ĐẠI HỌC BÁCH KHOA THÀNH PHỐ HỒ CHÍ MINH

Khoa Điện – Điện tử Bộ môn Điện tử

---------\*\*\*--------



BÁO CÁO XÂY DỰNG MỘT ALU 4 BIT

LAB 2

**GVHD: ThS. Trịnh Vũ Đăng Nguyên**

**Sinh viên thực hiện:**

**Ngô Minh Nhân MSSV: 1813327**

**Hồ Bá Phước MSSV: 1813638**

**TP.HCM, tháng 6 năm 2021**

MỤC LỤC

[1. MỤC TIÊU: 3](#_Toc28017)

[Bảng 1: Bảng chức năng của ALU cần thiết kế 3](#_Toc1054)

[Hình 1: Sơ đồ khối của một ALU 4-bit 3](#_Toc25928)

[2. GIẢI THÍCH NGUYÊN NHÂN CHỌN LỰA GIẢI PHÁP – CÁCH GIẢI QUYẾT 4](#_Toc7337)

[Thiết kế ALU 4](#_Toc18069)

*[ALU 4-bit là 4 ALU 1-bit](#_Toc30357)* [4](#_Toc30357)

[ALU gồm A và L 4](#_Toc17275)

[Hình 2: Sơ đồ khối của một ALU phân theo chức năng 4](#_Toc21386)

[3. CODE DESIGN – CODE TESTBENCH: 5](#_Toc22216)

[a. Code desing 5](#_Toc11201)

[● halfadder.v 5](#_Toc26091)

[● fulladder.v 5](#_Toc19613)

[● mux2to1.v 5](#_Toc22438)

[● logicunit.v 5](#_Toc15643)

[● arithmeticunit.v 6](#_Toc27967)

[● ALU.v 6](#_Toc7868)

[● ALU4bit.v 6](#_Toc23096)

[b. Code testbench 7](#_Toc14043)

[4. Hình ảnh: 9](#_Toc14044)

1. **MỤC TIÊU:**

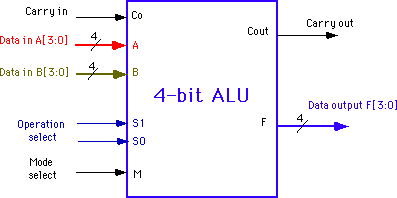
Bộ xử lý logic về toán học (ALU - Arithmetic and Logic Unit) là một mạch tổ hợp để xử lý các tác vụ về logic và toán học dựa trên hai số hạng. Các tác vụ cho ALU thực hiện được điều khiển bằng các ngõ nhập function-select. Mục đích của bài thí nghiệm này là thiết kế một ALU đơn giản như sau:

* Độ dài các toán hạng là 4-bit.
* Các ngõ nhập function-select gồm có: M, S0 và S1.
* Các tác vụ ALU thực hiện được cho trong bảng 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| M | S1 | S0 | Chức năng | Tác vụ |
| 0 | 0 | 0 | Ai.Bi | AND |
| 0 | 0 | 1 | Ai + Bi | OR |
| 0 | 1 | 0 | Ai (+) Bi | XOR |
| 0 | 1 | 1 | ~(Ai (+) Bi) | XNOR |
| 1 | 0 | 0 | A + C0 | Cộng A với Carry |
| 1 | 0 | 1 | A+B+C0 | Cộng A, B và Carry |
| 1 | 1 | 0 | A+B'+C0 | Cộng A với bù B và Carry |
| 1 | 1 | 1 | A'+B+C0 | Cộng B với bù A và Carry |

## Bảng 1: Bảng chức năng của ALU cần thiết kế

Sơ đồ khối của ALU như hình 1



## Hình 1: Sơ đồ khối của một ALU 4-bit

1. **GIẢI THÍCH NGUYÊN NHÂN CHỌN LỰA GIẢI PHÁP – CÁCH GIẢI QUYẾT**

**Thiết kế ALU**

Ta dùng nguyên tắc "chia để trị" để thiết kế ALU. Cách này module hoặc bản thiết kế thành các phần nhỏ hơn, dễ quản lý hơn và có thể tái sử dụng. Cách tiếp cận này giúp mọi thứ có tính hệ thống hơn và có thể phát triển những hệ thống phức tạp.

