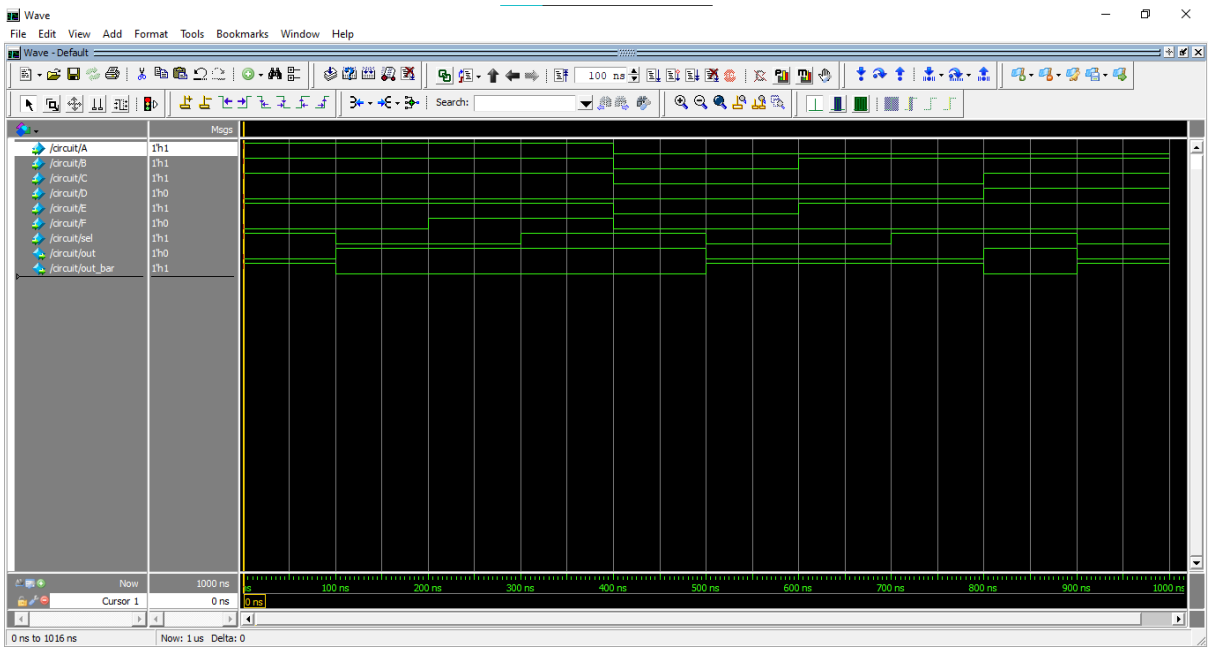
Combinational Circuit Design

1. The design has 7 inputs and 2 outputs

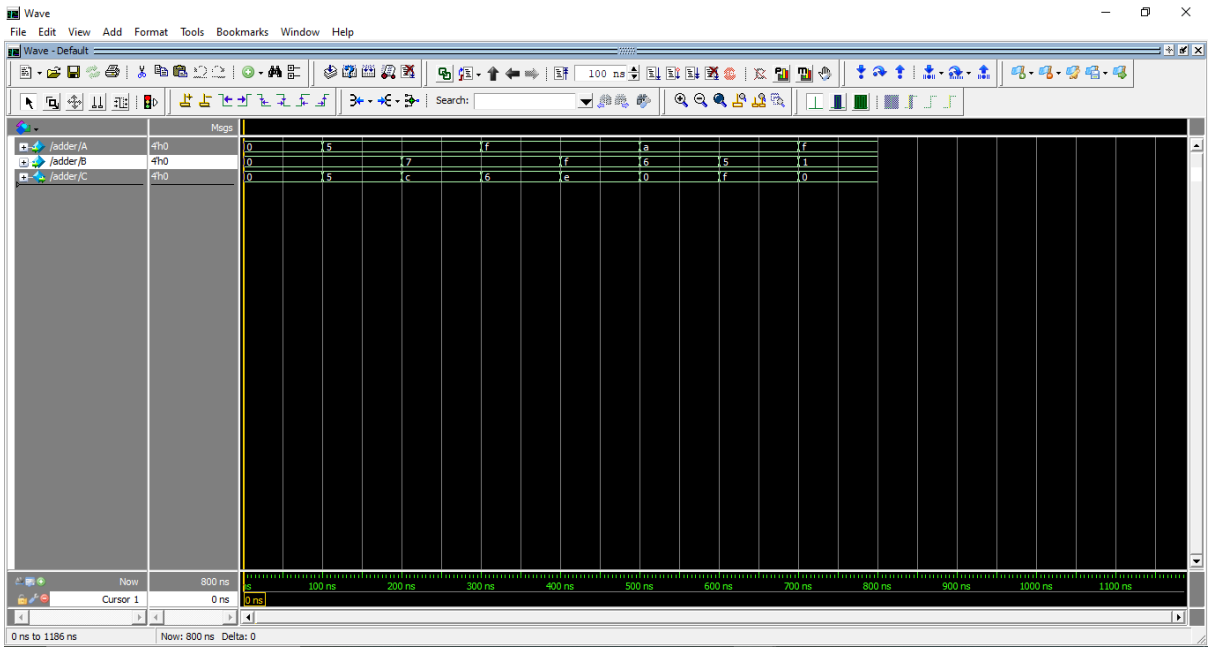
```
module circuit(
    input  A, B, C, D, E, F, sel,  // 7 inputs
    output out, out_bar            // 2 outputs
);
    // Intermediate signals for the AND1 and XNOR1 gates
    wire AND1_out, XNOR1_out;

    // AND1 gate
    assign AND1_out = A & B & C;

    // XNOR1 gate
    assign XNOR1_out = ~(D ^ E ^ F);

    // MUX1
    assign out = sel ? XNOR1_out : AND1_out;

    // NOT1 gate
    assign out_bar = ~out;

endmodule
```



