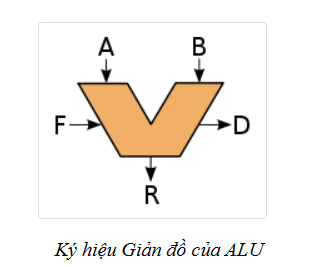
I.LÝ THUYẾT

ALU:



- Trong CPU, một bộ số học và logic (ALU - Arithmetic logic unit) là một mạch điện tử thực hiện phép tính số học và logic.

- ALU là viết tắt của "Arithmetic Logic Unit" (đơn vị tính toán logic), là một thành phần chính trong bộ xử lý của máy tính. ALU được sử dụng để thực hiện các phép tính số học như cộng, trừ, nhân, chia và các phép tính logic như AND, OR và NOT. Nó cũng có khả năng thực hiện các phép tính so sánh như xác định bằng, khác, lớn hơn hoặc nhỏ hơn. ALU có thể hoạt động nhanh và chính xác, và là một phần rất quan trọng trong việc thực hiện các thao tác tính toán của máy tính.

- Cấu trúc của ALU bao gồm các bộ đếm, bộ truyền đổi dữ liệu, các bộ kích hoạt và bộ tính toán logic. ALU có thể được thiết kế với nhiều kích thước khác nhau, từ nhỏ đến lớn, tùy thuộc vào yêu cầu của bộ xử lý.

- ALU kết hợp với các thành phần khác trong bộ xử lý để thực hiện các tác vụ tính toán phức tạp. Ví dụ, khi bạn thực hiện một phép tính toán trên máy tính, dữ liệu được lưu trữ trong bộ nhớ, sau đó ALU được sử dụng để thực hiện phép tính đó và kết quả được trả về cho bộ nhớ hoặc đưa vào bộ đệm. ALU cũng có thể được sử dụng để thực hiện các phép tính logic và phép toán trên dữ liệu được lưu trữ trong thanh ghi (register) của CPU.

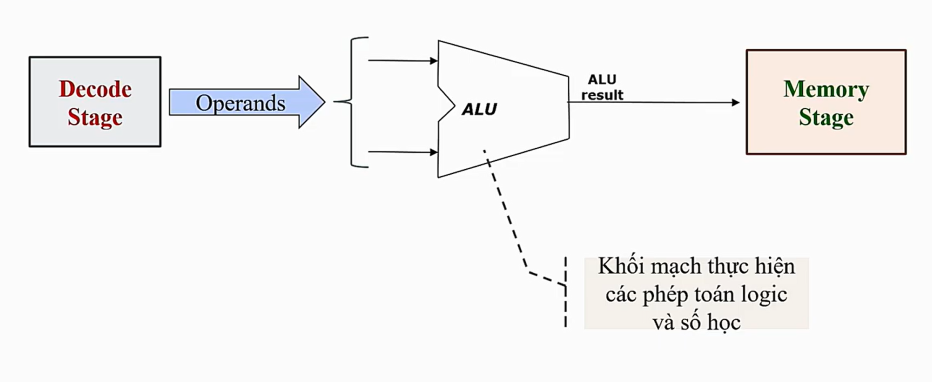
- Một ALU đơn giản sẽ bao gồm 2 phần là khối AU (Arithmetic Unit) chịu trách nhiệm thực hiện các tác vụ về toán học và khối LU (Logic Unit) chịu trách nhiệm thực hiện các tác vụ về logic.

- ALU (Arithmetic Logic Unit) bao gồm hai khối chính:

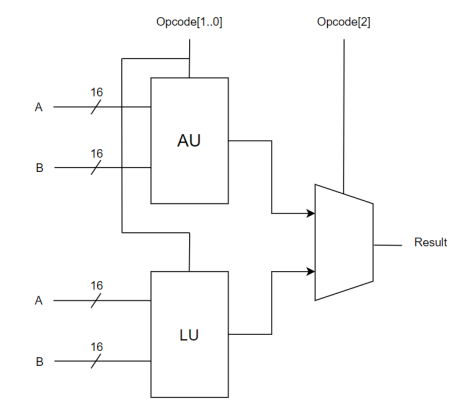
1. Khối tính toán số học (Arithmetic Unit): Thực hiện các phép tính số học cơ bản như cộng, trừ, nhân, chia và các phép tính liên quan đến số học.

2. Khối tính toán logic (Logic Unit): Thực hiện các phép tính logic cơ bản như AND, OR, NOT và các phép toán liên quan đến logic.

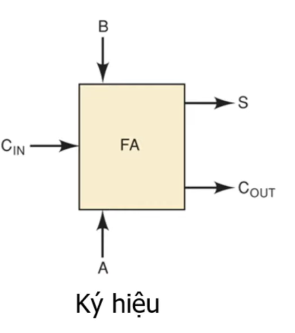
=> Những khối này hoạt động song song để thực hiện các phép tính số học và logic phức tạp. Các giá trị đầu vào được đưa vào ALU thông qua bộ truyền đổi dữ liệu và các kết quả được đưa ra qua bộ đệm kết quả. ALU được sử dụng trong các bộ xử lý để thực hiện các phép tính toán và các thao tác logic cần thiết để điều khiển hoạt động của máy tính.



- Hình ảnh ví dụ chi tiết về ALU (16 bit) :



FULL ADDER:



- Full adder là một mạch kết hợp được sử dụng để thực hiện phép tính cộng hai số nhị phân, bao gồm cả carry-in (nếu có) từ một bộ đếm trước đó. Một full adder có 3 đầu vào: hai số nhị phân và carry-in, và 2 đầu ra: tổng và carry-out.

