nhiệm vụ thực hiện các chức năng tính toán số học và logic thông qua sự điều khiển của bộ điều khiển (Control Unit).

Nhóm em đã chọn chức năng chia số học trong bộ xử lý số học ALU để nghiên cứu và thiết kế ra một mạch chia 8 bit nguyên không dấu đơn giản sử dụng ngôn ngữ mô tả phần cứng Verilog.

Đề tài "Thiết kế mạch chia 8 bit đơn giản sử dụng ngôn ngữ Verilog" nhằm mục đích nghiên cứu và triển khai một mạch điện tử có khả năng chia các tín hiệu số 8 bit một cách chính xác và hiệu quả. Việc sử dụng ngôn ngữ mô tả phần cứng Verilog giúp tối ưu hóa quá trình thiết kế, cho phép mô phỏng và kiểm tra chức năng của mạch trước khi triển khai thực tế. Đây là một đề tài có tính ứng dụng cao trong lĩnh vực vi xử lý và hệ thống nhúng, nơi các phép chia số học thường xuyên được sử dụng.

Lý do nhóm chúng em chọn đề tài này xuất phát từ nhu cầu thực tế của ngành công nghiệp điện tử, nơi việc tối ưu hóa và giảm thiểu chi phí thiết kế phần cứng luôn được đặt lên hàng đầu. Hơn nữa, việc sử dụng Verilog không chỉ giúp chúng em làm quen với công cụ mạnh mẽ trong thiết kế mạch số, mà còn mở ra nhiều cơ hội nghiên cứu và phát triển trong lĩnh vực kỹ thuật số hiện đại. Qua đề tài này, nhóm chúng em không chỉ nắm vững kiến thức lý thuyết mà còn tích lũy được thêm kinh nghiệm thực tiễn.

2. TỔNG QUAN

2.1. Đối tượng nghiên cứu

- Ngôn ngữ mô tả phần cứng Verilog: Tìm hiểu về ngôn ngữ Verilog, các cú pháp lập trình, ứng dụng dùng để lập trình Verilog và mô phỏng. Biết cách sử dụng Verilog để thiết kế ra mạch chia 8 bit đơn giản.

- Thuật toán phép chia: Tìm hiểu và nghiên cứu về thuật toán chia có khôi phục số nhớ (Restoring Division Algorithm) để ứng dụng vào thiết kế mạch chia 8 bit đơn giản.

- Cấu trúc mạch chia: Nghiên cứu và thiết kế được mạch chia 8 bit đơn giản theo hướng cấu trúc sử dụng khối Proccessing Unit và sử dụng mô hình máy trạng thái hữu hạn FSM (Finite State Machine).

2.2. Công cụ hỗ trợ

- Vivado: Công cụ hỗ trợ việc lập trình mô tả phần cứng mạch chia, synthesis, cũng như mô phỏng thử hoạt động của mạch chia.

- Xilinx ISE: Cũng giống như Vivado hỗ trợ việc lập trình mô tả phần cứng mạch chia, synthesis, dễ dàng mô phỏng và kiểm thử mạch chia.

2.3. Phân chia công việc

Người thực hiện / Công việc	Huỳnh Minh An	Võ Quang Huy	Hồ Gia Huyên	Kiều Chí Hưng	Huỳnh Minh Quý	Trần Hoàng Tấn
Tìm hiểu thuật toán phép chia						✓
Tìm hiểu mạch chia dùng FSM		✓				

Tìm hiểu cấu trúc mạch chia dùng khối PU sử dụng mạch cộng toàn phần FA	✓				✓	
Tìm hiểu cấu trúc mạch chia dùng khối PU sử dụng mạch cộng toàn phần FS			✓	✓		
Mô phỏng – lập trình		✓	✓		✓	
Tổng kết LUT và thời gian delay của từng phương pháp				✓		✓
Bản trình chiếu	✓	✓				
Báo cáo	✓	✓	✓	✓	✓	✓

3. THỰC HIỆN

3.1. Phương pháp chia khôi phục số nhớ (Restoring Division Algorithm)

3.1.1 Giới thiệu

Phương pháp chia khôi phục số nhớ (Restoring Division Algorithm) là một phương pháp được sử dụng để thực hiện các phép chia trên các số nguyên không dấu trong số học máy tính. Nó được thiết kế để tính toán thương số và số dư một cách hiệu quả khi chia một số nguyên không dấu (số bị chia) cho một số khác (số chia), đồng thời giảm thiểu số bước tính toán.

Quy trình khôi phục các số nguyên không dấu sẽ được sử dụng. Thuật ngữ "khôi phục" đề cập đến thực tế là sau mỗi lần lặp lại, giá trị của thanh ghi A sẽ được khôi phục.

3.1.2 Thuật toán phép chia

Các bước thực hiện thuật toán phép chia:

Bước 1: Thiết lập các thanh ghi với các giá trị tương ứng. Trong đó x là số bị chia, y là số chia, A là số dư tạm thời và bắt đầu là 0, i là biến đếm (số lần thực hiện vòng lặp).

Bước 2: Tiến hành dịch trái A và x, coi A và x là một đơn vị thống nhất.

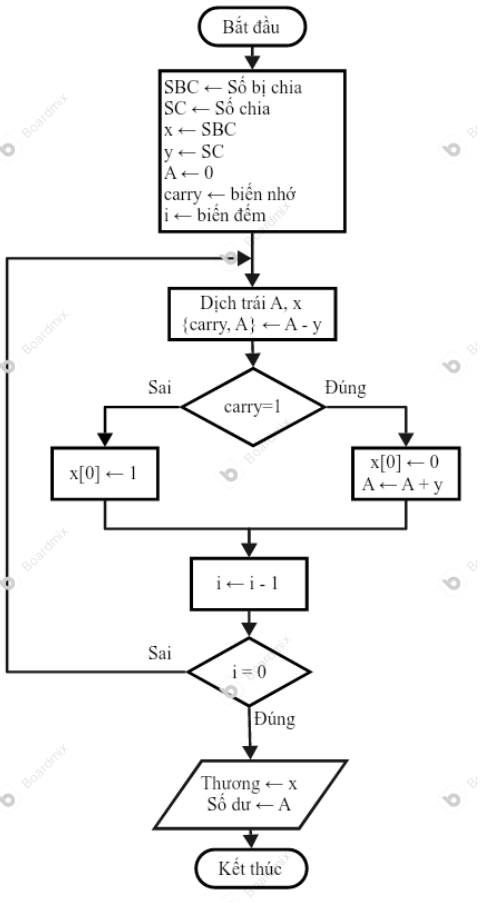
Bước 3: Thực hiện phép trừ A-y và gán vào {carry,A}.

Bước 4: Kiểm tra bit carry. Nếu là 0, đặt bit có trọng số thấp nhất của x là 1. Ngược lại nếu là 1, đặt bit có trọng số thấp của x là 0 và khôi phục giá trị của thanh ghi A về trạng thái của A trước khi thực hiện phép trừ.

Bước 5: Giảm giá trị của bộ đếm i đi một.

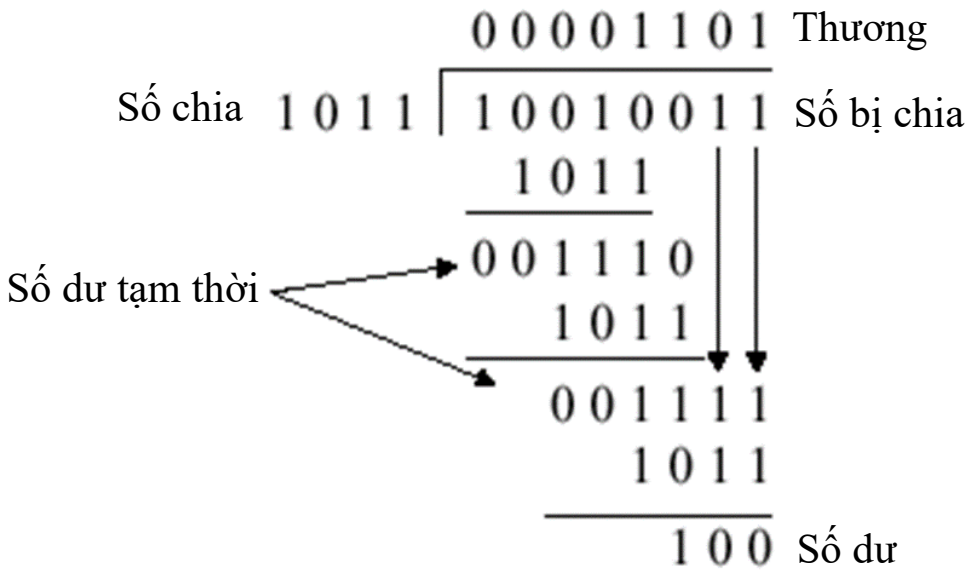
Bước 6: Kiểm tra xem giá trị i đã về 0 hay chưa. Nếu chưa, quay lại bước 2 và lặp lại quy trình.

Bước 7: Cuối cùng, kết thúc quá trình chia với thương số nằm trong thanh thi x, và phần số dư sẽ nằm trong thanh ghi A.



Hình 1: Flowchart thuật toán mạch chia khôi phục số nhớ

3.1.3 Ví dụ về phép chia sử dụng phương pháp khôi phục số nhớ



Hình 2: Ví dụ phép chia sử dụng phương pháp khôi phục số nhớ

Bảng 1: Bảng thực thi cho ví dụ trên

i	carry	A	x	y	Thực hiện
8		0000 0000	1001 0011	1011	Khởi tạo
		0000 0001	0010 011?		Dịch trái A,x
	1	1111 0110	0010 011?		{carry,A} ← A - y
7		0000 0001	0010 0110		carry = 1→ Đúng x[0]←0 Trả A về trước đó i = i - 1
		0000 0010	0100 110?		i = 0 → sai Dịch trái A,x
	1	1111 1000	0100 110?		{carry,A} ← A - y
6		0000 0010	0100 1100		carry = 1→ Đúng x[0]←0 Trả A về trước đó i = i - 1
		0000 0100	1001 100?		i = 0 → sai Dịch trái A,x
	1	1111 1010	1001 100?		{carry,A} ← A - y
5		0000 0100	1001 1000		carry = 1→ Đúng

					x[0]←0 Trả A về trước đó i = i - 1
		0000 1001	0011 000?		i = 0 → sai Dịch trái A,x
	1	1111 1110	0011 000?		{carry,A} ← A - y
4		0000 1001	0011 0000		carry = 1→ Đúng x[0]←0 Trả A về trước đó i = i - 1
		0001 0010	0110 000?		i = 0 → sai Dịch trái A,x
	0	0000 0111	0110 000?		{carry,A} ← A - y
3		0000 0111	0110 0001		carry = 1→sai x[0]←1 i = i - 1
		0000 1110	1100 001?		i = 0 → sai Dịch trái A,x
	0	0000 0011	1100 001?		{carry,A} ← A - y
2		0000 0011	1100 0011		carry = 1→sai x[0]←1 i = i - 1
		0000 0111	1000 011?		i = 0 → sai Dịch trái A,x
	1	1111 1100	1000 011?		{carry,A} ← A - y
1		0000 0111	1000 0110		carry = 1→ Đúng x[0]←0 Trả A về trước đó i = i - 1
		0000 1111	0000 110?		i = 0 → sai Dịch trái A,x
	0	0000 0100	0000 110?		{carry,A} ← A - y
0		0000 0100	0000 1101		carry = 1→sai x[0]←1 I = I – 1
		Số dư = 4_{10}	Thương = 13_{10}		I = 0 → Đúng Thương ← x Số dư ← A

