**Thực hành Thiết kế hệ thống số với HDL**

**CE213.L21.MTCL**

**Họ và tên sinh viên - Mã số sinh viên:**

**Phạm Quang Hải - 18520703**

**LAB01: THIẾT KẾ MẠCH TỔ HỢP**

Mô tả các mạch tổ hợp: Full Adder, Bộ cộng 16 bit, Mux, Mux 16 bit, Bộ giải mã 1:2, Bộ giải mã 2:4, Bộ giải mã 3:8, Bộ mở rộng dấu (5 bit -> 16 bit), ALU 16 bit (các phép toán ADD, ADDI, SUB, AND, OR, XOR)

**Full Adder**

Module full\_adder(out,in1,in2,c\_in,c\_out);

output wire out,c\_out;

input wire in1,in2,c\_in;

wire outxor1,outand1,outand2;

xor xor1(outxor1,in1,in2);

xor xor2 (out,c\_in,outxor1);

and and1(outand1,in1,in2);

and and2(outand2,outxor1,c\_in);

or or2(c\_out,outand1,outand2);

endmodule

**Bộ cộng 16 bit**

module full\_adder\_subtractor\_16bit(out,overflow,in1,in2,c\_in);

output wire [15:0] out;

output wire overflow;

input wire [15:0] in1,in2;

input wire c\_in;

wire [14:0] CO;

wire c\_last;

wire [15:0] outxor;

xor xor1(overflow,c\_last,CO[14]);

xor xor\_15[15:0](outxor[15:0],c\_in,in2[15:0]);

full\_adder full\_adder\_inst1(

.out(out[15]),

.c\_out(c\_last),

.in1(in1[15]),

.in2(outxor[15]),

.c\_in(CO[14])

);

full\_adder full\_adder\_inst2[14:1](

.out(out[14:1]),

.c\_out(CO[14:1]),

.in1(in1[14:1]),

.in2(outxor[14:1]),

.c\_in(CO[13:0])

);

full\_adder full\_adder\_inst3(

.out(out[0]),

.c\_out(CO[0]),

.in1(in1[0]),

.in2(outxor[0]),

.c\_in(c\_in)

);

Endmodule

**2-to-1 Mux**

module mux21(out,in1,in2,sel);

output wire out;

input wire in1,in2,sel;

wire outand1,outand2;

wire sel\_bar;

not notsel(sel\_bar,sel);

and and1(outand1,in1,sel);

and and2(outand2,in2,sel\_bar);

or or1(out,outand1,outand2);

endmodule

**2-to-1 Mux 16 bit**

module mux21\_16bit(out,in1,in2,sel);

parameter n = 16;

output wire [n-1:0] out;

input wire [n-1:0] in1,in2;

input wire sel;

mux21 mux21\_inst[n-1:0](

.out(out),

.in1(in1),

.in2(in2),

.sel(sel)

);

Endmodule

**4-to-1 Mux**

module mux41\_1bit(out,in1,in2,in3,in4,sel);

output wire out;

input wire [1:0] sel;

input wire in1,in2,in3,in4;

wire outinst1,outinst2;

mux21 mux21\_inst1(

.out(outinst1),

.in1(in4),

.in2(in3),

.sel(sel[0])

);

mux21 mux21\_inst2(

.out(outinst2),

.in1(in2),

.in2(in1),

.sel(sel[0])

);

mux21 mux21\_inst3(

.out(out),

.in1(outinst1),

.in2(outinst2),

.sel(sel[1])

);

Endmodule

**4-to-1 Mux 16 bit**

module mux41\_16bit(out,in1,in2,in3,in4,sel);

parameter n=16;

output wire [n-1:0] out;

input wire [n-1:0] in1,in2,in3,in4;

input wire [1:0] sel;

mux41\_1bit mux41\_1bit\_inst[n-1:0](

.out(out),

.in1(in1),

.in2(in2),

.in3(in3),

.in4(in4),

.sel(sel)

);

Endmodule

**8-to-1 Mux**

module mux81\_1bit(out,in1,in2,in3,in4,in5,in6,in7,in8,sel);

output wire out;

input wire [2:0] sel;

input wire in1,in2,in3,in4,in5,in6,in7,in8;

wire outinst1,outinst2;

mux41\_1bit mux41\_1bit\_inst1(

.out(outinst1),

.in1(in8),

.in2(in7),

.in3(in6),

.in4(in5),

.sel(sel[1:0])

);

mux41\_1bit mux41\_1bit\_inst2(

.out(outinst2),

.in1(in4),

.in2(in3),

.in3(in2),

.in4(in1),

.sel(sel[1:0])

);

mux21 mux21\_inst(

.out(out),

.in1(outinst2),

.in2(outinst1),

.sel(sel[2])

);

Endmodule

**8-to-1 Mux 16 bit**

module mux81\_16bit(out,in1,in2,in3,in4,in5,in6,in7,in8,sel);

parameter n=16;

output wire [n-1:0] out;

input wire [2:0] sel;

input wire [n-1:0] in1,in2,in3,in4,in5,in6,in7,in8;

mux81\_1bit mux81\_1bit\_inst[n-1:0](

.out(out),

.in1(in8),

.in2(in7),

.in3(in6),

.in4(in5),

.in5(in4),

.in6(in3),

.in7(in2),

.in8(in1),

.sel(sel)