- Chức năng của full adder là thực hiện phép tính cộng hai số nhị phân và carry-in, để tính toán tổng của hai số đó và trả về kết quả. Carry-out được sử dụng như một carry-in cho một full adder khác trong trường hợp các số đang được cộng là số lớn hơn 1 bit. Cụ thể, carry-out được sử dụng để tính toán tổng của các bit tiếp theo của các số nhị phân.

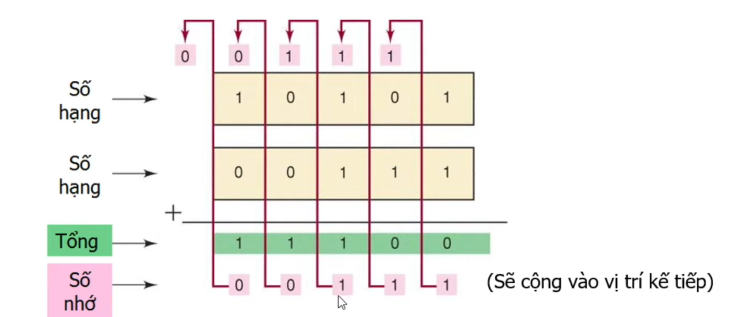
- Full adder có thể được sử dụng trong các ứng dụng liên quan đến việc xử lý số học trong máy tính, bao gồm việc xây dựng các bộ đếm, bộ chuyển đổi nhị phân sang thập phân và ngược lại, hay trong các ứng dụng liên quan đến mã hóa và giải mã thông tin.

- Full adder cộng những số 2 hoặc nhiều bit :

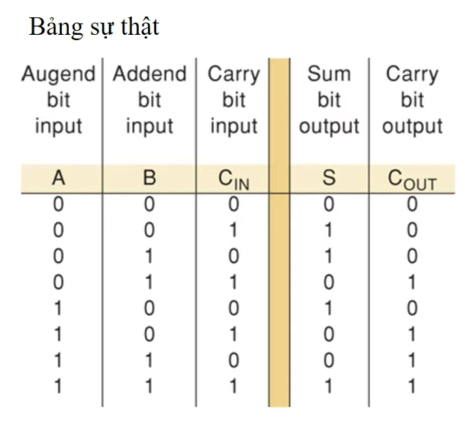
+ Cộng từng cặp bit bình thường

+ Nhưng ở vị trí cặp bit i, có thể có carry-in từ bit i-1

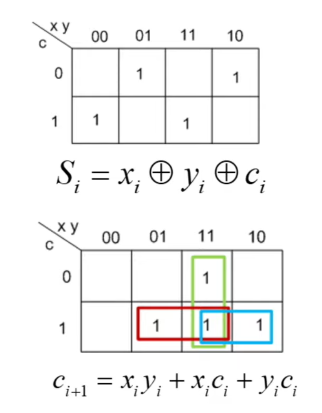
- Ví dụ cộng 2 số nhị phân :



- Bảng sự thật của Full Adder:

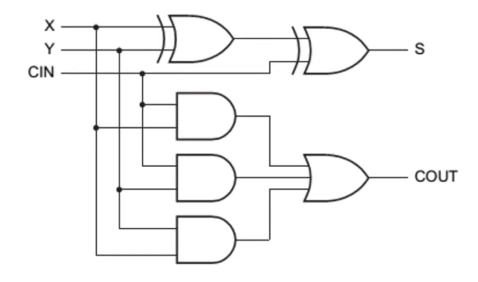


- Karnaugh map từ bảng sự thật bên trên :

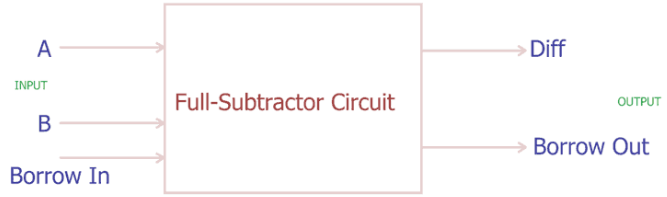


Trong đó C(i) = C(in) , C(i+1) = C(out)

- Từ trên ta có thiết kế chi tiết về Full Adder (2 bit) :



FULL SUBTRACTOR:



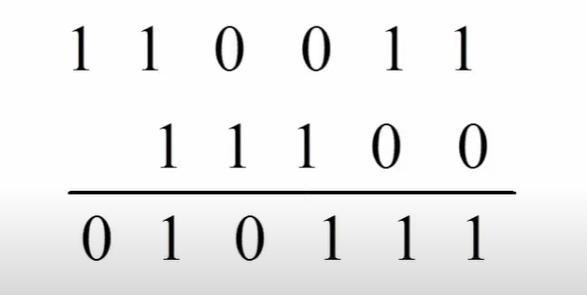
Ký hiệu full subtractor

- Full subtractor là một mạch kết hợp được sử dụng để thực hiện phép tính trừ hai số nhị phân, bao gồm cả borrow-in (nếu có) từ một bộ trừ trước đó. Một full subtractor có 3 đầu vào: hai số nhị phân và borrow-in, và 2 đầu ra: hiệu và borrow-out.