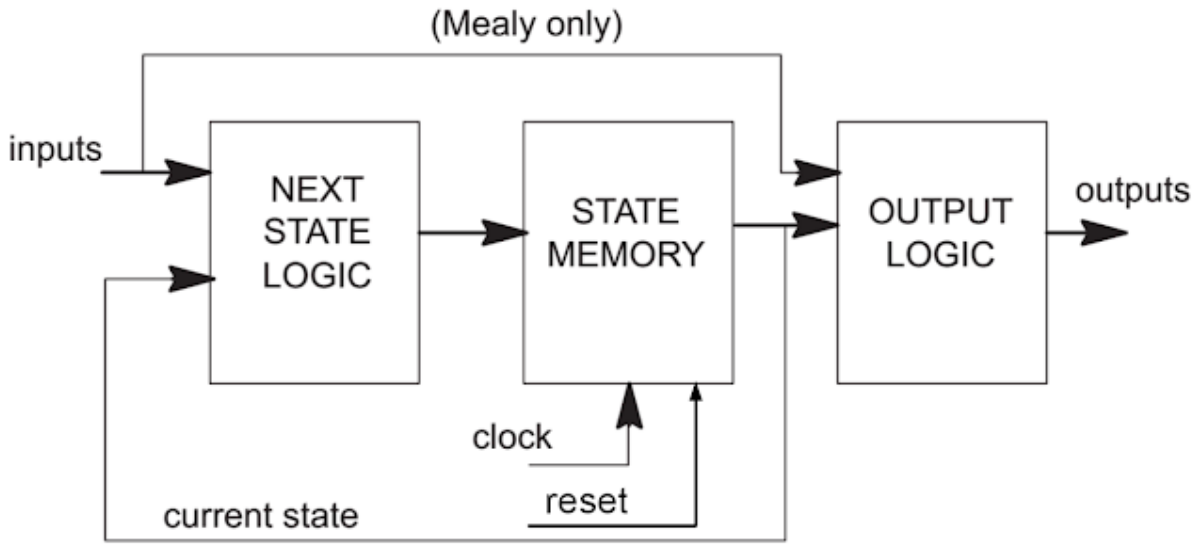
Thuật toán phép chia khôi phục số nhớ là một phương pháp hiệu quả và chính xác để thực hiện phép chia trên các số nguyên không dấu. Bằng cách kết hợp các nguyên tắc trừ, dịch chuyển và so sánh, thuật toán này đạt được kết quả chính xác đồng thời giảm thiểu các bước tính toán.

3.2. Mạch chia dùng máy trạng thái hữu hạn FSM (Finite State Machine)

3.2.1 Tổng quan về máy trạng thái hữu hạn

3.2.1.1 Giới thiệu

Finite State Machine (FSM) hay Máy trạng thái hữu hạn, là một phương pháp thiết kế mạch bằng cách mô hình hóa mạch tuần tự bằng một mô hình toán học, sau đó sẽ hiện thực mô hình toán học này xuống phần cứng.



Hình 3: Mô hình cơ bản của FSM

FSM gồm có 3 phần cơ bản:

- Mạch tạo trạng thái kế tiếp (Next state logic) là mạch tổ hợp phụ thuộc vào các ngõ vào (inputs) của FSM và giá trị trạng thái hiện tại lấy từ bộ nhớ trạng thái (State Memory).

- Bộ nhớ trạng thái (state memory) là phần tử lưu trạng thái hiện tại của FSM. Nó có thể là Flip-Flop, Latch, ... State Memory lấy ngõ vào từ mạch tạo trạng thái kế tiếp

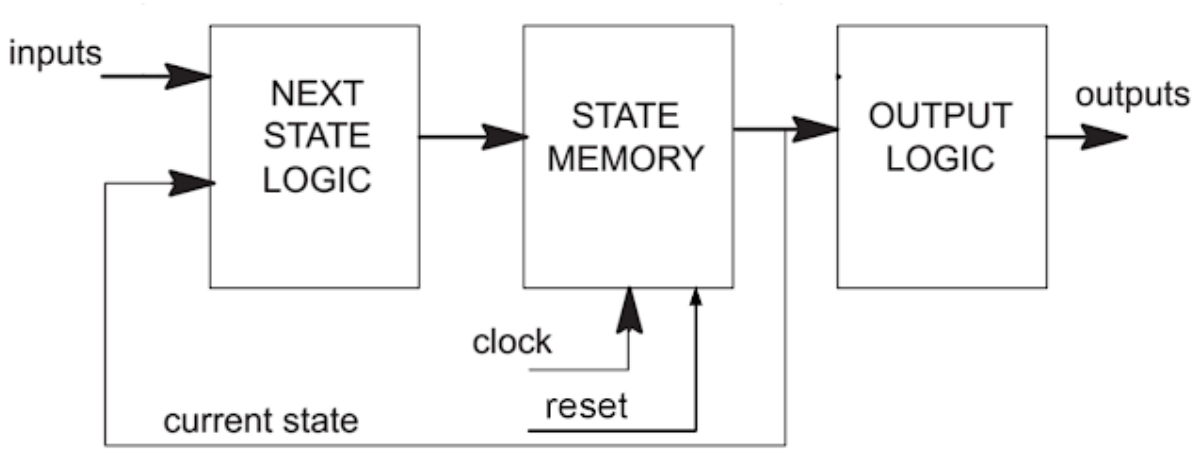
(Next State Memory). Bộ nhớ trạng thái thường được sử dụng trong các thiết kế đồng bộ là FF hoạt động theo xung clock. Một tín hiệu reset có thể phải sử dụng để khởi động FSM đến một giá trị ban đầu. Reset không cần sử dụng đối với các FSM luôn hoạt động đúng dù giá trị ban đầu của FF là bao nhiêu.

Mạch tạo ngõ ra (output logic) là mạch tổ hợp tạo giá trị ngõ ra tương ứng với trạng thái hiện tại của FSM. Mạch này lấy ngõ vào là giá trị trạng thái hiện tại và có thể tổ hợp thêm ngõ vào của FSM.

3.2.1.2 Phân loại

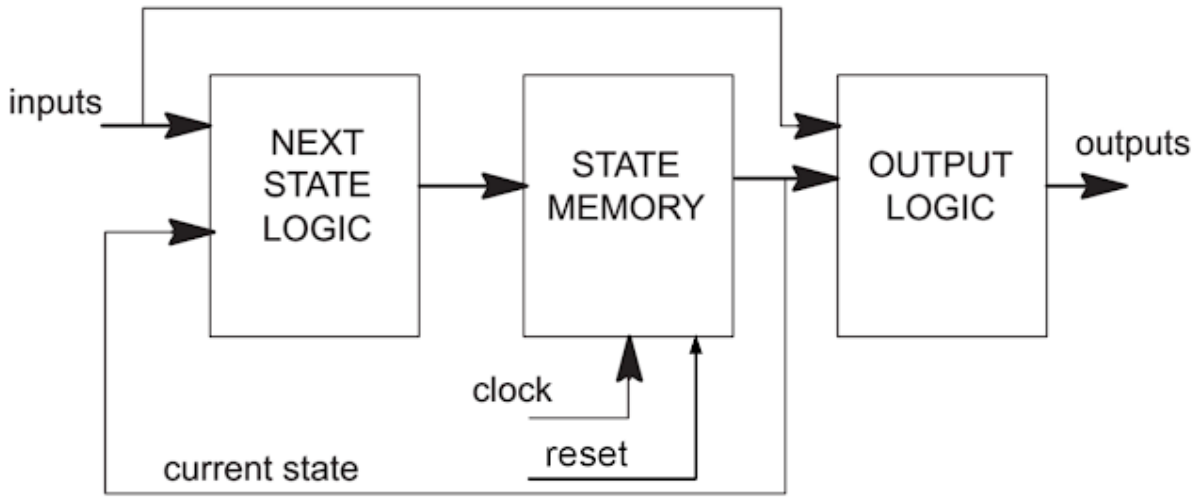
FSM được chia làm 2 loại:

- FSM Moore: Mạch tạo ngõ ra chỉ phụ thuộc vào trạng thái hiện tại (State Memory), không phụ thuộc các đầu vào (inputs).



Hình 4: FSM Moore

- FSM Mealy: Mạch tạo ngõ ra phụ thuộc cả các đầu vào (inputs) và trạng thái hiện tại.

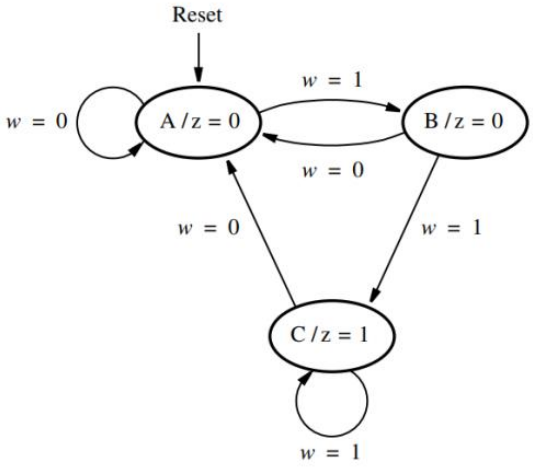


Hình 5: FSM Mealy

3.2.1.3 Sơ đồ trạng thái

Sơ đồ trạng thái là một cách biểu diễn chức năng của một FSM hướng đồ thị trực quan và dễ hiểu đối với những FSM đơn giản.