***ALU 4-bit là 4 ALU 1-bit***

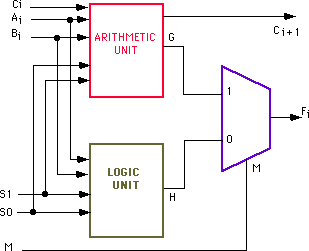
Nếu xét về cấu trúc, ta có thể xem một ALU n-bit được cấu thành từ n ALU 1-bit. Như vậy để thiết kế một ALU 4-bit, ta chỉ cần thiết kế một ALU 1-bit. Sau đó, có thể ghép nối 4 ALU 1-bit này lại với nhau để tạo thành ALU 4-bit. Mỗi ALU 1-bit như vậy được gọi là một bit-slice. Cách này có tính sử dụng lại rất cao và được dùng khá nhiều trong kỹ thuật thiết kế phần cứng, chẳng hạn thiết kế bộ nhớ.

Để thiết kế một bit-slice, có nhiều cách khác nhau. Một cách có thể là viết bảng sự thật để thiết kế. Bảng này có 6 ngõ nhập (M, S1, S0, C0, Ai, Bi) và hai ngõ xuất là Fi và Ci+1. Cách này nếu viết bằng tay thì khá công phu, nhưng hiệu quả. Trong VHDL (Verilog), có thể hiện thực thiết kế này bằng mô hình cấu trúc.

## **ALU gồm A và L**

Nếu xét về chức năng, ta có thể phân ALU thành hai phần chuyên biệt, một về logic và một về toán học. Sau đó, có thể dùng một MUX 2:1 để kết hợp hai khối này. Cách này có ưu điểm là thiết kế từng khối nhỏ sẽ dễ hơn so với thiết kế một bit-slice, vốn

cần thiết kế một ALU hoàn chỉnh. Hình sau thể hiện sơ đồ khối của một bit-slice ALU thực hiện theo ý tưởng này.



## Hình 2: Sơ đồ khối của một ALU phân theo chức năng

1. **CODE DESIGN – CODE TESTBENCH:**
2. **Code desing**

* **halfadder.v**

|  |
| --- |
| module halfadder(a,b,cout,s);  input a,b;  output cout,s;  assign s=a^b;  assign cout=a&b;  endmodule |

* **fulladder.v**

|  |
| --- |
| `include "hafladder.v";  module fulladder(a,b,cin,cout,s);  input a,b,cin;  output cout,s;  wire sum,carry1,carry2;  halfadder h1(a,b,carry1,sum);  halfadder h2(sum,cin,carry2,s);  assign cout=carry1|carry2;  endmodule |

* **mux2to1.v**

|  |
| --- |
| module mux2to1(a,b,s,f);  input a,b,s;  output f;  assign f=s?a:b;  endmodule |

* **logicunit.v**

|  |
| --- |
| module logic\_unit(s1,s0,a,b,f);  input s1,s0,a,b;  output f;  assign f=({s1,s0}==2'b00)?a&b:  ({s1,s0}==2'b01)?a|b:  ({s1,s0}==2'b10)?a^b:  ({s1,s0}==2'b11)?~(a^b):1'b0;  endmodule |

* **arithmeticunit.v**

|  |
| --- |
| `include "fulladder.v";  module arithmetic\_unit(s1,s0,a,b,cin,cout,f);  input s1,s0,a,b,cin;  output cout,f;  wire out\_01, out\_10, out\_11, cout\_01, cout\_10, cout\_11;  fulladder f01(.a(a),.b(b),.cin(cin),.cout(cout\_01),.s(out\_01));  fulladder f10(.a(a),.b(~b),.cin(cin),.cout(cout\_10),.s(out\_10));  fulladder f11(.a(~a),.b(b),.cin(cin),.cout(cout\_11),.s(out\_11));  assign {cout,f}=({s1,s0}==2'b00)?a+cout: ({s1,s0}==2'b01)?{cout\_01,out\_01}:  ({s1,s0}==2'b10)?{cout\_10,out\_10}:  ({s1,s0}==2'b11)?{cout\_11,out\_11}: 2'b00;  endmodule |