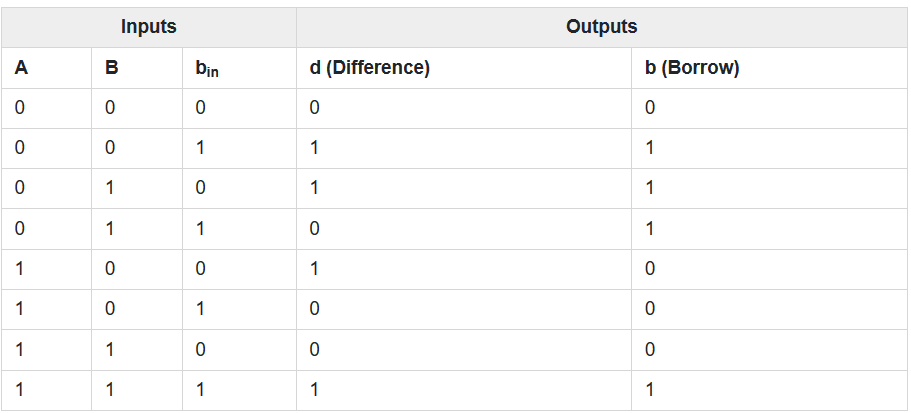
- Chức năng của full subtractor là thực hiện phép tính trừ hai số nhị phân và borrow-in, để tính toán hiệu của hai số đó và trả về kết quả. Borrow-out được sử dụng như một borrow-in cho một full subtractor khác trong trường hợp các số đang được trừ là số lớn hơn 1 bit. Cụ thể, borrow-out được sử dụng để tính toán hiệu của các bit tiếp theo của các số nhị phân.

- Full subtractor có thể được sử dụng trong các ứng dụng liên quan đến việc xử lý số học trong máy tính, bao gồm việc xây dựng các bộ trừ, bộ chuyển đổi nhị phân sang thập phân và ngược lại, hay trong các ứng dụng liên quan đến mã hóa và giải mã thông tin.

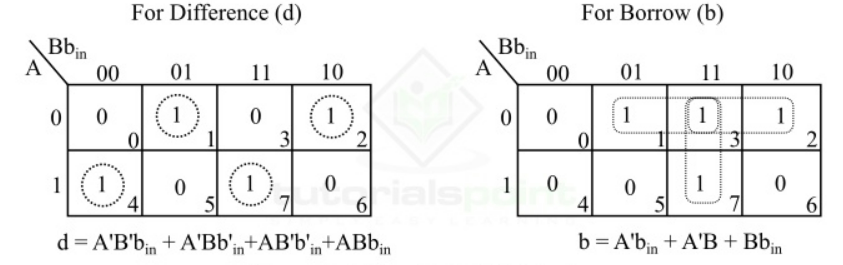
- Ví dụ trừ 2 số nhị phân :



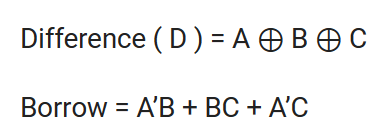
- Bảng sự thật của Full Subtractor :



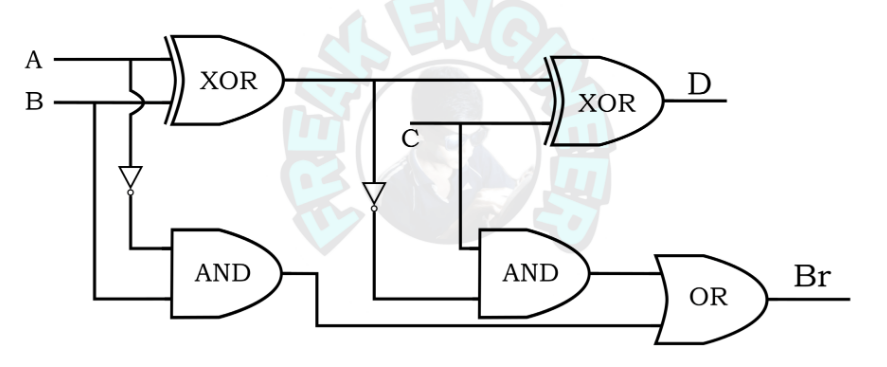
- Karnaugh map từ bảng sự thật bên trên :



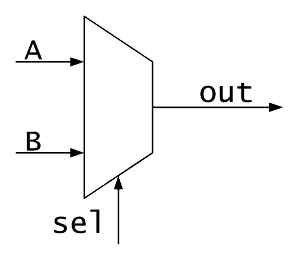
Rút gọn :



- Từ trên ta có thiết kế chi tiết về full subtractor (2 bit):



MUX :

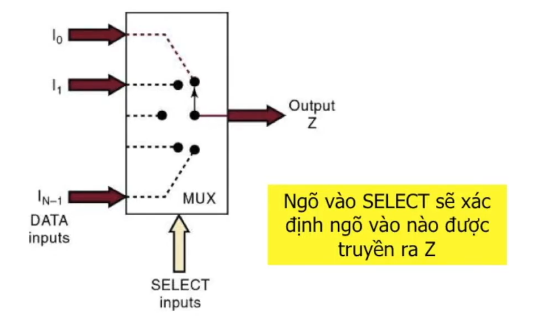


Ký hiệu của MUX

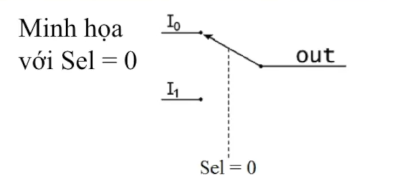
- Mux là một mạch logic đơn giản, viết tắt của từ "Multiplexer". Mux có chức năng chọn một trong nhiều tín hiệu đầu vào và đưa ra tín hiệu duy nhất đầu ra. Nó có nhiều đường đầu vào và một đường đầu ra, và đường điều khiển để chọn đường vào nào sẽ được đưa ra đầu ra. Đường điều khiển thường được gọi là "select line".