Bước đầu tiên trong việc thiết kế một máy trạng thái hữu hạn là xác định cần bao nhiêu trạng thái và những chuyển đổi nào có thể thực hiện được từ trạng thái này sang trạng thái khác



Hình 6: Sơ đồ trạng thái FSM

Bảng 2: Bảng trạng thái tương ứng với hình 6

Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	A	B	0
B	A	C	0
C	A	C	1

Cấu tạo của sơ đồ trạng thái bao gồm:

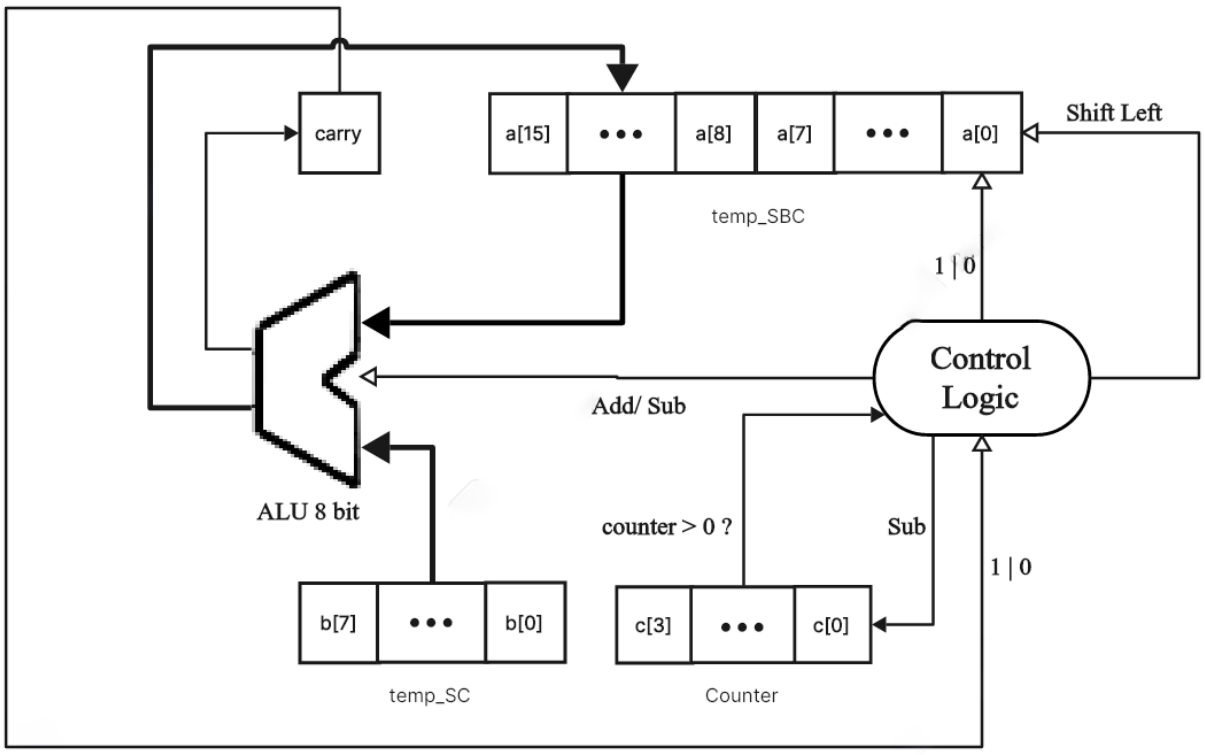
Các trạng thái (hay còn gọi là các nút): Mỗi trạng thái được biểu diễn bởi một hình tròn bao quanh tên của trạng thái hoặc chuỗi bit mã hóa cho trạng thái.

Các đường chuyển trạng thái (hay còn gọi là các cạnh): Việc chuyển trạng thái được biểu diễn bằng các mũi tên kết nối từ trạng thái này tới trạng thái khác kèm theo điều kiện chuyển trạng thái. Điểm xuất phát của mỗi mũi tên được gọi là trạng thái hiện tại, điểm kết thúc của mỗi mũi tên được gọi là trạng thái kế tiếp. Điều kiện chuyển trạng thái thường là điều kiện của các ngõ vào hoặc thậm chí là không có điều kiện. Việc chuyển trạng thái diễn ra khi ngõ vào clock tích cực.

Sơ đồ trạng thái giúp ta có một cái nhìn khái quát và trực quan về thuật toán và việc chuyển đổi giữa các trạng thái. Tuy nhiên, khi số lượng trạng thái cũng như ngõ vào và ngõ ra tăng lên thì sơ đồ trên trở nên phức tạp. Để giải quyết vấn đề đó, ta dùng đến bảng trạng thái (Hình 3.2.4).

Bảng trạng thái tương đương với sơ đồ trạng thái và chúng có thể ánh xạ 1:1 từ sơ đồ trạng thái. Vì thế, nó cũng thể hiện trực quan chức năng của FSM. Khi số lượng trạng thái tăng lên thì mở rộng thêm hàng. Khi số lượng ngõ vào, ra tăng lên thì mở rộng thêm cột. Điều đó giúp ta thể hiện được chức năng của FSM mà ko làm phức tạp sơ đồ trạng thái.

3.2.2. Cấu tạo mạch chia dùng FSM



Hình 7: Sơ đồ cấu trúc mạch chia

Mạch chia bao gồm:

- Thanh ghi 16 bits temp_SBC giúp lưu trữ số bị chia lúc đầu và thương và số dư khi kết thúc.

- Thanh ghi 8 bits temp_SC giúp lưu trữ số chia.

- Biến carry giúp lưu trữ bit tràn của thanh ghi temp_SBC.

- Biến counter 4 bits đếm số bit đã xử lý.

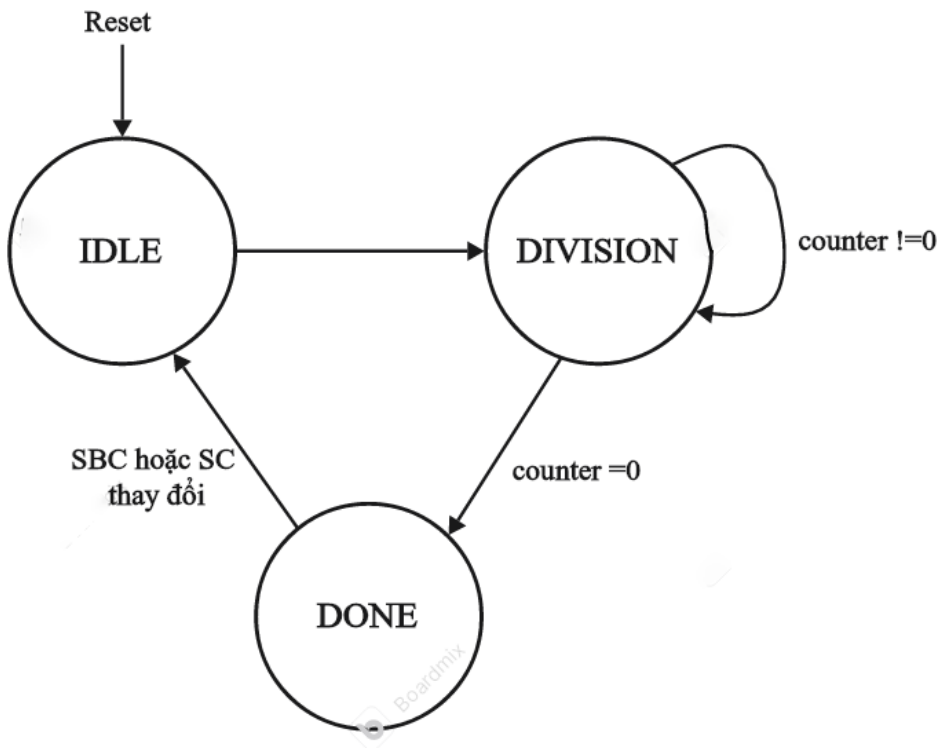
- Khối ALU 8 bits có chức năng xử lý các phép tính cộng/ trừ.

- Khối Control Unit có chức năng điều khiển hoạt động của cả mạch.

3.2.3. Nguyên lý hoạt động

Nguyên lý hoạt động của mạch chia dùng FSM được dựa trên thuật toán chia khôi phục số nhớ và thuật toán sẽ được mô hình hóa bằng phương pháp máy trạng thái hữu hạn FSM.

Mạch chia gồm 3 trạng thái:



Hình 8: Sơ đồ trạng thái mạch chia FSM

- IDLE: Trạng thái này là trạng thái đầu tiên của chu kì. Ở trạng thái này, mọi thứ sẽ được chuẩn bị cho quá trình thực hiện phép chia. Số bị chia sẽ được nạp vào 8 bit cuối của thanh ghi temp_SBC và các bit còn lại cảu thanh ghi sẽ được lấp đầy bởi các bit 0. Số chia được nạp vào thanh ghi temp_SC. Biến counter được đặt thành 1000B tương ứng với 8 bit của mạch chia. Sau khi thực hiện xong những tác vụ trên thì sẽ chuyển sang trạng thái tiếp theo, DIVISION.

- DIVISION: Đây là trạng thái cốt lõi nơi phép chia được thực hiện. Khi bắt đầu, Khối Control Logic sẽ điều khiển thanh ghi temp_SBC dịch trái một bit. Lúc này, ta có thể xem 8 bit đầu của thanh ghi temp_SBC là nơi lưu số dư tạm thời của phép chia. Sau đó, số dư tạm thời hiện tại sẽ được trừ cho temp_SC, hiệu của phép tính sẽ được lưu trở lại 8 bit đầu của temp_SBC và biến carry lưu các bit tràn của nó.

Nếu carry = 1 tức là số dư tạm thời hiện tại âm. Lúc này, Control Logic sẽ điều khiển ALU cộng số dư tạm thời hiện tại cho temp_SC nhằm khôi phục số dư tạm thời lại ban đầu, đồng thời temp_SBC[0] = 0 và biến counter trừ đi 1.

Trái lại, nếu carry = 0 thì số dư tạm thời sẽ được giữ nguyên, temp_SBC[0] = 1 và counter trừ 1.

Mạch sẽ thực hiện và lặp lại trạng thái cho đến khi counter về 0. Khi đó, mạch sẽ chuyển sang trạng thái tiếp theo, DONE.

- DONE: Sau khi thực hiện phép chia ở trạng thái trước, giá trị thương sẽ nằm ở 8 bit cuối của thanh ghi temp_SBC và số dư sẽ là 8 bit đầu. Trạng thái này có nhiệm vụ đưa giá trị thương và số dư tới các đầu ra tương ứng của mạch. Đồng thời, thực hiện lại chu kì trạng thái mới nếu số bị chia hoặc số chia có sự thay đổi.

3.3. Mạch chia khôi phục số nhớ dùng khối PU sử dụng mạch cộng toàn phần

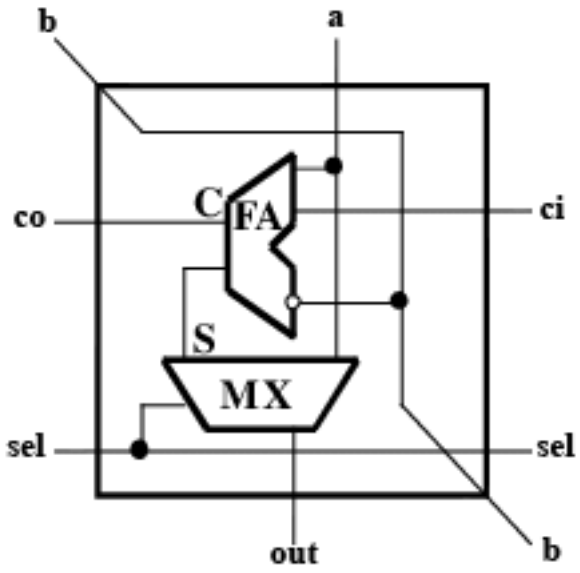
3.3.1. Khối Processing Unit (PU) sử dụng FA

Khối Processing Unit (PU) bao gồm mạch cộng toàn phần (FA) với số bị chia a, số chia b và mạch đa hợp 2 sang 1 (Mux21), sẽ đảm nhận việc xử lý 1 bit của tín hiệu đầu vào thông qua mạch cộng toàn phần (FA) và cổng NOT được nối vào bit của số chia b. Đầu ra sẽ là thương hoặc số dư, cụ thể ở đây là "co" và "out".

