**Logo

Description automatically generated San Francisco Bay University**

**EE488 - Computer Architecture**

**Homework Assignment #6**

**Due day: 4/10/2023**

**Name: Simon Gezae ID: 19830**

**Instruction:**

1. **Push the answer sheet to GitHub in word file**
2. **Overdue homework submission could not be accepted.**
3. **Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)**
4. Write the design module in RTL level to implement"Carry Lookahead" addition for 8-bits adder including carry-in bit, and then verify it by the testbench.

Design:

module cla8(input [7:0] a, b,

input cin,

output [7:0] sum,

output cout);

wire carry\_mid;

cla4 u1 (a[3:0],b[3:0],cin,sum[3:0],carry\_mid); //cout of the first half

cla4 u2 (a[7:4],b[7:4],carry\_mid,sum[7:4],cout);//is used as cin in the next

endmodule

module cla4(input [3:0] a, b,

input cin,

output [3:0] sum,

output cout);

wire [3:0] c, p, g;

assign p = a^b;

assign g = a&b;

assign c[0] = cin;

assign c[1]= g[0] | (p[0]&c[0]); // g[0] = a[0]&b[0] p[0] = a[0]&b[0]

assign c[2]= g[1] | (p[1]&g[0]) | p[1]&p[0]&c[0];

assign c[3]= g[2] | (p[2]&g[1]) | p[2]&p[1]&g[0] | p[2]&p[1]&p[0]&c[0];

assign cout= g[3] | (p[3]&g[2]) | p[3]&p[2]&g[1] | p[3]&p[2]&p[1]&g[0] | p[3]&p[2]&p[1]&p[0]&c[0];

assign sum=p^c; //p=(a^b) sum=a^b^c

endmodule

Testbench:

module tb;

reg [7:0] a, b;

reg cin;

wire [7:0] sum;

wire cout;

integer i;

cla8 u3 (a,b,cin,sum,cout);

initial begin

$display("A\t\tB\t\tCin\tSum\t\tCout");

$monitor("%b\t%b\t%b\t%b\t%b", a,b,cin,sum,cout);

for (i=508; i<512; i=i+1)begin

{a,b,cin} = i;

#2;

end

#2 a = 8'b0; b = 8'b0; cin = 8'b0;

#2 a = 8'b1111\_1111; b = 8'b1111\_1111; cin = 8'b0;

#5 $finish;

end

endmodule

Edaplayground: <https://edaplayground.com/x/PHYp>

Question 1 method 2

Design:

module CLAhead8(input [7:0] a,

input [7:0] b,

input cin,

output [7:0] sum,

output cout);

wire [8:0] p, g;

wire [7:0] c;

assign p = a^b;

assign g = a&b;

assign c[0] = cin;

genvar gv;

generate

for (gv = 1; gv < 8; gv = gv + 1) begin : lag

assign c[gv] = g[gv] | p[gv] & c[gv - 1];

end

endgenerate

assign sum = a + b + cin;

assign cout = g[7] | p[7] & c[7]; // cout == ci+1 = gi+pi&ci

endmodule

Testbench:

module TB();

reg [7:0] a, b;

reg cin;

wire [7:0] sum;

wire cout;

CLAhead8 U1 (.a(a), .b(b), .cin(cin), .sum(sum), .cout(cout));

initial begin

$display("A\t\tB\t\tCin\tSum\t\tCout");

$monitor("%b\t%b\t%b\t%b\t%b", a,b,cin,sum,cout);

a = 8'b00000000; b = 8'b00000000; cin = 0;

#1 a = 8'b00000001; b = 8'b00000001; cin = 0;

#1 a = 8'b00000011; b = 8'b00000011; cin = 0;

#1 a = 8'b00111111; b = 8'b00111111; cin = 0;

#1 a = 8'b01111111; b = 8'b01111111; cin = 1;

#1 a = 8'b11111111; b = 8'b11111111; cin = 0;

#1 a = 8'b11111111; b = 8'b11111111; cin = 1;