- Mux có thể được thực hiện bằng cách sử dụng nhiều cổng logic AND và OR. Ví dụ, một Mux 4 đường vào có thể được thực hiện bằng cách sử dụng 4 cổng AND 2 đầu vào và 2 cổng OR 2 đầu vào. Đường điều khiển được sử dụng để kết nối các đầu vào tới các cổng AND và OR.

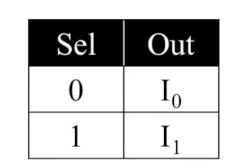
- Chức năng của Mux là đưa ra một tín hiệu duy nhất đầu ra từ nhiều tín hiệu đầu vào, dựa trên giá trị đường điều khiển. Nó thường được sử dụng trong các ứng dụng điều khiển, bộ nhớ và các ứng dụng logic khác trong các mạch điện tử và máy tính. Mux cũng được sử dụng để chuyển đổi các định dạng tín hiệu hoặc chuyển đổi giữa các tín hiệu analog và digital.



- Ví dụ minh họa với Mux 2-1 (1bit) với select = 0:



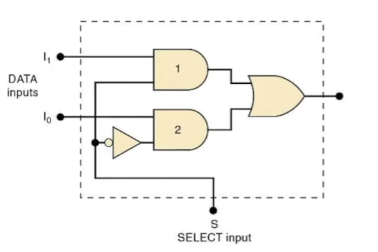
- Bảng sự thật của MUX:



=> Biểu thức đại số :

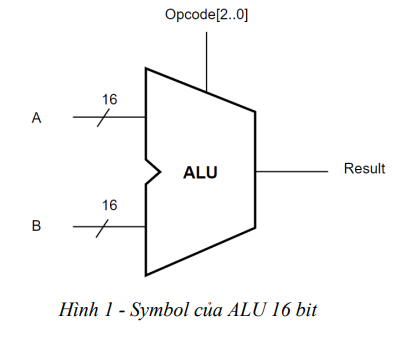


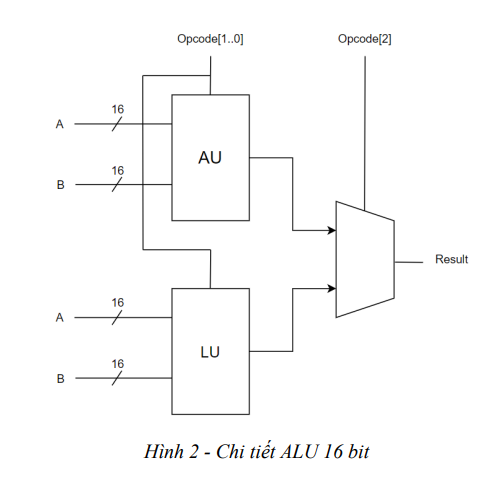
- Chi tiết thiết kế về MUX 2-1(1bit) :



II. THỰC HÀNH

Sinh viên thực hiện thiết kế và mô phỏng một ALU có 2 toán hạng (16 bit) và các phép toán cộng, cộng 1, trừ, trừ 1, and, or, nand, xor theo đúng thứ tự tương ứng với tín hiệu điều khiển (Opcode) từ 0→7.



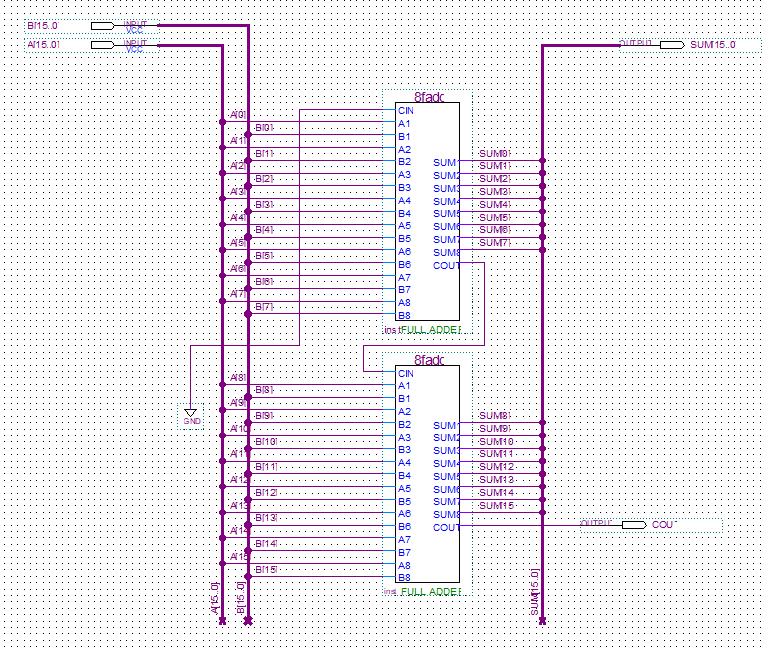


- Từ opcode 0 -> 7 ta thực hiện như sau :