Để thực hiện hiện phép chia thì bit của số chia sẽ được đảo thông qua cổng NOT rồi mới nối vào mạch cộng toàn phần (FA) để tính toán và cho ra thương và số dư. Kết quả cuối cùng tại đầu ra của số dư sẽ được quyết định giá trị là "0" hoặc "1" thông

qua chân chọn "sel" của bộ ghép kênh hay mạch đa hợp 2 sang 1. Bộ ghép kênh này sẽ chọn giữa hai đầu vào là S và a đưa ra một giá trị duy nhất tại đầu ra dựa trên giá trị của chân chọn "sel".

Tóm lại, khối này không chỉ đơn giản là một phần tử trong mạch số, mà còn là một thành phần quan trọng đảm bảo rằng các phép toán số học liên quan đến chia và lấy dư được thực hiện chính xác và đáng tin cậy thông qua sự phối hợp chặt chẽ giữa mạch cộng toàn phần, cổng NOT, và bộ ghép kênh.

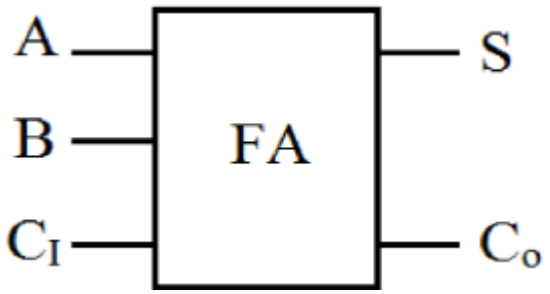


Hình 9: Sơ đồ khối PU sử dụng Full Adder

Dựa vào thành phần và những điều trên mà mô tả lại bằng Verilog.

3.3.2. Mạch cộng toàn phần (Full Adder)

Mạch cộng toàn phần được thiết kế dựa theo bảng trạng thái và sau đó được mô tả lại bằng Verilog.



Hình 10: Sơ đồ khối mạch cộng toàn phần

Bảng 3: Bảng trạng thái mạch cộng toàn phần

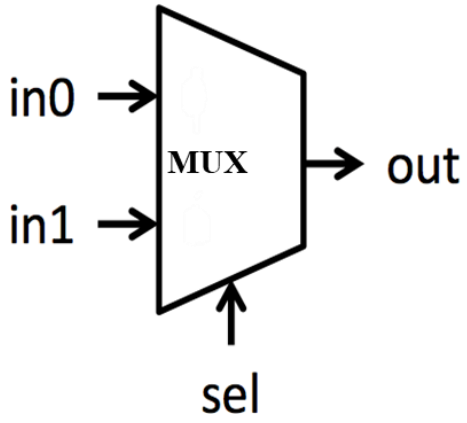
Ngõ vào			Ngõ ra	
A	B	C_I	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Dựa vào bảng trạng thái mà mô tả lại bằng Verilog thông qua có thể sử dụng bìa Karnaugh để rút ra được hàm ngõ ra:

$$S = A \oplus B \oplus Ci$$

$$Co = AB + ACi + BCi$$

3.3.3 Mạch đa hợp 2 sang 1 (Mux 21) trong khối PU sử dụng FA



Hình 11: Sơ đồ khối đa hợp 2 sang 1

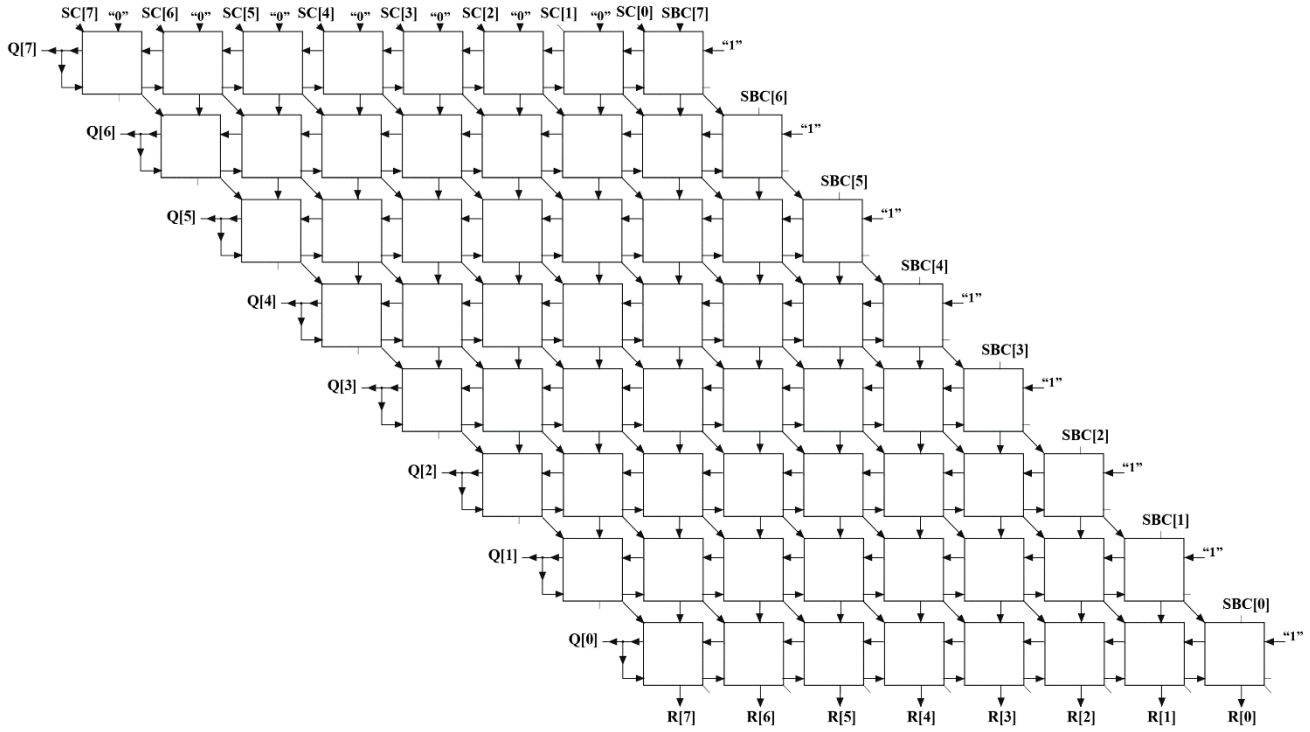
Mạch đa hợp 2 sang 1 được sử dụng ở đây nhằm chọn ngõ ra của khối PU cụ thể trong khối PU là số dư thông qua chân sel điều đó được thể hiện qua bảng trạng thái dưới đây:

Bảng 4: Bảng trạng thái mạch đa hợp 2 sang 1

sel	in0	in1	out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Dựa vào bảng trạng thái ta mô tả lại bằng Verilog.

3.3.4. Sơ đồ khối mạch chia



Hình 12: Cấu trúc mạch chia 8 bit dùng khối PU sử dụng mạch cộng toàn phần

Vì là mạch chia 8 bit nên sẽ có tổng cộng 8 hàng tượng trưng cho 8 bit số bị chia (SBC) đầu vào và 8 bit thương (Q) đầu ra. Mỗi hàng sẽ gồm 8 khối PU được ghép nối với nhau tượng trưng cho 8 bit số chia (SC) đầu vào ở hàng đầu tiên và 8 bit số dư (R) đầu ra ở hàng cuối cùng.

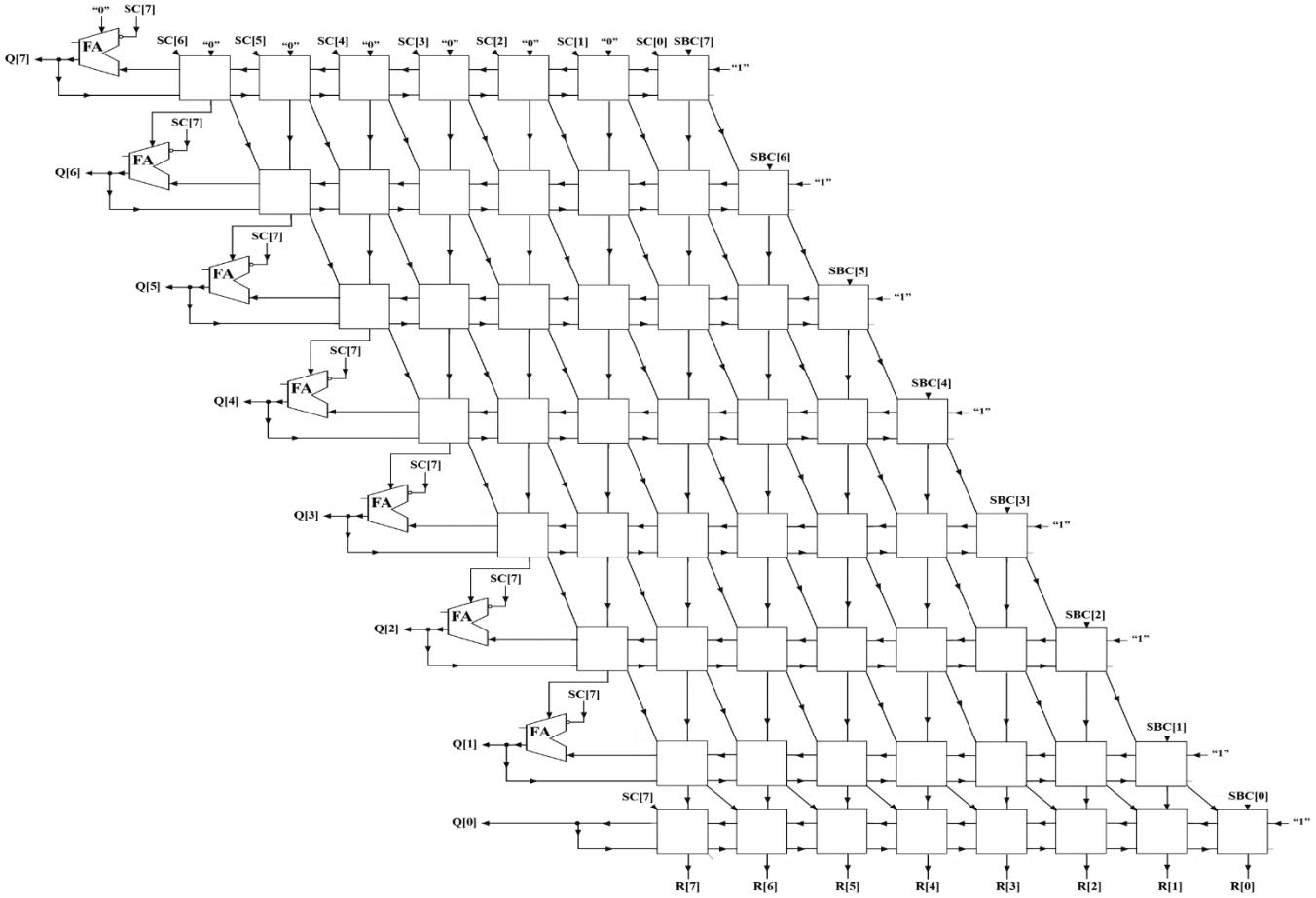
Đầu vào của từng khối PU sẽ là từng bit của số bị chia và số chia, mỗi khối PU sẽ lần lượt xử lý các bit đầu vào này, các bit số chia sẽ được đưa qua cổng đảo trước khi thực hiện thao tác trừ với số bị chia. Trong trường hợp này, thao tác trừ giữa các bit số bị chia và số chia là phép cộng bù 2 nên ở đầu vào "Ci" của các khối PU đầu tiên mỗi hàng sẽ được đưa vào mức logic '1'.

