PHYS 234 Lab 7: Microprocessor Components: Sign Extension, ALU

Ethan Yang

Sign extension is the process of increasing the number of bits in a signed binary number while preserving its original sign and value by copying the most significant bits to the positions on the left. On the other hand, ALU, or an arithmetic logic unit, is a digital circuit that performs arithmetic and bitwise operations on integer binary numbers. This lab will be implementing both the sign extension and ALU circuit, and displaying the results on the 7-segment display on the FPGA board.

The Verilog code below shows a basic ALU circuit combined with a 4 to 8 bit sign extension. A case statement was used to define which arithmetic operation to perform according to the f value. The table below shows each f's corresponding operation. SLT stands for "Set if Less Than", and returns 1 if A > B, and 0 otherwise.

f	**Operation**
000	A & B
001	A | B
010	A + B
011	X
100	A & ~B
101	A | ~B
110	A - B
111	SLT

The "dout" assignment line implements the sign extension function, where the most significant digit was copied 4 times.

```
`timescale 1ns / 1ps

module eelab7(
    input [3:0] dinA,
    input [3:0] dinB,
    input [2:0] f,
    output [7:0] dout
    );
```

```
reg [3:0] result;

always @ (*) begin
    case (f)
        0: result = dinA & dinB;
        1: result = dinA | dinB;
        2: result = dinA + dinB;
        4: result = dinA & ~dinB;
        5: result = dinA | ~dinB;
        6: result = dinA - dinB;
        7: begin
            if (dinA < dinB) result = 1;
            else result = 0;
        end
    endcase
end

assign dout = {{4{result[3]}}, result[3:0]};
endmodule
```

As a result, all 8 LEDs would be used to display the resulting number of the ALU operation in 8 bits. For example, if we input A = 1010 and B = 0101 and f = 010, the circuit would output 1111 1111 because the sum 1111 was sign extended from 4 bit to 8 bit.

Next, a 8 to 16 bit sign extension circuit that displays the result on the 7-segment display was designed. The x7segb module takes a 16 bit input and displays it across 4 7-segment displays by rapidly cycling through them using a divided clock. The divided clock selects one 4 bit portion of the input at a time and decodes that portion into the correct segment pattern for that digit. Only one digit is enabled at a time, but the switching happens so quickly that all digits appear simultaneously.

Our eelab7 module simply passes all the parameter values to the x7segb module and performs the sign extension function by copying the most significant bit 8 times.

```
`timescale 1ns / 1ps

module x7segb(
    input wire [15:0] x ,
    input wire mclk , //This is the 50 MHz clock
    output reg [6:0] a_to_g ,
    output reg [3:0] an ,
    output wire dp
);

wire [1:0] s;
reg [3:0] digit ;
wire [3:0] aen ;
reg [19:0] clkdiv;

assign dp = 1 ; //Turn off the decimal point
assign s = clkdiv[19:18] ; //Count every 5.2 ms = 190.7 Hz = 50,000,000 Hz
/ 218 = 50,000,000 Hz / 262,144
assign aen[3] = | x[15:12] ; //Does the first digit have a value? (If not,
let's turn it off, called "Blanking".)
assign aen[2] = | x[15:8] ; //Do the first two digits have a value?
assign aen[1] = | x[15:4] ; //Do the first three digits have a value?
assign aen[0] = 1 ; //We will not turn off the 4th digit even if it's
zero. So we'll always leave it on.
// Quad 4-to-1 MUX: mux44
always @(*)
    case(s) // "s" is a clock with two bits and frequency 190.7 Hz
        0: digit = x[3:0]; // there are four possibilities for "s" 00,
                              01, 10, y 11
        1: digit = x[7:4]; // "x" is a 16 bit number.
        2: digit = x[11:8]; // when "s" has a value of 10, we'll copy
                               bits [11:8] to "digit".
        3: digit = x[15:12];
        default: digit = x[3:0];
    endcase
// 7-segment decoder: hex7seg
always @ (*)
    case(digit)
        0: a_to_g = 7'b0000001;
        1: a_to_g = 7'b1001111;
        2: a_to_g = 7'b0010010;
        3: a_to_g = 7'b0000110;
        4: a_to_g = 7'b1001100;
        5: a_to_g = 7'b0100100;
        6: a_to_g = 7'b0100000;
        7: a_to_g = 7'b0001111;
        8: a_to_g = 7'b0000000;
        9: a_to_g = 7'b0000100;
        'hA: a_to_g = 7'b0001000;
        'hB: a_to_g = 7'b1100000;
        'hC: a_to_g = 7'b0110001;
        'hD: a_to_g = 7'b1000010;
        'hE: a_to_g = 7'b0110000;
        'hF: a_to_g = 7'b0111000;
        default: a_to_g = 7'b0000001;
    endcase

always @(*)
    begin
    an = 4'b1111; // We'll start by turning all of the display's off. .
    if(aen[s] == 1) // If the value of the "s" bit isn't blanked
        an[s] = 0; // then we'll turn it on
    end

//Clock divider
always @(posedge mclk)
    clkdiv <= clkdiv + 1;
endmodule

module eelab7(
        input [7:0] din,
        input mclk , //This is the 50 MHz clock
        output wire [6:0] a_to_g ,
        output wire [3:0] an ,
        output wire dp,
        output [15:0] x
    );

assign x = {{8{din[7]}}, din[7:0]};
x7segb u(x, mclk, a_to_g, an, dp);
endmodule
```

The 7-segment display displays the results in hexadecimal, and all 8 bit inputs are sign extended to 16 bits. For example, if “din” is 0000 0001, the 7-segment display shows 0001. For another instance, if “din” is 1111 1111, the 7-segment display would show “FFFF”. The circuit treats 1111 1111 as a negative number in two’s complement, and also outputs the hexadecimal representation of -1, which is “FFFF”.