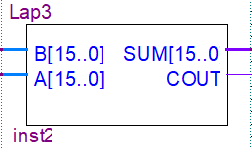
|  |  |  |
| --- | --- | --- |
| opcode |  |  |
| 000 | A + B | Full adder (16bit) |
| 001 | A + 1 |
| 010 | A – B | Full subtractor (16bit) |
| 011 | A – 1 |
| 100 | A and B | And (16bit) |
| 101 | A or B | Or(16bit) |
| 110 | A xor B | Xor(16bit) |
| 111 | A nand B | Nand(16bit) |

**FULL ADDER :**

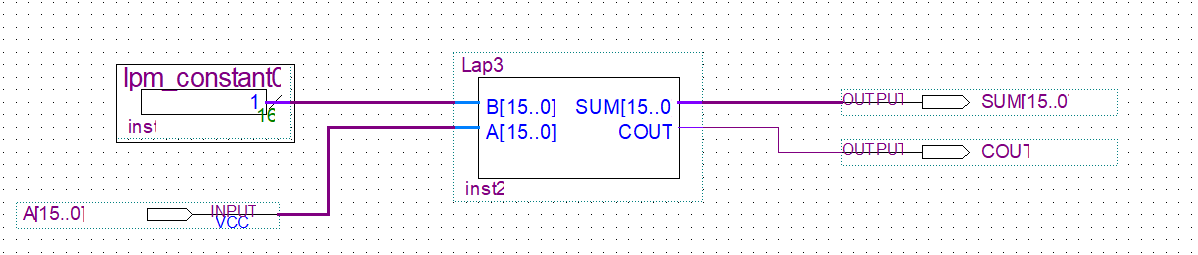
- Ta vẽ mạch Full Adder cho A + B bằng quartus :



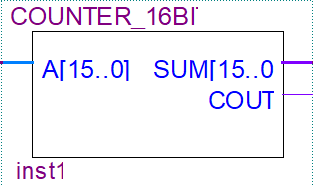
- Đóng gói full adder 16 bit :



- Ta vẽ mạch Counter ( A + 1 ) 16 bit từ Full adder 16 bit :



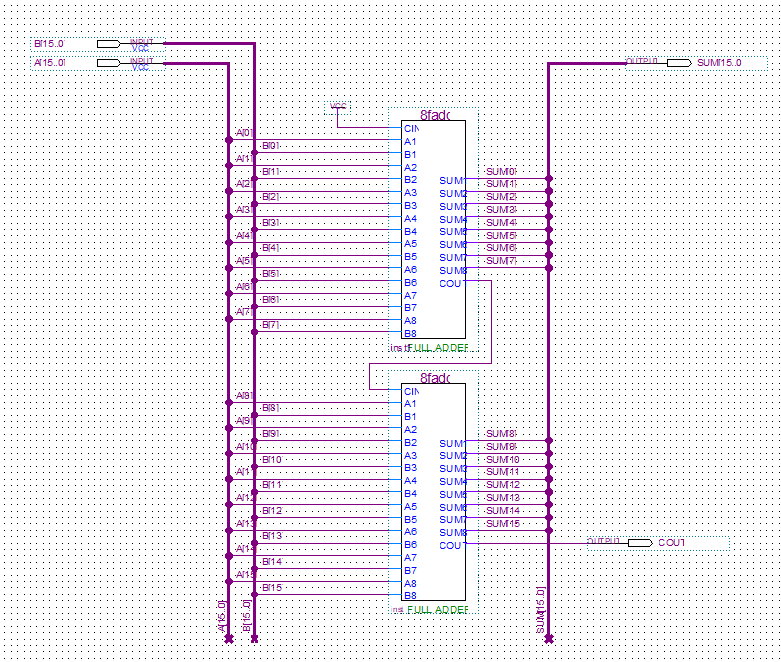
- Đóng gói mạch Counter (A +1 ):



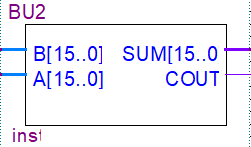
**FULL SUBTRACTOR:**

- Ta vẽ mạch (A + B + 1) 16 bit từ full adder 16 bit bằng cách thay chân cin từ GND thành VCC

Giải thích : Nếu chân CIN gắn GND thì kết quả A + B + 0 còn chân CIN gắn VCC thì kết quả A + B + 1 ( A xor B xor C và C = AC + AB + BC )

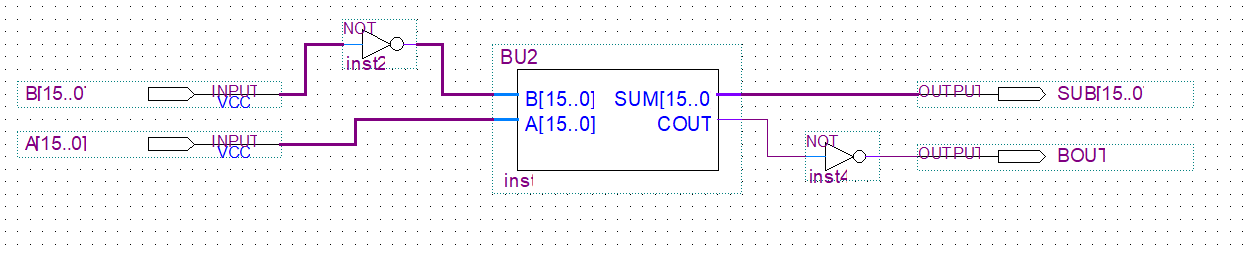


- Đóng gói mạch A+B+1 16 bit:

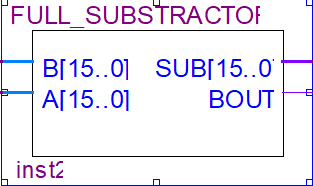


- Ta vẽ mạch full subtractor (A + B’ + 1) 16 bit dựa trên A + B +1 16 bit

Giải thích : để tạo mạch A – B thì ta phải lấy A cộng cho bù 2 của B, như rút gọn biểu thức karnaugh ( A xor B’ xor C và C = B’A + B’C + AC )



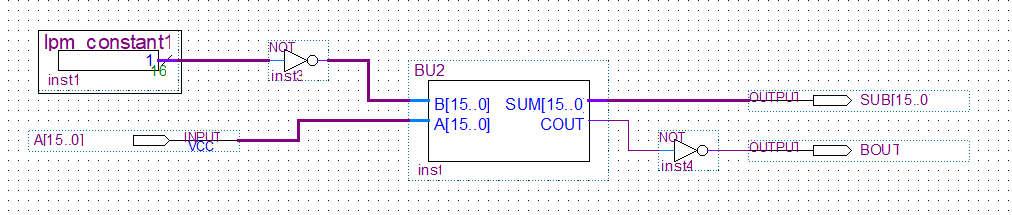
- Đóng gói mạch full subtractor ( A + B’ + 1 ) 16 bit :



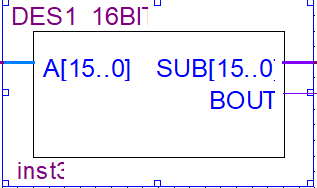
- Ta vẽ mạch DESCENT ( A -1 ) từ mạch ( A + B + 1 )

Giải thích: biến biểu thức A + B + 1 thành A + B’ + 1 rồi gán B = 1 ta sẽ được

A - 1

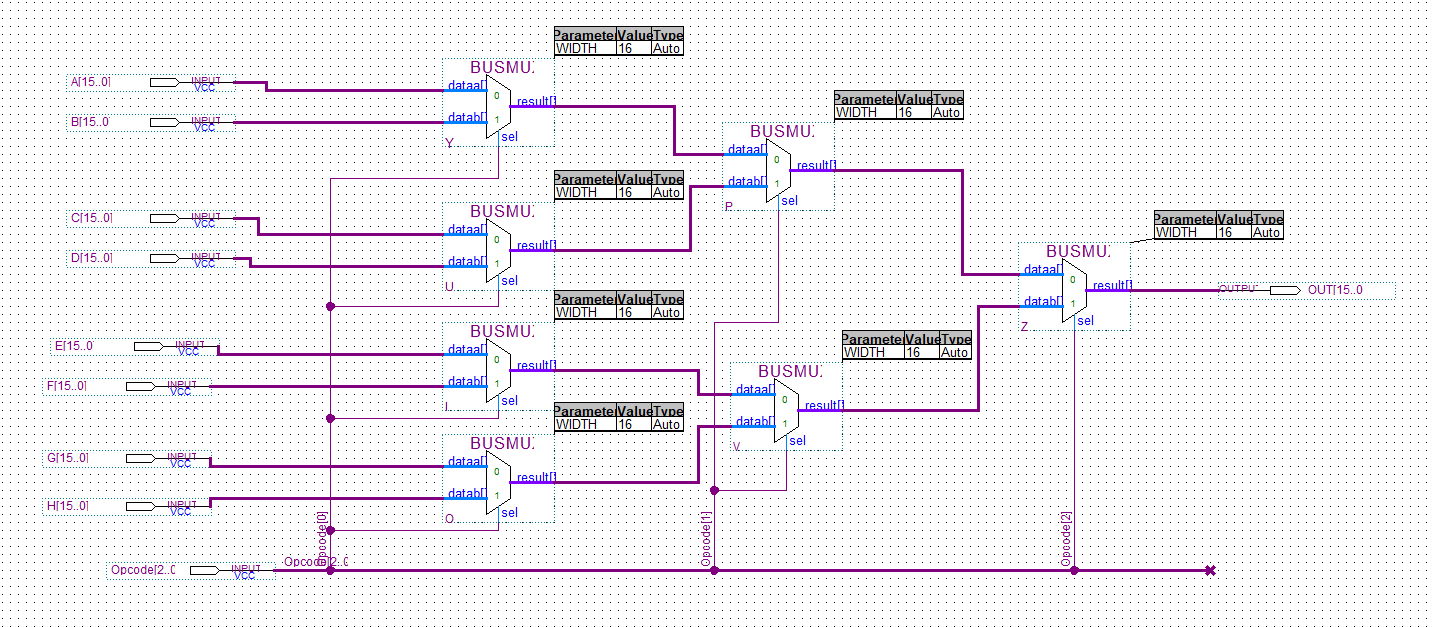


- Đóng gói mạch DESCENT ( A -1 ) :