Các đầu ra "Co" của từng khối PU trên một hàng sẽ nối tiếp vào "Ci" của khối PU tiếp theo cho đến khối PU cuối cùng (PU thứ 8), đầu ra "Co" sẽ là bit thương (Q) đồng thời sẽ đóng vai trò là chân sel chọn ngõ ra "out" của khối mux 2 sang 1 bên trong khối PU trên cùng một hàng đó.

Nếu chân “Co” là bit 0 tức là thao tác cộng bù 2 giữa số bị chia và số chia không xuất hiện bit tràn, bit thương (Q) sẽ ghi nhận bit 0 và đồng thời mạch đa hợp trong khối PU sẽ thực hiện khôi phục số bị chia ban đầu làm đầu ra “out” để nối tiếp vào đầu vào của các khối PU ở hàng phía dưới. Ngược lại, chân “Co” cho ra bit 1 tức là xuất hiện bit tràn của phép cộng bù 2, bit thương (Q) sẽ ghi nhận bit 1 và mạch đa hợp trong khối PU sẽ thực hiện lấy kết quả của phép cộng bù 2 làm đầu ra “out” nối tiếp vào đầu vào của các khối PU ở hàng dưới. Sau đó, sẽ thực hiện dịch số chia sang phải 1 bit để tiếp tục so sánh với số bị chia và sẽ lặp lại bước trên.

Mạch chia sẽ thực hiện các thao tác trên cho đến khi đến hàng PU cuối cùng thì đầu ra “out” của các khối PU sẽ là số dư của phép chia. Thương (Q) cuối cùng sẽ là các bit đầu ra “Co” của các khối PU cuối cùng mỗi hàng, bit MSB ở hàng trên cùng cho đến bit LSB ở hàng cuối cùng.

3.3.5. Tối ưu sơ đồ khối



Hình 13: Cấu trúc mạch chia 8 bit dùng khối PU sử dụng mạch cộng toàn phần tối ưu khối đa hợp 2 sang 1

Nhận thấy ở những khối PU cuối cùng của mỗi hàng trừ hàng cuối cùng ra, đầu ra "out" của khối mux 2 sang 1 không được nối làm giá trị đầu vào cho các khối PU ở hàng tiếp theo phía dưới nên chúng ta sẽ tiến hành lược bỏ bớt khối mux 2 sang 1 để đánh giá xem mạch chia có còn cho ra đúng kết quả như mong muốn và thời gian delay sẽ thay đổi như thế nào so với thiết kế ban đầu.

Sau khi lược bỏ khối mux 2 sang 1 ở những khối PU cuối cùng của mỗi hàng thì chỉ còn loại khối cộng toàn phần FA, chúng ta chỉ quan tâm đến đầu ra "Co" của khối FA. Các thao tác thực hiện của mạch chia này cũng tương tự như trình bày ở phần trên, đầu ra "Co" của khối FA cuối cùng mỗi hàng sẽ ghi nhận bit thương cũng như chân sel chọn giá trị đầu ra cho các khối PU trên cùng 1 hàng.

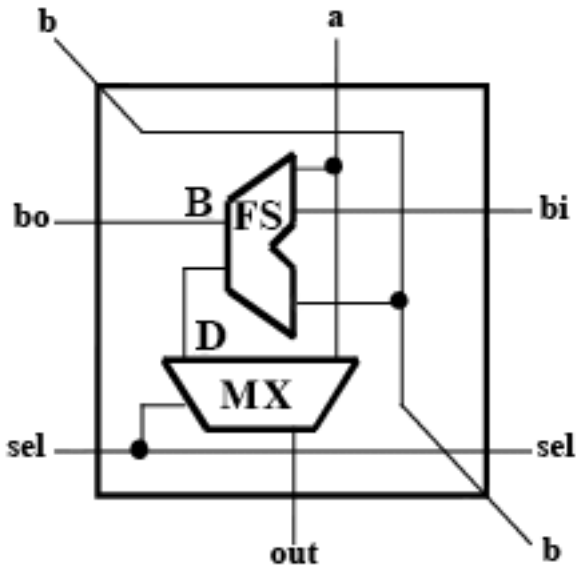
Ở hàng cuối cùng, vì đầu ra "out" của mỗi khối PU là giá trị bit của số dư (R) phép chia nên sẽ được giữ nguyên.

3.4. Mạch chia khôi phục số nhớ dùng khối PU sử dụng mạch trừ toàn phần

Để tối ưu hóa thiết kế, chúng ta sẽ thay thế mạch cộng toàn phần trong khối PU bằng mạch trừ toàn phần, sau đó so sánh hai thiết kế này với nhau.

Mạch chia 8 bit theo phương pháp khôi phục số nhớ sử dụng khối PU sẽ được thực hiện bằng cách ghép các khối PU sử dụng mạch trừ toàn phần lại thành một mạch hoàn chỉnh.

3.4.1. Khối Processing Unit (PU) sử dụng FS



Hình 14: Cấu tạo của khối PU sử dụng FS

Khối PU được thiết kế với 4 ngõ vào (a, b, bi, sel) và 2 ngõ ra (bo, out), nhằm xử lý từng cặp bit của số bị chia (SBC) và số chia (SC). Bên trong khối PU gồm mạch trừ toàn phần và mạch đa hợp 2 sang 1.

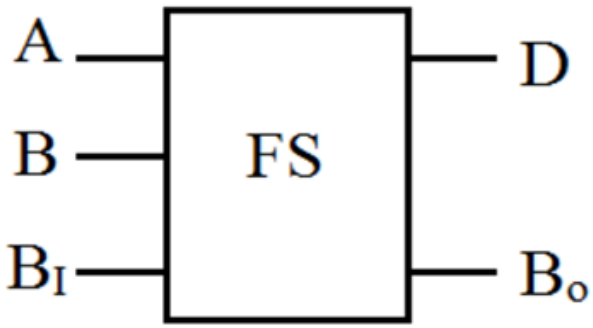
Mạch trừ toàn phần có nhiệm vụ thực hiện phép trừ và đồng thời cũng là phép chia bởi vì khi a trừ b mà không cần mượn (bo = 0), điều này có nghĩa là a ≥ b, hay nói

cách khác, a có thể chia cho b. Ngược lại bo=1 thì a < b, a không thể chia cho b. Ở khối PU sử dụng mạch trừ toàn phần thì không cần đảo ngõ vào b vì mạch trừ toàn phần thực hiện phép trừ trực tiếp không phải "cộng bù 2".

Mạch đa hợp 2 sang 1, với ngõ vào bo nối đến chân sel, sẽ chọn đưa a hoặc D ra ngõ out sau khi thực hiện phép chia. Nếu bo = 0, ngõ ra out sẽ là D, tương ứng với việc số bị chia (SBC) chia được cho số chia (SC), khi đó SBC trừ SC để lấy số dư. Ngược lại, nếu bo = 1, ngõ ra out sẽ là a (SBC), tức là phép chia không thành công và số dư chính là a (SBC).

3.4.2. Mạch trừ toàn phần (Full Subtractor)

Mạch trừ được thiết kế như dựa trên bảng trạng thái và sau đó được mô tả lại bằng Verilog.



Hình 15: Sơ đồ khối mạch trừ toàn phần

Bảng 5: Bảng trạng thái mạch trừ toàn phần

Ngõ vào			Ngõ ra	
A	B	B_I	D	B_o
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

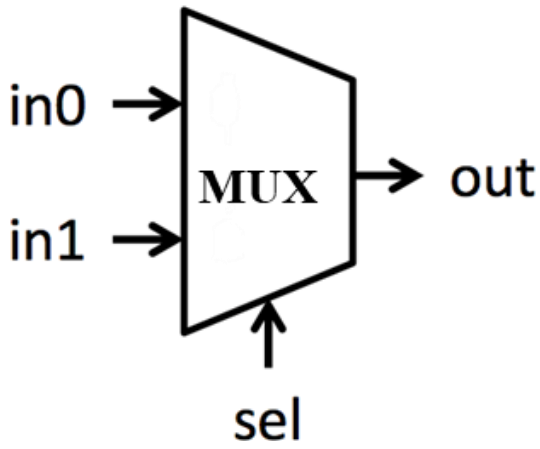
Dựa vào bảng trạng thái ta có thể sử dụng bìa Karnaugh để rút ra được hàm ngõ ra như sau:

$$D = A \oplus B \oplus Bi$$

$$Bo = \bar{A}B + \bar{A}Bi + Bbi$$

3.4.3. Mạch đa hợp 2 sang 1 (Mux 21) trong khối PU sử dụng FS

Khác với khi sử dụng mạch cộng toàn phần, nếu co = 1 thì a ≥ b, còn nếu co = 0 thì a < b. Do đó khi thiết kế mạch MUX 2 sang 1, ta phải đảo lại điều kiện chọn tín hiệu đưa ra ngõ out. Vì vậy, bảng trạng thái của mạch sẽ khác so với khi sử dụng mạch cộng toàn phần.



