ROM Squarer and Square Rooter Circuits

Verilog Lab 5

Chase A. Lotito, *SIUC Undergraduate*

Section A: Squaring Circuit

The following Verilog code implements a read-only memory (ROM) block that stores data for the squares of numbers; specifically, the squares for unsigned numbers 0 to 15, and signed numbers -8 to +7.

```
/*
    Chase Lotito - SIUC
    ECE426 - Lab 5 - Design of Memory in Verilog
    Section A: ROM Squaring Circuit
        -> square unsigned 0-15
        -> square signed -8-7
*/

module rom_a (n, sign, square);

// I/O
input [3:0] n;              // signed number
input sign;                 // bit (1 = signed) (0 = unsigned)
output reg [7:0] square;    // answer = n**2

// procedural block to compute squares as inputs come in
always @ (n or sign)
begin
    if (sign == 1'b0)
    begin
        case (n)            // UNSIGNED
            0 :square <= 0;
            1 :square <= 1;
            2 :square <= 4;
            3 :square <= 9;
            4 :square <= 16;
            5 :square <= 25;
            6 :square <= 36;
            7 :square <= 49;
            8 :square <= 64;
            9 :square <= 81;
            10 :square <= 100;
            11 :square <= 121;
            12 :square <= 144;
            13 :square <= 169;
            14 :square <= 196;
            15 :square <= 225;
```

```
            default: square <= 1'bx; // invalid input
        endcase
    end
    else
    begin
        case(n)             // SIGNED
            0 :square <= 0;
            1 :square <= 1;
            2 :square <= 4;
            3 :square <= 9;
            4 :square <= 16;
            5 :square <= 25;
            6 :square <= 36;
            7 :square <= 49;
            8 :square <= 64; // -8 x -8
            9 :square <= 49;
            10 :square <= 36;
            11 :square <= 25;
            12 :square <= 16;
            13 :square <= 9;
            14 :square <= 4;
            15 :square <= 1;
            default: square <= 1'bx; // invalid input
        endcase
    end
end

endmodule
```

To simulate the functionality of this ROM block, I sent in binary values 0000 to 1111 twice, where one iteration has the sign-bit set to 0 (unsigned) and then the sign-bit set to 1 (signed).

```
/*
    chase lotito
    lab 5 section a
    rom squarer testbench
*/

`include "./rom_a.v"
`timescale 1us / 1us

module rom_a_tb();

// GTKWAVE (waveform simulator)
initial begin : GTKWAVE
    $dumpfile("rom_a.vcd");
    $dumpvars(0, rom_a_tb);
```

```
16 end
17
18 // I/O
19 reg [3:0] n;              // signed number
20 reg sign;                 // bit (1 = signed) (0 = unsigned)
21 wire [7:0] square;   // answer = n**2
22
23 // Initialize the Squarer ROM
24 rom_a U1 (
25     .n(n), .sign(sign), .square(square)
26 );
27
28 // Send 4-bit numbers 0000 -> 1111
29 initial begin : stimulus
30
31     // begin sim with unsigned numbers
32     sign = 0;
33
34     // n = 0000 -> 1111
35     for(integer i = 0; i < 16; i = i + 1)
36         #10 n = i;
37
38     #10 n = 0;       // reset n
39     sign = 1;    // flip to signed
40     #20;         // small delay til next loop
41
42     // n = 0000 -> 1111
43     for(integer i = 0; i < 16; i = i + 1)
44         #10 n = i;
45
46     #10 $finish;
47 end
48
49
50 endmodule
```

This gave the following waveforms in GTKWAVE:

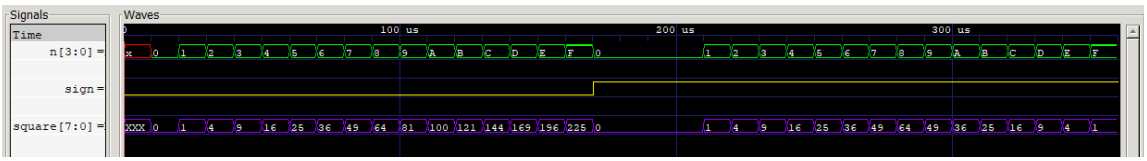

Figure 1: Squarer Circuit Output Waveforms

Section B: Square Rooting Circuit

The following Verilog code implements a read-only memory (ROM) block that stores data for the square roots of 0 to 15 (unsigned). Since we cannot set a register directly to a decimal with a non-integer value, we will set the register to a concatenated version containing the integer part and the decimal part, then use a continuous assignment to extract that information after the ROM has been accessed.

```
/*
    Chase Lotito - SIUC
    ECE426 - Lab 5 - Design of Memory in Verilog
    Section B: ROM Square Root Circuit
        -> sq_root root unsigned 0-15
            -> provide 3 decimals
*/

module rom_b (n, sq_root);

// I/O
input [3:0] n;                // unsigned number
output reg [11:0] sq_root;    // answer = n**1/2

wire [1:0] sq_root_int;       // integer part of sq_root
wire [9:0] sq_root_dp;        // decimal part of sq_root

// procedural block to compute roots as inputs come in
always @ (n)
begin
    case (n)
        0 :sq_root <= 0;
        1 :sq_root <= {2'd1, 10'd0};
        2 :sq_root <= {2'd1, 10'd414};
        3 :sq_root <= {2'd1, 10'd732};
        4 :sq_root <= {2'd2, 10'd0};
        5 :sq_root <= {2'd2, 10'd236};
        6 :sq_root <= {2'd2, 10'd450};
        7 :sq_root <= {2'd2, 10'd646};
        8 :sq_root <= {2'd2, 10'd828};
        9 :sq_root <= {2'd3, 10'd0};
        10 :sq_root <= {2'd3, 10'd162};
        11 :sq_root <= {2'd3, 10'd317};
        12 :sq_root <= {2'd3, 10'd464};
        13 :sq_root <= {2'd3, 10'd606};
        14 :sq_root <= {2'd3, 10'd742};
        15 :sq_root <= {2'd3, 10'd873};
        default: sq_root <= 12'd0; // reset by default
    endcase
```

```
40 end
41
42 // continuous assignments for int and decimal parts
43 assign sq_root_int = sq_root [11:10];
44 assign sq_root_dp = sq_root [9:0];
45
46 endmodule
```

To simulate the functionality of this ROM block, I sent in binary values 0000 to 1111 into the circuit to see the values we get out.

```
1  /*
2      chase lotito
3      lab 5 section b
4      rom square root testbench
5  */
6
7  `include "./rom_b.v"
8  `timescale 1us / 1us
9
10 module rom_b_tb();
11
12 // GTKWAVE (waveform simulator)
13 initial begin : GTKWAVE
14     $dumpfile("rom_b.vcd");
15     $dumpvars(0, rom_b_tb);
16 end
17
18 // I/O
19 reg [3:0] n;              // signed number
20 wire [11:0] sq_root;      // square root of n
21
22 // Initialize the Squarer ROM
23 rom_b U1 (
24     .n(n), .sq_root(sq_root)
25 );
26
27 // Send 4-bit numbers 0000 -> 1111
28 initial begin : stimulus
29
30     // n = 0000 -> 1111
31     for(integer i = 0; i < 16; i = i + 1)
32         #10 n = i;
33
34     #10 $finish;
35 end
36
```

```
37
38 endmodule
```

This gave the following waveforms in GTKWAVE:

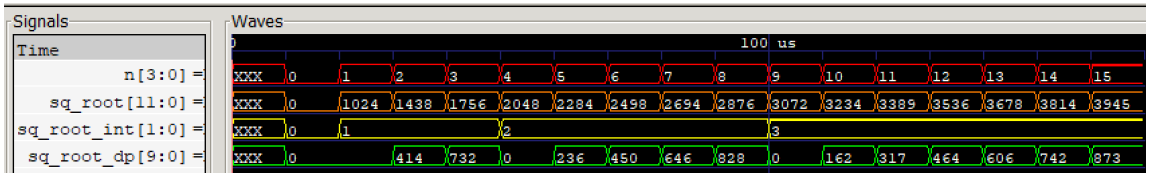

Figure 2: Square Rooter Circuit Output Waveforms