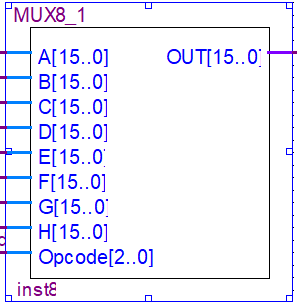


**MUX**

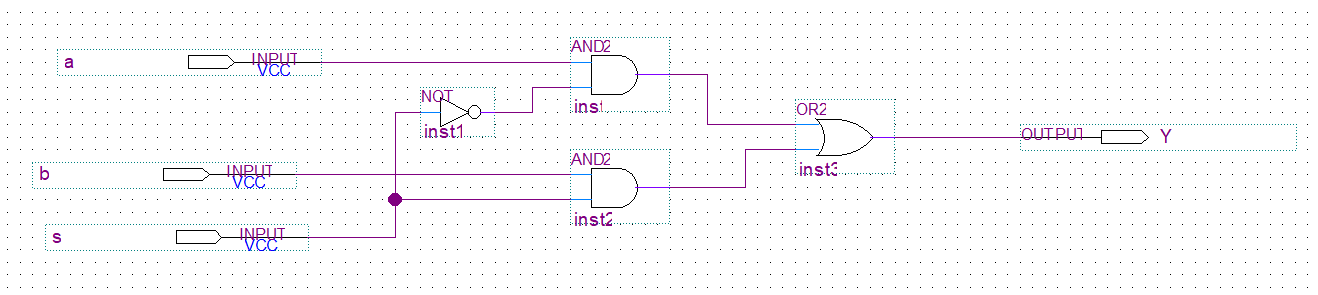
- Ta thiết kế MUX8-1 16 bit bằng bus-mux :



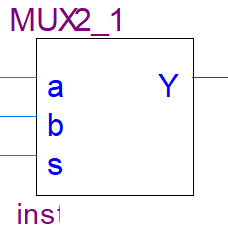
- Đóng gói mạch MUX8-1 16BIT :



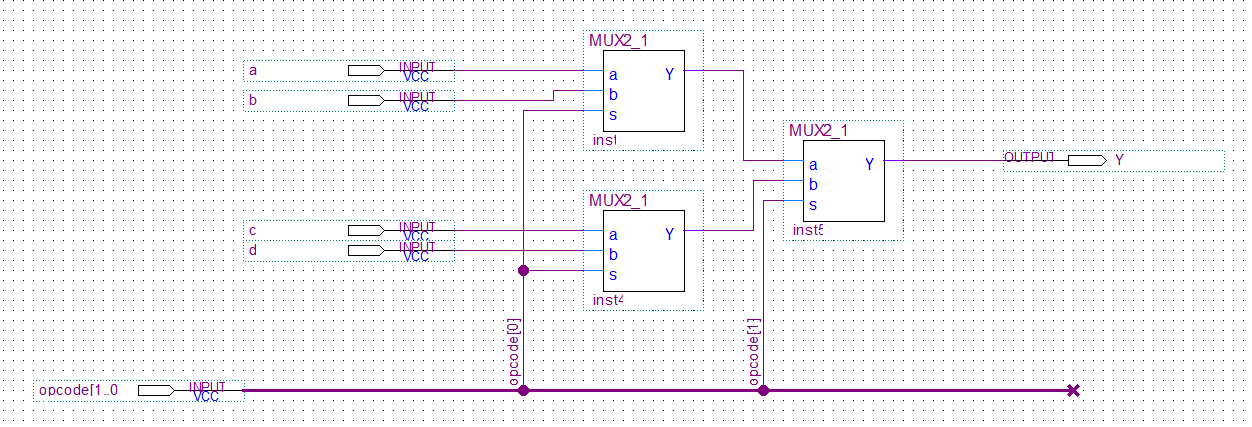
- Ta thiết kế MUX2-1 1bit :



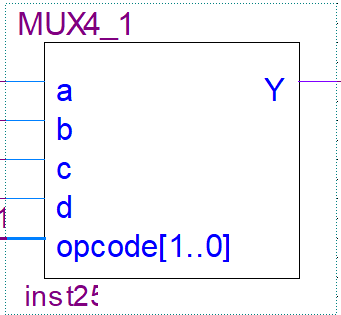
- Ta đóng gói mạch MUX2-1 1bit :



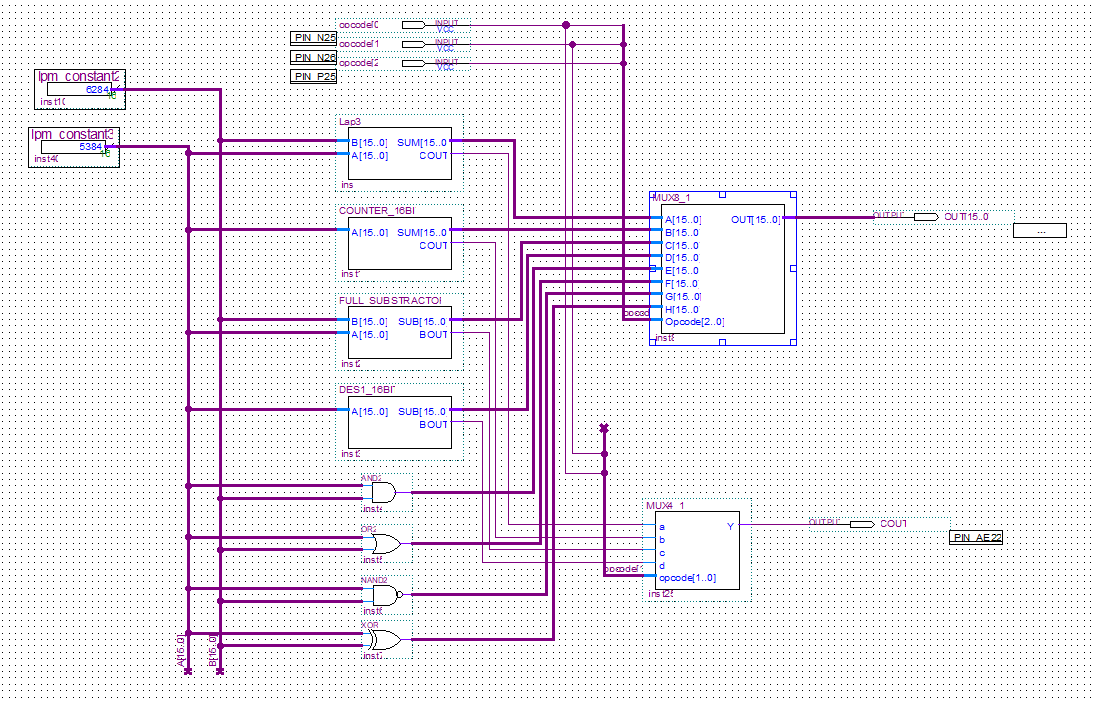
- Ta thiết kế MUX4-1 1bit từ MUX2-1 1bit :



