Verilog project3　　b10601002 廖品捷

Github: https://github.com/JamesLiao714/DLD-verilog

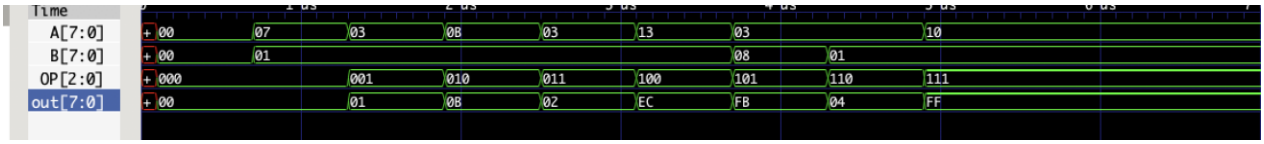
Test bench:

```
`timescale 1 ns/100ps
//-TestBench-
module alu_test();
  reg [7:0]A;
  reg [7:0]B;
  wire [7:0]out;
  reg [2:0]OP;


  alu test(A,B,OP,out);


  initial
    begin
      #100
      A = 8'b0; B = 8'b0; OP = 3'b000;
      #600 A = 8'b00000111; B = 8'b00000001; OP = 3'b000;
      #600 A = 8'b00000011; B = 8'b00000001; OP = 3'b001;
      #600 A = 8'b00001011; B = 8'b00000001; OP = 3'b010;
      #600 A = 8'b00000011; B = 8'b00000001; OP = 3'b011;
      #600 A = 8'b00010011; B = 8'b00000001; OP = 3'b100;
      #600 A = 8'b00000011; B = 8'b00001000; OP = 3'b101;
      #600 A = 8'b00000011; B = 8'b00000001; OP = 3'b110;
      #600 A = 8'b00010000; B = 8'b00000001; OP = 3'b111;
    end
initial #9000 $finish;
initial $dumpvars;
initial begin
     $display("====== show simulation ==========");
     $monitor("time: %4d %b(A) %b %b(B) out = %b \n",$time, A, OP, B, out);
     end
endmodule
```

ALU Waveform:



Sel	Operation	Description
000	y = 8'b0	Minimum value
001	y = A & B	Bitwise AND
010	y = A | B	Bitwise OR
011	y = A ^ B	Bitwise exclusive OR
100	y = ~A	Bitwise complement
101	y = A - B	Subtract
110	y = A + B	Add (Assume A and B are unsigned)
111	y = 8'hFF	Maximum value

implementation of $monitor, $display and $time:

```
VCD info: dumpfile dump.vcd opened for output.
======= show simulation ==========
time:    0 xxxxxxxx(A) xxx xxxxxxxx(B) out = xxxxxxxx

time:  100 00000000(A) 000 00000000(B) out = 00000000

time:  700 00000111(A) 000 00000001(B) out = 00000000

time: 1300 00000011(A) 001 00000001(B) out = 00000001

time: 1900 00001011(A) 010 00000001(B) out = 00001011

time: 2500 00000011(A) 011 00000001(B) out = 00000010

time: 3100 00010011(A) 100 00000001(B) out = 11101100

time: 3700 00000011(A) 101 00001000(B) out = 11111011

time: 4300 00000011(A) 110 00000001(B) out = 00000100

time: 4900 00010000(A) 111 00000001(B) out = 11111111
```

ALU Model source code:

```
//ALU model
module alu (
Data_A, // I 8-bit : first input signal
Data_B, // I 8-bit : second input signal
OP_Code, // I 3-bit : operation
Data_Out // I 16-bit : result
);
input [7:0] Data_A, Data_B;
input [2:0] OP_Code;
output [7:0] Data_Out;
reg [7:0] Data_Out;
always@(Data_A or Data_B or OP_Code) begin
case (OP_Code)
  3'b000: Data_Out = 8'b0;
  3'b001: Data_Out = Data_A & Data_B;
  3'b010: Data_Out = Data_A | Data_B;
  3'b011: Data_Out = Data_A ^ Data_B;
  3'b100: Data_Out = ~Data_A;
  3'b101: Data_Out = Data_A - Data_B;
  3'b110: Data_Out = Data_A + Data_B;
  3'b111: Data_Out = 8'hFF;
default : Data_Out = 16'b0;
endcase
end
endmodule
```