2. Implement 4-bit adder using addition operator and assign statement.

The design takes 2 inputs (A, B) and the summation is assigned to output (C) ignoring the carry

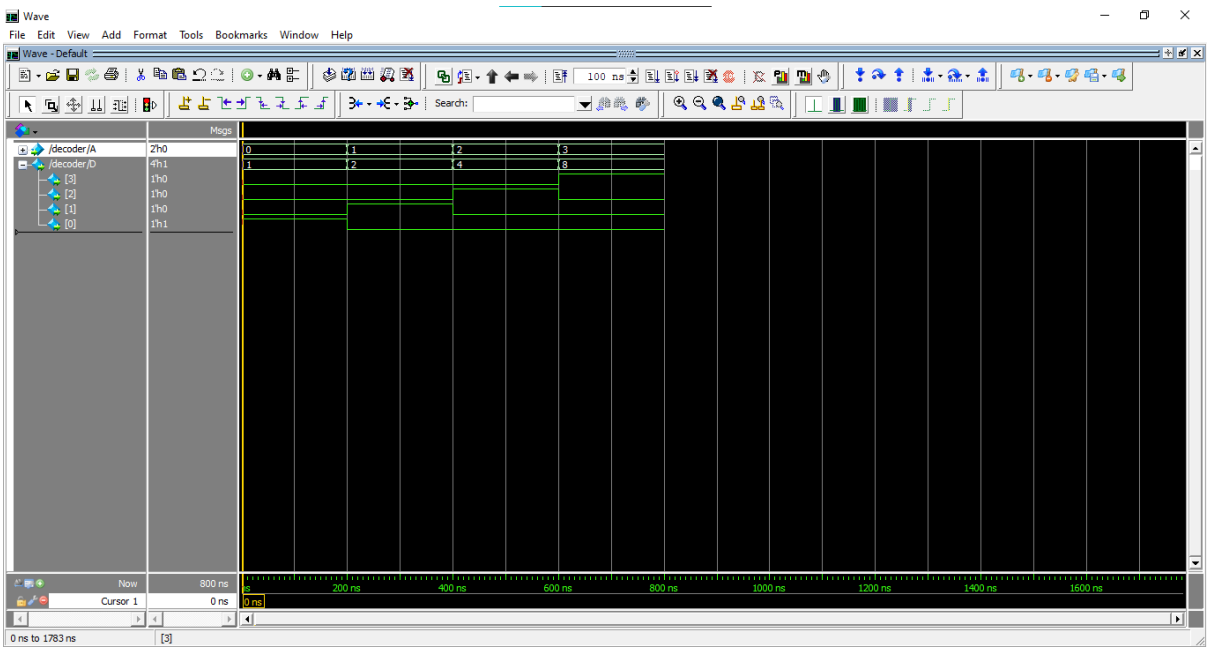
```
module adder (
    input  [3:0] A,  // 4-bit input A
    input  [3:0] B,  // 4-bit input B
    output [3:0] C   // 4-bit output C
);
    // Sum the inputs and assign only the lower 4 bits to C
    assign C = A + B;

endmodule
```



3. Implement 2-to-4 Decoder using conditional operator (A logic decoder has n input lines and 2^n output lines. Each output line corresponds to a unique combination of the input values).