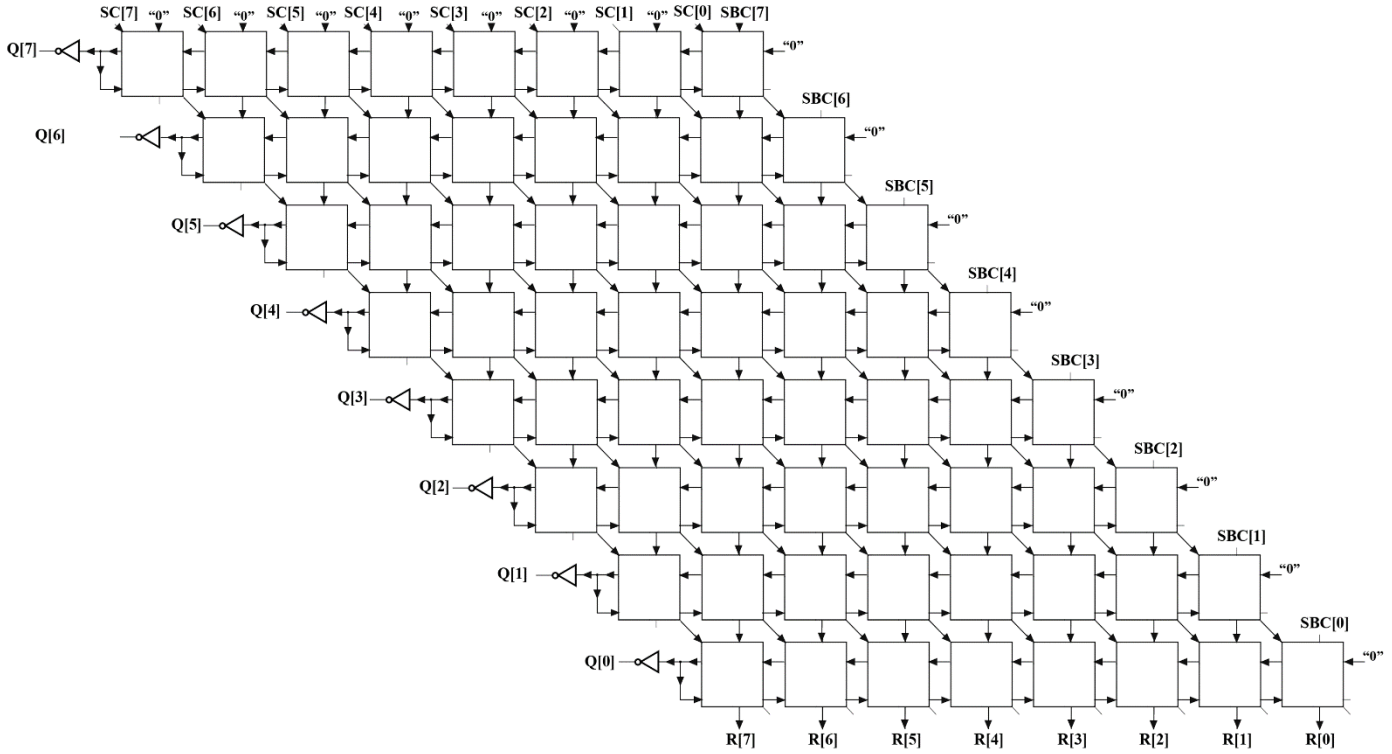
Hình 16: sơ đồ khối Mux 2 sang 1

Bảng 6: Bảng trạng thái Mux 2 to 1

sel	in0	in1	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Dựa vào bảng trạng thái ta mô tả lại bằng Verilog.

3.4.4. Sơ đồ khối mạch chia



Hình 17: Cấu trúc mạch chia 8 bit dùng khối PU sử dụng mạch trừ toàn phần

Vì là mạch chia 8 bit nên sẽ có tổng cộng 8 hàng tượng trưng cho 8 bit số bị chia (SBC) đầu vào và 8 bit thương (Q) đầu ra. Mỗi hàng sẽ gồm 8 khối PU được ghép nối với nhau tượng trưng cho 8 bit số chia (SC) đầu vào ở hàng đầu tiên và 8 bit số dư (R) đầu ra ở hàng cuối cùng.

Đầu vào của từng khối PU sẽ là từng bit của số bị chia và số chia. Mỗi khối PU sẽ lần lượt xử lý các bit đầu vào này. Khác với việc sử dụng bộ cộng toàn phần, khi sử dụng mạch trừ toàn phần, các bit của số chia không cần phải qua cổng đảo trước khi thực hiện phép trừ với số bị chia. Thay vào đó, phép trừ được thực hiện trực tiếp giữa số bị chia (SBC) và số chia (SC). Do đây là phép trừ trực tiếp, đầu vào "Bi" của khối PU đầu tiên trong mỗi hàng sẽ được đặt ở mức logic '0'.

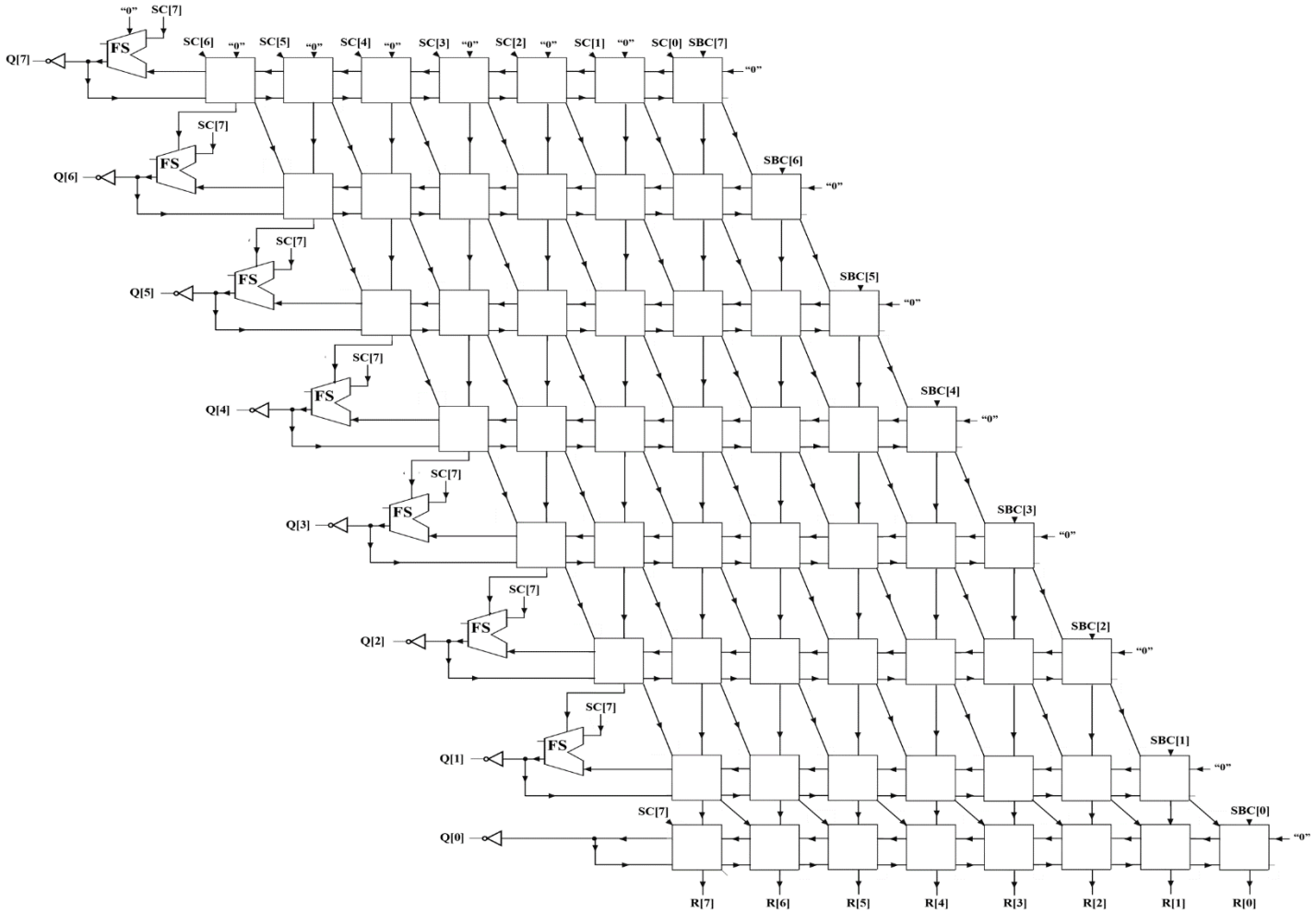
Các đầu ra "Bo" của từng khối PU trên một hàng sẽ nối tiếp vào "Bi" của khối PU tiếp theo thành một mạch trừ 8 bit toàn phần cho đến khối PU cuối cùng (PU thứ 8),

đầu ra “Bo” sẽ là bit thương (Q) khi đưa qua cổng đảo do đã nói ở trên khi chia được (Bo = 0), sau khi đảo Bo, ta sẽ có bit thương Q = 1, ngược lại không chia được (Bo=1) thì đưa qua cổng đảo ta có Q =0, đồng thời Bo sẽ đóng vai trò là chân sel chọn ngõ ra “out” của khối Mux 2 sang 1 bên trong tất cả khối PU trên cùng một hàng đó. Nếu chia được, kết quả của phép trừ 8 bit sẽ được sử dụng làm số dư, nếu không chia được, các số bị chia (SBC) sẽ được giữ lại làm số dư.

Các hàng được sắp xếp chéo và lệch nhau 1 khối PU để thể hiện thao tác chia và dịch thêm một bit của số bị chia (SBC) vào để chia tiếp sau mỗi bước chia, cho đến khi hết 8 bit. Các số dư sau khi chia sẽ được chuyển xuống hàng tiếp theo để tiếp tục chia. Ở hàng đầu tiên, chỉ có khối PU đầu tiên (bên phải) nhận SBC[7] vào ngõ A, còn các khối PU còn lại sẽ nhận giá trị 0. Điều này thể hiện việc chỉ chia bit đầu tiên của SBC, sau đó dần dần dịch các bit còn lại của SBC vào để tiếp tục chia cho đến hết 8 bit.

Mạch chia sẽ thực hiện các thao tác trên cho đến khi đến hàng PU cuối cùng thì đầu ra “out” của các khối PU sẽ là số dư (R) của phép chia. Thương (Q) cuối cùng sẽ là các bit đầu ra “Bo” của các khối PU cuối cùng bên trái mỗi hàng, bit MSB ở hàng trên cùng cho đến bit LSB ở hàng cuối cùng.

3.4.5. Tối ưu sơ đồ khối



Hình 18: Cấu trúc mạch chia 8 bit dùng khối PU sử dụng mạch trừ toàn phần tối ưu khối đa hợp 2 sang 1

Nhận thấy ở những khối PU cuối cùng bên trái của mỗi hàng trừ hàng cuối cùng ra, đầu ra "out" của khối mux 2 sang 1 không được nối làm giá trị đầu vào cho các khối PU ở hàng tiếp theo phía dưới nên chúng ta sẽ tiến hành lược bỏ bớt khối mux 2 sang 1 để đánh giá xem mạch chia có còn cho ra đúng kết quả như mong muốn và thời gian delay sẽ thay đổi như thế nào so với thiết kế ban đầu.

Sau khi lược bỏ khối mux 2 sang 1 ở những khối PU cuối cùng của mỗi hàng thì chỉ còn loại khối trừ toàn phần FS, chúng ta chỉ quan tâm đến đầu ra "Bo" của khối FS. Các thao tác thực hiện của mạch chia này cũng tương tự như trình bày ở phần trên,

đầu ra “Bo” của khối FS cuối cùng mỗi hàng sẽ ghi nhận bit thương cũng như chân sel chọn giá trị đầu ra cho các khối PU trên cùng 1 hàng.

Ở hàng cuối cùng, vì đầu ra “out” của mỗi khối PU là một bit của số dư (R) phép chia nên sẽ giữ nguyên khối PU cuối cùng bên trái.