* **ALU.v**

|  |
| --- |
| `include "arithmeticunit.v";  `include "mux2to1.v";  `include "logicunit.v";  module ALU1bit(A,B,Cin,S1,S0,M,Cout,F);  input A, B, Cin, S1, S0, M;  output F, Cout;  wire G,H;  logic\_unit LU(S1,S0,A,B,H);  arithmetic\_unit AU(S1,S0,A,B,Cin,Cout,G);  mux2to1 MUX(G,H,M,F);  endmodule |

* **ALU4bit.v**

|  |
| --- |
| `include "ALU.v";  module ALU4bit(a,b,cin,s1,s0,m,cout,f, clk, rst\_n);  input [3:0] a;  input [3:0] b;  input cin, s1, s0, m;  input clk, rst\_n;  output reg [3:0] f;  output reg cout;  wire [2:0] cout\_tmp;  wire [3:0] f\_tmp;  wire cout\_tmp1;  ALU1bit A1(.A(a[0]),.B(b[0]),.Cin(cin),.S1(s1),.S0(s0),.M(m),.Cout(cout\_tmp[0]),.F(f\_tmp[0]));  ALU1bit A2(.A(a[1]),.B(b[1]),.Cin(cout\_tmp[0]),.S1(s1),.S0(s0),.M(m),.Cout(cout\_tmp[1]),.F(f\_tmp[1]));  ALU1bit A3(.A(a[2]),.B(b[2]),.Cin(cout\_tmp[1]),.S1(s1),.S0(s0),.M(m),.Cout(cout\_tmp[2]),.F(f\_tmp[2]));  ALU1bit A4(.A(a[3]),.B(b[3]),.Cin(cout\_tmp[2]),.S1(s1),.S0(s0),.M(m),.Cout(cout\_tmp1),.F(f\_tmp[3]));  always @(posedge clk or negedge rst\_n) begin  if (!rst\_n) begin  f <= 4'b0;  cout <= 1'b0;  end else begin  f <= f\_tmp;  cout <= cout\_tmp1;  end  end  endmodule |

1. **Code testbench**

|  |
| --- |
| `include "ALU4bit.v";  module testbench;  //input  reg [3:0] A;  reg [3:0] B;  reg Cin, S1, S0, M;  reg t\_clk, t\_rst\_n;  //output  wire [3:0] F;  wire Cout;  //tao xung clk  always begin  t\_clk = 1'b0;  # 10;  t\_clk = 1'b1;  # 10;  end  //instance define  ALU4bit test1(.a(A), .b(B), .cin(Cin), .s1(S1), .s0(S0), .m(M), .cout(Cout), .f(F), .clk(t\_clk), .rst\_n(t\_rst\_n));  //display the output  initial begin  $monitor("time=%d, clk=%b, rst\_n=%b, a=%b, b=%b, cin=%b, s1=%b, s0=%b, m=%b, cout=%b, f=%b \n",  $time, t\_clk, t\_rst\_n, A, B, Cin, S1, S0, M, Cout, F);  end  //declare testcase  initial begin  #1 t\_rst\_n = 0;  #10 t\_rst\_n = 1;  //case 1  #10 A = 4'b1111; B = 4'b1010; Cin = 0; S1 = 0; S0 = 0; M = 0;  //case 2  #10 A = 4'b1011; B = 4'b1010; Cin = 1; S1 = 0; S0 = 1; M = 0;  //case 3  #1 A = 4'b1111; B = 4'b1010; Cin = 0; S1 = 1; S0 = 0; M = 0;  //case 4  #10 A = 4'b1011; B = 4'b1010; Cin = 1; S1 = 1; S0 = 1; M = 0;  //case 5  #10 A = 4'b1011; B = 4'b1010; Cin = 0; S1 = 0; S0 = 0; M = 1;  //case 6  #10 A = 4'b1011; B = 4'b1010; Cin = 1; S1 = 0; S0 = 1; M = 1;  //case 7  #10 A = 4'b1011; B = 4'b1010; Cin = 0; S1 = 1; S0 = 0; M = 1;  //case 8  #10 A = 4'b1011; B = 4'b1010; Cin = 1; S1 = 1; S0 = 1; M = 1;  #100 $finish;  end  initial begin  $vcdplusfile ("testALU.vpd");  $vcdpluson();  end  endmodule |

1. **Hình ảnh:**

