EE2003: Computer Organisation

Indian Institute of Technology, Madras (Jul - Nov '23)

Assignment 2: Arithmetic using Verilog

Arnav Mahajan [EP21B004]

(P43) Create a Verilog module that rounds a 32-bit binary-coded decimal (BCD) number to a specified number of decimal places. The module should support rounding up or down based on the fractional part of the BCD representation.

Logic Used

The input BCD number (`[31:0] in`) is passed through the multiplexer to get the digit up to which truncation is performed. `[2:0] sel` selects the digit (indexed from 0). It is passed through compare with 5 logic and the output is shifted left `4*sel + 4` times so that it can be added to get the truncated number. An array of ones is also shifted left `4*sel + 4` times and ANDed with the input to get the lower bound. This and the shifted compare output are both passed through the BCD adder for every digit to obtain the final number rounded up or down as decided by the `sel` digit greater or less than 5.

Program

Verilog and Testbench Files:
`https://drive.google.com/drive/folders/1Vykgrn_Uw-595uzjRzEzvh9cVFVUjxA2?usp=sharing`

Main Module

```
‘timescale 1ns / 1ps

module bcd_truncation(
    input [31:0] in,
    input [2:0] sel,
    wire [3:0] selected,
    wire [31:0] compare_out,
    wire [31:0] compare_out_shifted,
    wire [31:0] extract_digits,
    wire [31:0] extract_digits_shifted,
    wire [31:0] extracted_digits,
    wire [7:0] carry,
    output [31:0] out,
    output carry_out);

    assign selected = in[4*sel + 3 -: 4]; // select the digit to be truncated

    assign compare_out[31:1] = 31’b0; // 31-bit array of zeros. last bit reserved
        for comparison out

    assign compare_out[0] = selected[3] | selected[0] & selected[2] |
```

```
        selected[1] & selected[2]; // check if selected digit >= 5

    assign compare_out_shifted = compare_out << 4*sel + 4; // shift to get the
        compared output on new LSB

    assign extract_digits = 32'hffffffff; // array of ones
    assign extract_digits_shifted = extract_digits << 4*sel + 4;

    assign extracted_digits = in & extract_digits_shifted;

    bcd_adder dig_0 (.a(extracted_digits[3:0]), .b(compare_out_shifted[3:0]),
        .carry_in(1'b0), .carry(carry[0]), .sum(out[3:0]));
    bcd_adder dig_1 (.a(extracted_digits[7:4]), .b(compare_out_shifted[7:4]),
        .carry_in(carry[0]), .carry(carry[1]), .sum(out[7:4]));
    bcd_adder dig_2 (.a(extracted_digits[11:8]), .b(compare_out_shifted[11:8]),
        .carry_in(carry[1]), .carry(carry[2]), .sum(out[11:8]));
    bcd_adder dig_3 (.a(extracted_digits[15:12]), .b(compare_out_shifted[15:12]),
        .carry_in(carry[2]), .carry(carry[3]), .sum(out[15:12]));
    bcd_adder dig_4 (.a(extracted_digits[19:16]), .b(compare_out_shifted[19:16]),
        .carry_in(carry[3]), .carry(carry[4]), .sum(out[19:16]));
    bcd_adder dig_5 (.a(extracted_digits[23:20]), .b(compare_out_shifted[23:20]),
        .carry_in(carry[4]), .carry(carry[5]), .sum(out[23:20]));
    bcd_adder dig_6 (.a(extracted_digits[27:24]), .b(compare_out_shifted[27:24]),
        .carry_in(carry[5]), .carry(carry[6]), .sum(out[27:24]));
    bcd_adder dig_7 (.a(extracted_digits[31:28]), .b(compare_out_shifted[31:28]),
        .carry_in(carry[6]), .carry(carry[7]), .sum(out[31:28]));

    assign carry_out = carry[7];

endmodule
```

BCD Adder Module

```
`timescale 1ns / 1ps

module bcd_adder(a, b, carry_in, sum, carry);
    input [3:0] a, b;
    input carry_in;
    output [3:0] sum;
    output carry;

    reg [4:0] sum_temp;
    reg [3:0] sum;
    reg carry;

    always @(a, b, carry_in)
    begin
        sum_temp = a + b + carry_in;
        if(sum_temp > 9) begin
```

```
            sum_temp = sum_temp + 6;
            carry = 1;
            sum = sum_temp[3:0]; end
        else begin
        carry = 0;
        sum = sum_temp[3:0]; end
    end

endmodule
```

Testbench Module

```
‘timescale 1ns / 1ps

module bcd_truncation_tb();

    reg [31:0] t_in;
    reg [2:0] t_sel;
    wire [31:0] t_out;
    wire t_carry_out;

    bcd_truncation dut(.in(t_in), .sel(t_sel), .out(t_out),
        .carry_out(t_carry_out));

    initial begin

    t_in = 32'b11101100101010000110010000010000; t_sel = 3'b101;
        // t_in = 76543210 t_sel = 5
    #10
    t_in = 32'b10011001100110011001100110011001; t_sel = 3'b000;
        // t_in = 99999999 t_sel = 0
    #10
    t_in = 32'b00011001001110010101100101111001; t_sel = 3'b001;
        // t_in = 19395979 t_sel = 1
    #10
    t_in = 32'b10011001100110011001100110010101; t_sel = 3'b000;
        // t_in = 99999995 t_sel = 0
    #10
    t_in = 32'b10010001100110011001100110011001; t_sel = 3'b110;
        // t_in = 91999999 t_sel = 6
    #10

    $stop;

    end

endmodule
```

Output

Testbench Output