4. KẾT QUẢ THỰC HIỆN

4.1. Chia 2 số nguyên 8 bit không dấu

Sau khi tiến hành mô tả phần cứng các mô hình đã trình bày phía trên với ngôn ngữ Verilog bằng phần mềm Vivado và sử dụng chính phần mềm này để mô phỏng hoạt động của từng mô hình, ta thu được những kết quả sau. Tất cả các mô phỏng và Synthesis đều được thực hiện kit phát triển Kintex-7 xc7k70tfbg484-3.

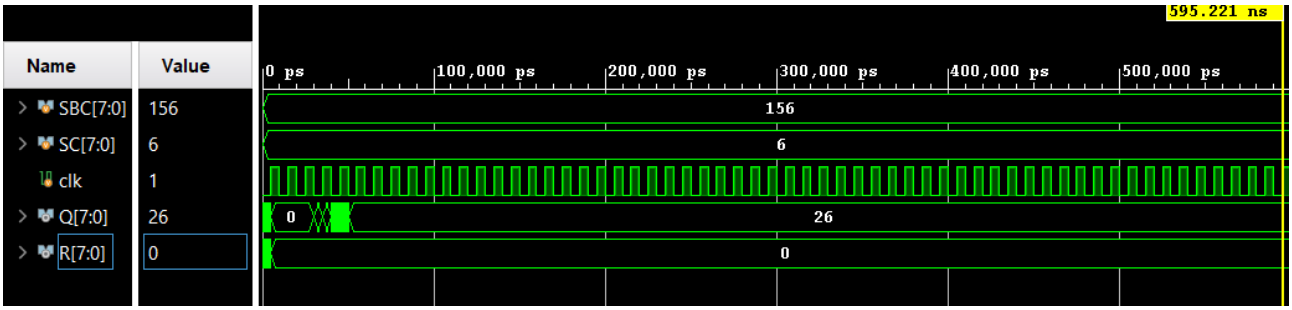
4.1.1 Mạch chia dùng máy trạng thái hữu hạn FSM (Finite State Machine)

- Testcase 1:

Thiết lập: Số bị chia: 156, Số chia: 6

Kỳ vọng: Thương: 26, Dư : 0

Kết quả thực hiện:



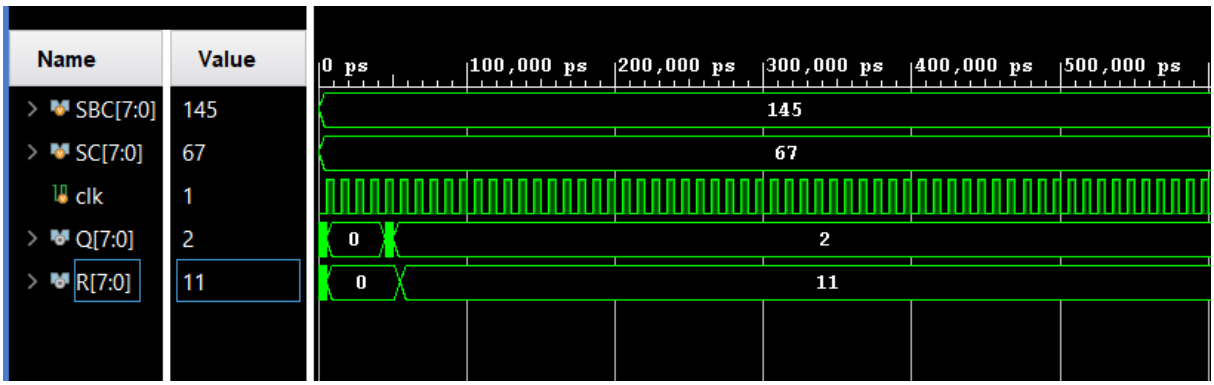
Hình 19: Sơ đồ dạng sóng mô tả kết quả thực hiện của mạch chia dùng FSM với testcase 1

- Testcase 2:

Thiết lập: Số bị chia: 145, Số chia: 67

Kỳ vọng: Thương: 2, Dư : 11

Kết quả thực hiện:



Hình 20: Sơ đồ dạng sóng mô tả kết quả thực hiện của mạch chia dùng FSM với testcase 2

4.1.2 Mạch chia khôi phục số nhớ dùng khối PU sử dụng mạch cộng toàn phần

Thực hiện đồng thời 4 testcase lần lượt:

- Testcase 1:

Thiết lập: Số bị chia: 243, Số chia: 3

Kỳ vọng: Thương: 81, Dư: 0

- Testcase 2:

Thiết lập: Số bị chia: 255, Số chia: 5

Kỳ vọng: Thương: 51, Dư: 0

- Testcase 3:

Thiết lập: Số bị chia: 255, Số chia: 255

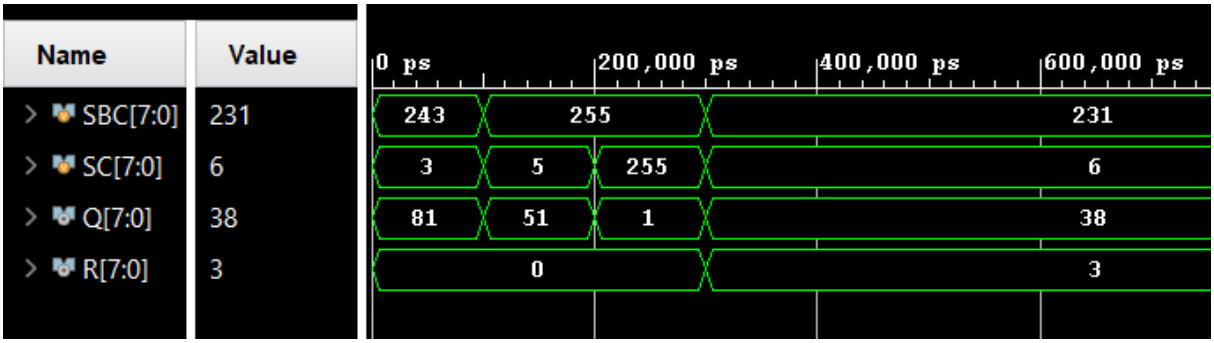
Kỳ vọng: Thương: 1, Dư: 0

- Testcase 4:

Thiết lập: Số bị chia: 231, Số chia: 6

Kỳ vọng: Thương: 38, Dư: 3

Kết quả thực hiện:



Hình 21: Sơ đồ dạng sóng mô tả kết quả thực hiện của Mạch chia khôi phục số nhớ dùng khối PU sử dụng mạch cộng toàn phần

4.1.3 Mạch chia khôi phục số nhớ dùng khối PU sử dụng mạch cộng toàn phần tối ưu khối đa hợp 2 sang 1

Tiếp tục thực hiện đồng thời 4 testcase lần lượt:

- Testcase 1:

Thiết lập: Số bị chia: 136, Số chia: 86

Kỳ vọng: Thương: 1, Dư: 47

- Testcase 2:

Thiết lập: Số bị chia: 255, Số chia: 5

Kỳ vọng: Thương: 10, Dư: 5

- Testcase 3:

Thiết lập: Số bị chia: 25, Số chia: 55

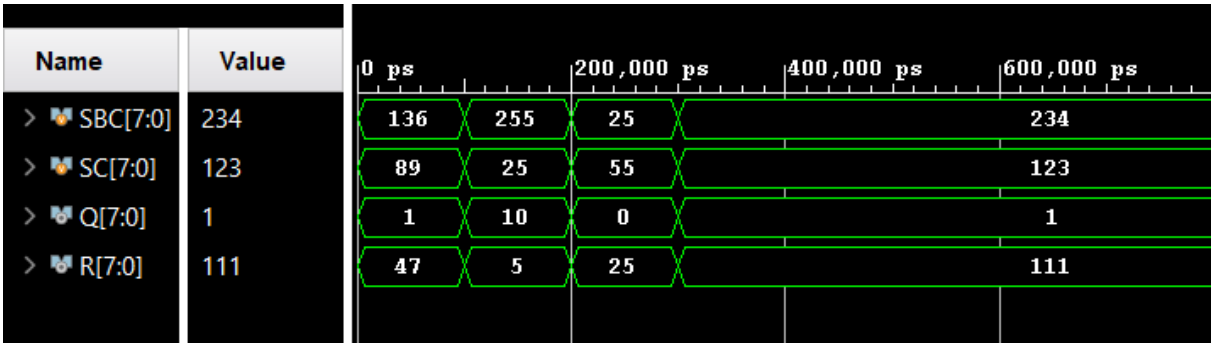
Kỳ vọng: Thương: 0, Dư: 25

- Testcase 4:

Thiết lập: Số bị chia: 234, Số chia: 123

Kỳ vọng: Thương: 1, Dư: 111

Kết quả thực hiện:



Hình 22: Sơ đồ dạng sóng mô tả kết quả thực hiện của mạch chia khôi phục số nhớ dùng khối PU sử dụng mạch cộng toàn phần tối ưu khối đa hợp 2 sang 1

4.1.4 Mạch chia khôi phục số nhớ dùng khối PU sử dụng mạch trừ toàn phần

Tiếp tục thực hiện đồng thời 4 testcase lần lượt:

- Testcase 1:

Thiết lập: Số bị chia: 45, Số chia: 178

Kỳ vọng: Thương: 0, Dư: 45

- Testcase 2:

Thiết lập: Số bị chia: 215, Số chia: 96

Kỳ vọng: Thương: 3, Dư: 8

- Testcase 3:

Thiết lập: Số bị chia: 33, Số chia: 3

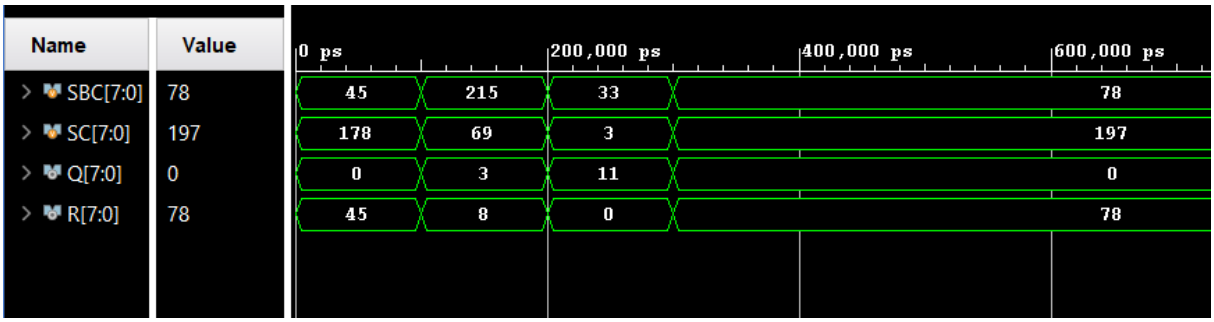
Kỳ vọng: Thương: 11, Dư: 0

- Testcase 4:

Thiết lập: Số bị chia: 78, Số chia: 197

Kỳ vọng: Thương: 0, Dư: 78

Kết quả thực hiện:



Hình 23: Sơ đồ dạng sóng mô tả kết quả thực hiện của Mạch chia khôi phục số nhớ dùng khối PU sử dụng mạch trừ toàn phần

4.1.5 Mạch chia khôi phục số nhớ dùng khối PU sử dụng mạch trừ toàn phần tối ưu khối đa hợp 2 sang 1

Tiếp tục thực hiện đồng thời 4 testcase lần lượt:

- Testcase 1:

Thiết lập: Số bị chia: 156, Số chia: 45

Kỳ vọng: Thương: 3, Dư: 21

- Testcase 2:

Thiết lập: Số bị chia: 99, Số chia: 172

Kỳ vọng: Thương: 0, Dư: 99

- Testcase 3:

Thiết lập: Số bị chia: 245, Số chia: 78

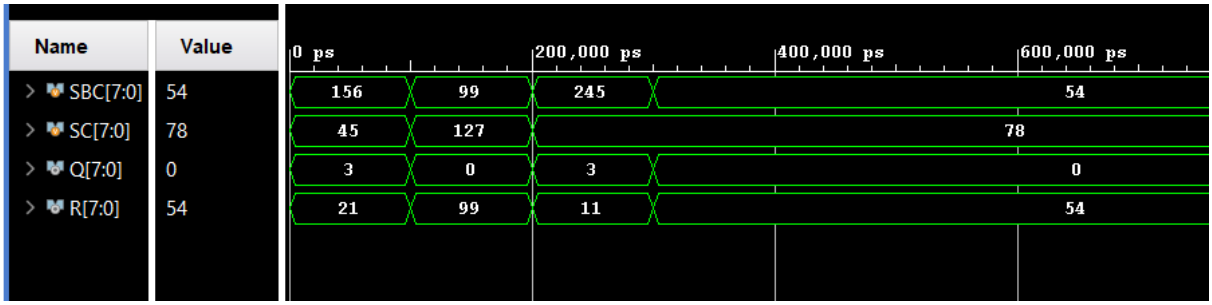
Kỳ vọng: Thương: 3, Dư: 11

- Testcase 4:

Thiết lập: Số bị chia: 54, Số chia: 78

Kỳ vọng: Thương: 0, Dư: 54

Kết quả thực hiện:



Hình 24: Sơ đồ dạng sóng mô tả kết quả thực hiện của mạch chia khôi phục số nhớ dùng khối PU sử dụng mạch cộng toàn phần tối ưu khối đa hợp 2 sang 1

4.1.6 Tổng kết

Qua các testbench của từng mô hình, ta có thể thấy kết quả của các mô hình phép chia cho ra đúng với các phép chia thực tế. Điều đó đồng nghĩa với việc các mô hình phép chia trên đã thực hiện đúng chức năng chia các số 8 bit không dấu có phạm vi thập phân từ (0 - 255).

4.2. Tổng kết LUT và thời gian delay

Bảng 7: Bảng sánh số lượng LUT và Delay Time giữa các mạch chia dùng phương pháp khác nhau

	FSM	PU dùng FA	PU dùng FA (tối ưu)	PU dùng FS	PU dùng FS (tối ưu)
LUT	45	78	82	92	92
Delay Time (ns)	3.820	15.231	13.873	13.697	13.697

Từ bảng số liệu được đưa ra, ta có thể tiến hành so sánh hiệu suất thực thi của 3 phương pháp chia chính: FSM, PU dùng FA, PU dùng FS và so sánh các phương pháp tối ưu so với nguyên bản.

*FSM (Finite State Machine)

- LUT: 45

- Delay Time: 3.820 ns

- Phân tích: FSM có số lượng LUT thấp nhất và thời gian trễ nhỏ nhất. Điều này cho thấy FSM là lựa chọn tối ưu về tốc độ, giúp hệ thống hoạt động nhanh với độ trễ thấp nhất. Tuy nhiên, nó có thể không đủ mạnh để xử lý các tác vụ phức tạp như PU.

*PU dùng FA (Processing Unit dùng Full Adder)

- LUT: 78

- Delay Time: 15.231 ns

- Phân tích: PU dùng FA có số lượng LUT và thời gian trễ lớn hơn FSM. Điều này cho thấy việc sử dụng FA giúp hệ thống có thể thực hiện các tác vụ phức tạp hơn, nhưng thời gian xử lý lâu hơn.

*PU dùng FA (tối ưu)

- LUT: 82

- Delay Time: 13.873 ns

- Phân tích: Tối ưu PU dùng FA tăng nhẹ số lượng LUT (từ 78 lên 82), nhưng giảm thời gian trễ còn 13.873 ns. Điều này cho thấy tối ưu hóa giúp giảm thời gian xử lý mà chỉ cần tăng nhẹ tài nguyên phần cứng.

* PU dùng FS (Processing Unit dùng Full Subtractor)

- LUT: 92

- Delay Time: 13.697 ns

- Phân tích: PU dùng FS có số lượng LUT cao hơn cả PU dùng FA và FSM, nhưng thời gian trễ lại giảm so với PU dùng FA. Điều này chứng tỏ FS có thể là lựa chọn tốt hơn FA trong việc cân bằng giữa tài nguyên phần cứng và thời gian xử lý.

*PU dùng FS (tối ưu)

- LUT: 92

- Delay Time: 13.697 ns

- Phân tích: Tối ưu PU dùng FS không thay đổi số lượng LUT và thời gian trễ so với bản thường. Điều này có thể cho thấy tối ưu đã đạt đến mức tối đa về hiệu quả mà không cần tăng thêm tài nguyên phần cứng.

Tổng kết:

- Tốc độ nhanh nhất: FSM với thời gian trễ 3.820 ns.

- Tối ưu giữa tốc độ và tài nguyên phần cứng: PU dùng FS (tối ưu) với 92 LUT và thời gian trễ 13.697 ns.

- Cải thiện đáng kể khi tối ưu: PU dùng FA giảm từ 15.231 ns xuống 13.873 ns khi tối ưu.

Qua phân tích này, chúng ta thấy mỗi phương pháp có ưu và nhược điểm riêng, và việc chọn lựa sẽ phụ thuộc vào yêu cầu cụ thể về tốc độ và tài nguyên phần cứng của hệ thống.

5. KẾT LUẬN VÀ PHƯƠNG HƯỚNG PHÁT TRIỂN

Bài báo cáo này đã giúp chúng em hiểu sâu hơn về việc thiết kế và triển khai mạch chia 8 bit đơn giản bằng ngôn ngữ mô tả phần cứng Verilog. Qua quá trình thực hiện, chúng em đã đạt được một số kết quả tích cực:

+ Củng cố kiến thức và kỹ năng: Thông qua đề tài, chúng em đã củng cố và mở rộng kiến thức về kiến trúc máy tính, đặc biệt là bộ xử lý số học ALU và các thuật toán chia số học. Đồng thời, chúng em cũng nâng cao kỹ năng lập trình Verilog và sử dụng các công cụ thiết kế như Vivado và Xilinx ISE.

+ Thiết kế và mô phỏng: Việc sử dụng Verilog để thiết kế và mô phỏng mạch chia giúp chúng em nhận ra tầm quan trọng của việc kiểm tra và tối ưu hóa mạch trước khi triển khai thực tế. Các kết quả mô phỏng cho thấy mạch chia hoạt động chính xác và hiệu quả.

+ Ứng dụng thực tiễn: Đề tài có tính ứng dụng trong lĩnh vực vi xử lý và hệ thống nhúng, nơi các phép chia số học thường xuyên được sử dụng. Mạch chia 8 bit này có thể áp dụng vào các dự án thực tế, góp phần tối ưu hóa và giảm thiểu chi phí thiết kế phần cứng.

+ Tối ưu hóa thiết kế: Chúng em đã tìm hiểu và triển khai các phương pháp tối ưu hóa thiết kế mạch, bao gồm việc sử dụng khối PU với mạch cộng toàn phần (FA-PU) và mạch trừ toàn phần (FS-PU), cũng như việc loại bỏ các khối Mux 2 sang 1 tại các khối PU đầu tiên tại các hàng PU từ hàng 1 đến hàng 7 của thiết kế (tối ưu cả FA-PU và FS-PU). So sánh hai thiết kế này và bản tối ưu của nó giúp chúng em hiểu rõ hơn về các yếu tố ảnh hưởng đến hiệu suất thực thi của mạch.

Phương hướng phát triển:

Dựa trên những kết quả đạt được, chúng em xin đề xuất một số phương hướng phát triển trong tương lai:

Tiếp tục tìm hiểu và đưa ra giải pháp tốt hơn cho vấn đề tối ưu hiệu suất của các phương pháp chia đã đề ra.

Tiếp tục nghiên cứu và phát triển các mạch chia có độ phức tạp cao hơn, chẳng hạn như mạch chia với số bit lớn hơn hoặc mạch chia có dấu.

Khám phá và áp dụng các kỹ thuật tối ưu hóa khác như sử dụng các phương pháp logic khác, tối ưu hóa bề mặt sử dụng chip, và giảm thiểu thời gian trễ.

Tích hợp mạch chia vào các hệ thống vi xử lý và hệ thống nhúng thực tế để kiểm tra hiệu suất và tính khả thi trong môi trường hoạt động thực tế.

Hợp tác với các nhóm nghiên cứu khác để trao đổi kinh nghiệm, học hỏi lẫn nhau, và cùng nhau phát triển các dự án lớn hơn.

Những kết quả và kinh nghiệm thu được từ đề tài này sẽ là nền tảng cho các dự án nghiên cứu và phát triển trong tương lai. Chúng em hy vọng rằng, với sự nỗ lực không ngừng, nhóm sẽ tiếp tục học hỏi và đạt được nhiều thành công hơn nữa.

6. TÀI LIỆU THAM KHẢO

[1] James E. Stine, *Digital Computer Arithmetic Datapath Design Using Verilog HDL*, Springer, 2003.

[2] All About Electronics, *How to Design a Binary Division Circuit ? Binary Division Circuit Explained (with Simulation)*

Link: https://www.youtube.com/watch?v=Wf_1mf6yCoc&t=707s

[3] KPUTE, *Thiết kế vi mạch - Thiết kế bộ bộ xử lý số học ALU-8bit -Cadence*

Link: https://www.youtube.com/watch?v=3rf5bA7-BpE&t=1289s

[4] Tutorialspoint, *Restoring Division Algorithm for Unsigned Integer*

Link: https://www.youtube.com/watch?v=PzV6gYpVLuc&t=1s

[5] Engineering Funda, *Restoring Division Algorithm*

Link: https://www.youtube.com/watch?v=Ek-Ea2NnjeQ

[6] David A. Patterson, John L. Hennessy, *4th edition – Computer Organization and Design: The Hardware/Software Interface.*

[7] Charles H. Roth Jr, *Fundamentals of Logic Design*, Cengage Learning, 2018.

Link: https://uomustansiriyah.edu.iq/media/lectures/5/5_2018_12_17!07_25_39_PM.pdf