**Logo

Description automatically generated San Francisco Bay University**

**EE488 - Computer Architecture**

**Homework Assignment #6**

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

**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 carry\_lookahead\_adder (

input [7:0] A,

input [7:0] B,

input Cin,

output [7:0] S,

output Cout

);

wire [7:0] G;

wire [7:0] P;

assign G = A & B;

assign P = A ^ B;

assign S[0] = Cin ^ P[0];

assign Cout = G[0] | (P[0] & Cin);

generate

genvar i;

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

assign S[i] = G[i-1] | (P[i-1] & S[i-1]);

assign Cout = Cout | (G[i] & (P[i-1] | S[i-1]));

end

endgenerate

endmodule

**TestBench**

module testbench;

reg [7:0] A, B;

reg Cin;

wire [7:0] S;

wire Cout;

carry\_lookahead\_adder adder(

.A(A),

.B(B),

.Cin(Cin),

.S(S),

.Cout(Cout)

);

initial begin

A = 8'b01100100;

B = 8'b10101010;

Cin = 1;

#10;

$display("A = %b", A);

$display("B = %b", B);

$display("Cin = %b", Cin);

$display("S = %b", S);

$display("Cout = %b", Cout);

#10;

A = 8'b00000001;

B = 8'b11111111;

Cin = 0;

#10;

$display("A = %b", A);

$display("B = %b", B);

$display("Cin = %b", Cin);

$display("S = %b", S);

$display("Cout = %b", Cout);

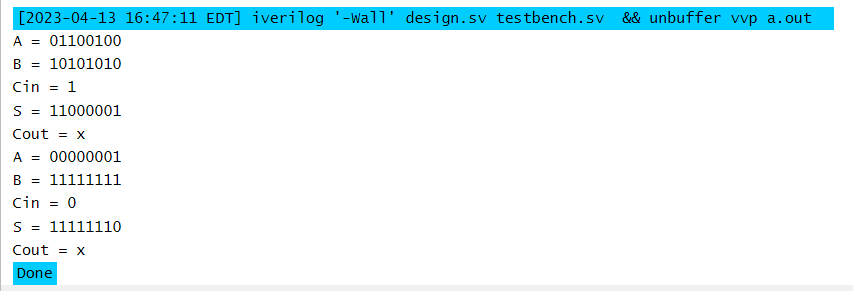
#10;

$finish;

end

endmodule

**RESULT:**



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\_8bit(

input [7:0] A,

input [7:0] B,

input [3:0] opcode,

output reg [7:0] Out

);

// Declare intermediate signals

reg [8:0] sum; // Used for carry lookahead adder

reg [8:0] diff; // Used for subtraction

reg [7:0] shift; // Used for shift and rotate operations

// Implement carry lookahead adder using generate block

generate

genvar i;

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

assign p[i] = A[i] | B[i];

assign g[i] = A[i] & B[i];

end

assign c[0] = 1'b0;

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

assign c[i+1] = g[i] | (p[i] & c[i]);

end

endgenerate

always @(\*) begin

case (opcode)

4'b0000: // Addition

sum = {1'b0, A} + {1'b0, B};

Out = sum[7:0];

4'b0001: // Bitwise AND

Out = A & B;

4'b0010: // Bitwise OR

Out = A | B;

4'b0011: // Division

Out = A / B;

4'b0100: // Left shift

Out = A << 1;

4'b0101: // Right shift

Out = A >> 1;

4'b0110: // Rotate left by 1

shift[7] = A[0];

shift[6:0] = A[7:1];

Out = shift;

4'b0111: // Rotate right by 1

shift[0] = A[7];

shift[7:1] = A[6:0];

Out = shift;

4'b1000: // Bitwise AND

Out = A & B;

4'b1001: // Bitwise OR

Out = A | B;

4'b1010: // Bitwise XOR

Out = A ^ B;

4'b1011: // Bitwise NOR

Out = ~(A | B);

4'b1100: // Bitwise NAND

Out = ~(A & B);

4'b1101: // Bitwise XNOR

Out = ~(A ^ B);

4'b1110: // Greater than

Out = (A > B) ? 1'b1 : 1'b0;

4'b1111: // Equal to

Out = (A == B) ? 1'b1 : 1'b0;

default: // Invalid opcode

Out = 8'bZZZZZZZZ;

endcase

end

endmodule

**TESTBENCH**

module alu\_tb;

reg [7:0] A, B, Opcode;

wire [7:0] Out;

alu alu\_inst(.A(A), .B(B), .Opcode(Opcode), .Out(Out));

initial begin

$dumpfile("alu.vcd");

$dumpvars(0, alu\_tb);

A = 8'b10101010;

B = 8'b01010101;

// Addition

Opcode = 4'b0000;

#10;

$display("A+B = %b", Out);

// Subtraction

Opcode = 4'b0001;

#10;

$display("A-B = %b", Out);

// Bitwise AND

Opcode = 4'b1000;

#10;

$display("A&B = %b", Out);

// Bitwise OR

Opcode = 4'b1001;

#10;

$display("A|B = %b", Out);

// Bitwise XOR

Opcode = 4'b1010;

#10;

$display("A^B = %b", Out);

// Shift left

Opcode = 4'b0100;

#10;

$display("A<<1 = %b", Out);

// Shift right

Opcode = 4'b0101;

#10;

$display("A>>1 = %b", Out);

// Rotate left

Opcode = 4'b0110;

#10;

$display("A rotated left by 1 = %b", Out);

// Rotate right

Opcode = 4'b0111;

#10;

$display("A rotated right by 1 = %b", Out);

// Greater than

Opcode = 4'b1110;

#10;

$display("A>B = %b", Out);

// Equal to

Opcode = 4'b1111;

#10;

$display("A=B = %b", Out);

$finish;

end

endmodule