);

Endmodule

**Bộ giải mã 1:2**

module decoder1to2(out,in);

output wire [1:0] out;

input wire in;

wire in\_bar;

not notin(in\_bar,in);

assign out[1] = {in};

assign out[0] = {in\_bar};

endmodule

**Bộ giải mã 2:4**

module decoder2to4 (out,in);

output wire [3:0] out;

input wire [1:0] in;

wire in\_bar[1:0];

not notin1 (in\_bar[1],in[1]);

not notin0 (in\_bar[0],in[0]);

and and1(out[3],in[1],in[0]);

and and2(out[2],in[1],in\_bar[0]);

and and3(out[1],in\_bar[1],in[0]);

and and4(out[0],in\_bar[1],in\_bar[0]);

endmodule

**Bộ giải mã 3:8**

module decoder3to8 (out,in);

output wire [7:0] out;

input wire [2:0] in;

wire in\_bar[2:0];

not notin2(in\_bar[2],in[2]);

not notin1(in\_bar[1],in[1]);

not notin0(in\_bar[0],in[0]);

and and1(out[7],in[2],in[1],in[0]);

and and2(out[6],in[2],in[1],in\_bar[0]);

and and3(out[5],in[2],in\_bar[1],in[0]);

and and4(out[4],in[2],in\_bar[1],in\_bar[0]);

and and5(out[3],in\_bar[2],in[1],in[0]);

and and6(out[2],in\_bar[2],in[1],in\_bar[0]);

and and7(out[1],in\_bar[2],in\_bar[1],in[0]);

and and8(out[0],in\_bar[2],in\_bar[1],in\_bar[0]);

endmodule

**Sign extend 5 to 16 bit**

module signextend(out,in);

output wire [15:0] out;

input wire [4:0] in;

assign out = {10'd0,in[4:0]};

endmodule

**ALU 16 bit (các phép toán ADD, ADDI, SUB, AND, OR, XOR)**

**Module AND**

module mdl\_and(out,in1,in2);

output wire [15:0] out;

input wire [15:0] in1,in2;

and and\_inst[15:0](out[15:0],in1[15:0],in2[15:0]);

endmodule

**module OR**

module mdl\_or(out,in1,in2);

output wire [15:0] out;

input wire [15:0] in1,in2;

or or\_inst[15:0](out[15:0],in1[15:0],in2[15:0]);

endmodule

**module XOR**

module mdl\_xor(out,in1,in2);

output wire [15:0] out;

input wire [15:0] in1,in2;

xor xor\_inst[15:0](out[15:0],in1[15:0],in2[15:0]);

endmodule

**Module Overflow flag**

module overflow (out,in1,in2,in3,sel);

output tri out;

input wire in1,in2,in3;

input wire [2:0] sel;

wire [2:0] sel\_bar;

not nots2 (sel\_bar[2],sel[2]);

not nots1 (sel\_bar[1],sel[1]);

not nots0 (sel\_bar[0],sel[0]);

wire outand1,outand2,outand3;

and and1(outand1,sel\_bar[2],sel\_bar[1],sel\_bar[0]);

and and2(outand2,sel\_bar[2],sel\_bar[1],sel[0]);

and and3(outand3,sel\_bar[2],sel[1],sel\_bar[0]);

bufif1 entri1 (out,in1,outand1);

bufif1 entri2 (out,in2,outand2);

bufif1 entri3 (out,in3,outand3);

endmodule

**module ALU**

module ALU\_16bit(out,overflow,in1,in2,sel);

output wire [15:0] out;

output wire overflow;

input wire [15:0] in1,in2;

input wire [2:0] sel;

wire [15:0] outFAS1;

wire [15:0] outFAS2;

wire [15:0] outFAS3;

wire [15:0] outand;

wire [15:0] outor;

wire [15:0] outxor;

wire overflow1,overflow2,overflow3;

overflow overflow\_inst(

.out(overflow),

.in1(overflow1),

.in2(overflow2),

.in3(overflow3),

.sel(sel)

);

full\_adder\_subtractor\_16bit full\_adder\_subtractor\_16bit\_add(

.out(outFAS1),

.overflow(overflow1),

.in1(in1),

.in2(in2),

.c\_in(1'b0)

);

full\_adder\_subtractor\_16bit full\_adder\_subtractor\_16bit\_addi(

.out(outFAS2),

.overflow(overflow2),

.in1(in1),

.in2(in2),

.c\_in(1'b0)

);

full\_adder\_subtractor\_16bit full\_adder\_subtractor\_16bit\_sub(

.out(outFAS3),

.overflow(overflow3),

.in1(in1),

.in2(in2),

.c\_in(1'b1)

);

mdl\_and mdl\_and\_inst(

.out(outand),

.in1(in1),

.in2(in2)

);

mdl\_or mdl\_or\_inst(

.out(outor),

.in1(in1),

.in2(in2)

);

mdl\_xor mdl\_xor\_inst(

.out(outxor),

.in1(in1),

.in2(in2)

);

mux81\_16bit mux81\_16bit\_inst(

.out(out),

.in1(outFAS1),

.in2(outFAS2),

.in3(outFAS3),

.in4(outand),

.in5(outor),

.in6(outxor),

.in7(16'd0),

.in8(16'd0),

.sel(sel)

);

endmodule