#1 $finish();

end

endmodule

Edaplayground: <https://www.edaplayground.com/x/vhM8>

1. Write a Verilog module to design 8-bits ALU based on the following opcodes. In the summation operation, the submodule instantiated by "generate" block in the top module should implement "Carry Lookahead". And then verify your design by the testbench.

|  |  |
| --- | --- |
| Opcode | Operations |
| 0000 | Out = A + B |
| 0001 | Out = A - B |
| 0010 | Out = A \* B |
| 0011 | Out = A / B |
| 0100 | Out = A << 1 |
| 0101 | Out = A >> 1 |
| 0110 | Out = A rotated left by 1 |
| 0111 | Out = A rotated right by 1 |
| 1000 | Out = A and B |
| 1001 | Out = A or B |
| 1010 | Out = A xor B |
| 1011 | Out = A nor B |
| 1100 | Out = A nand B |
| 1101 | Out = A xnor B |
| 1110 | Out = 1 if A>B else 0 |
| 1111 | Out = 1 if A=B else 0 |

Design:

module ALU(

input [7:0] a,

input [7:0] b,

input [3:0] opcode,

output reg [15:0] out);

//using generate

wire [7:0] pc;

genvar i;

generate

wire [7:0] gc;

assign gc = 1'b0;

for (i = 0; i < 8; i = i + 1) begin : cla

assign gc[i] = a[i] & b[i];

end

for (i = 1; i < 8; i = i + 1) begin : p

assign pc[i] = a[i-1] | b[i-1];

end

assign pc[0] = 1'b0;

endgenerate

always @(\*) begin

case(opcode)

4'b0000: out = a + b;

4'b0001: out = a - b;

4'b0010: out = a \* b;

4'b0011: out = a / b;

4'b0100: out = a << 1;

4'b0101: out = a >> 1;

4'b0110: out = {a[6:0], a[7]};

4'b0111: out = {a[0], a[7:1]};

4'b1000: out = a & b;

4'b1001: out = a | b;

4'b1010: out = a ^ b;

4'b1011: out = ~(a | b);

4'b1100: out = ~(a & b);

4'b1101: out = ~(a ^ b);

4'b1110: out = (a > b) ? 1'b1 : 1'b0;

4'b1111: out = (a == b) ? 1'b1 : 1'b0;

default: out = 8'bx;

endcase

end

endmodule

Testbench:

module TB;

reg [7:0] A;

reg [7:0] B;

reg [3:0] opcode;

wire [15:0] out;

ALU U1 (.a(A),

.b(B),

.opcode(opcode),

.out(out));

initial begin

A = 8'b11010101; B = 8'b00101010; opcode = 4'b0000;

#2 $display("A = %b, B=%b, out", A, B);

#2 $display("A + B = %b", out[7:0]);

opcode = 4'b0001;

#2 $display("A - B = %b", out[7:0]);

opcode = 4'b0010;

#2 $display("A \* B = %b", out);

opcode = 4'b0011;

#2 $display("A / B = %b", out);

opcode = 4'b0100;

#2 $display("A << 1 = %b", out[7:0]);

opcode = 4'b0101;

#2 $display("A >> 1 = %b", out[7:0]);

opcode = 4'b0110;

#2 $display("ROL = %b", out[7:0]);

opcode = 4'b0111;

#2 $display("ROR = %b", out[7:0]);

opcode = 4'b1000;

#2 $display("A & B = %b", out[7:0]);

opcode = 4'b1001;

#2 $display("A | B = %b", out[7:0]);

opcode = 4'b1010;

#2 $display("A ^ B = %b", out[7:0]);

opcode = 4'b1011;

#2 $display("~(A | B) = %b", out[7:0]);

opcode = 4'b1100;

#2 $display("~(A & B) = %b", out[7:0]);

opcode = 4'b1101;

#2 $display("~(A ^ B) = %b", out[7:0]);

opcode = 4'b1110;

#2 $display("A > B = %b",out[7:0]);

opcode = 4'b1111;

#2 $display("A == B = %b", out[7:0]);

end

endmodule

Edaplayground: <https://www.edaplayground.com/x/KR7r>