- Ta đóng gói MUX4-1 :

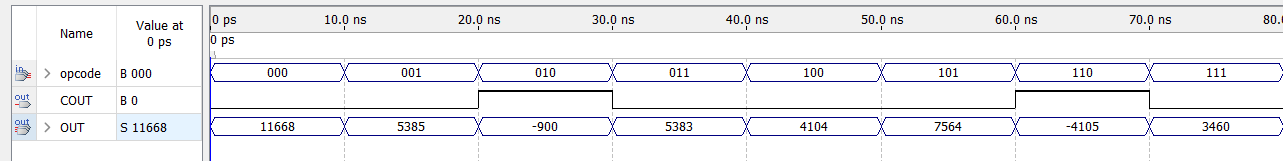


**MẠCH ALU HOÀN CHỈNH**

****

- Chạy waveform trong đó

A : 5384 = 0001 0101 0000 1000 ; B : 6284 = 0001 1000 1000 1100



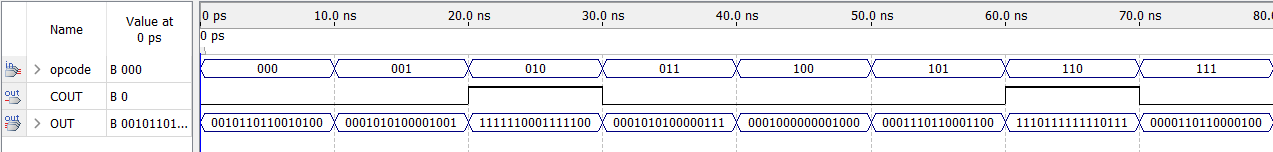
Kiểm tra kết quả Full adder và full subtractor (chỉnh về signed decimal để kiểm tra) :

A + B = 11668 ( đúng )

A + 1 = 5385 ( đúng )

A – B = -900 ( đúng ) , COUT có kích cạnh lên do có mượn bit

A - 1 = 5383 ( đúng )



Kiểm tra kết quả and , or , nand , xor :

A = 0001 0101 0000 1000 (5384)

B = 0001 1000 1000 1100 (6284)

And = 0001 0000 0000 1000 ( đúng )

Or = 0001 1101 1000 1100 ( đúng )

Nand= 1110 1111 1111 0111 ( đúng )

Xor = 0000 1101 1000 0100 ( đúng )

LINK GOOGLE DRIVE CHẠY ALU (16BIT) TRÊN KIT :

<https://drive.google.com/drive/u/2/folders/1JHmgACVxwcwbVTKyei3yq2TxBfJjtFrv>

Giải thích chạy ALU trên KIT :

Gán chân opcode[2..0] = SW[2..0]

Gán chân Out[15..0] = LEDR[15..0]

Gán chân Cout = LEDG[0]

- Như trong video thì ta nhìn thấy đèn chân out được biểu thị bằng LEDR có ánh sáng màu đỏ bên trái và sáng từ LEDR[15] đến LEDR[0] biểu thị cho kết quả của các phép tính A – B, A -1 , A + B , A + 1 , A and B, A xor B , A nand B , A xor B.

- Còn đèn sáng xanh lá bên phải là đèn LEDG[0] biểu thị chỉ cho FULL ADDER và FULL SUBTRACTOR mang ý nghĩa là phép tính toán có dư bit hay là mượn bit hay không.

- Đèn LEDR[15..0] sáng khi ta kích SW[2..0] từ 000 đến 111.

- Ta thấy LEDG sáng lên 2 lần 1 lần tại 010 ( A – B ) và 1 lần sáng tại 110 ( A nand B ) . Lý do bởi vì ta làm mạch Cout dựa vào mux4-1 và select là opcode[1..0]

|  |  |  |
| --- | --- | --- |
| Opcode[2] | Opcode[1] | Opcode[0] |
| 0 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| 1 | 1 | 1 |

=> ta chỉ cần dùng opcode[1] và opcode[0] để xác định cho bit mượn và bit dư của 4 phép tính . Tại 110 là vì vô tình A – B phải mượn bit và 2 opcode[1] và opcode[0] là 1 và 0 nên COUT kích cạnh lên.

=> Vậy nên ta không cần quan tâm khi COUT lại sáng lên tại các phép tính AND , OR, NAND , XOR.

=> Đèn LEDR[15..0] sáng khi ta kích SW[2..0] từ 000 đến 111 **đã chạy đúng kết quả với kết quả đã test trên WAVEFORM**