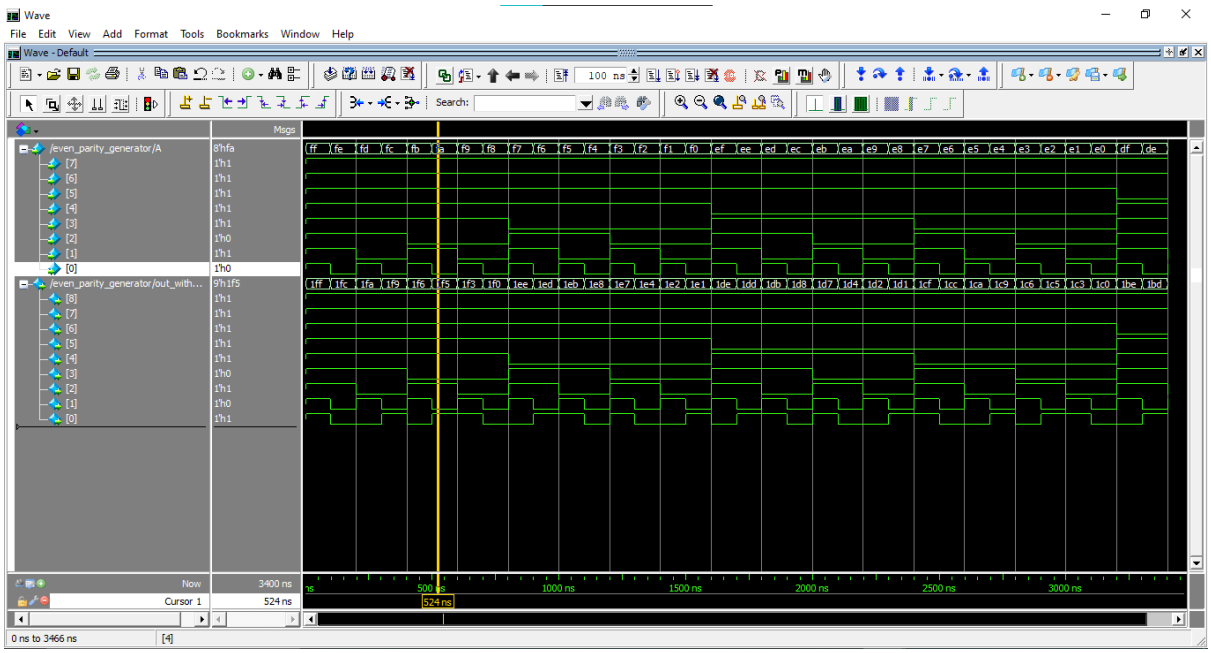
The design has input A (2 bits) and output D (4 bits) you can use the following format for the conditional operator.

```
module decoder (
    input  [1:0] A,  // 2-bit input
    output [3:0] D   // 4-bit output
);
    /*
    // Conditional assignments for each output bit
    assign D[0] = (A == 2'b00) ? 1'b1 : 1'b0;
    assign D[1] = (A == 2'b01) ? 1'b1 : 1'b0;
    assign D[2] = (A == 2'b10) ? 1'b1 : 1'b0;
    assign D[3] = (A == 2'b11) ? 1'b1 : 1'b0;
    */

    // Assign output based on A[1] first, then A[0]
    assign D = (A[1] == 1'b1) ? ((A[0] == 1'b1) ? 4'b1000 : 4'b0100) :
               ((A[0] == 1'b1) ? 4'b0010 : 4'b0001);

endmodule
```



4. The design has 1 input A (8 bits) and 1 output out_with_parity (9 bit) where the parity bit calculated will be inserted in the least significant bit of the output bus and the remaining bits will be the input A (Hint: use concatentation).

```
module even_parity_generator (
    input  [7:0] A,              // 8-bit input data
    output [8:0] out_with_parity // 9-bit output (A + parity bit)
);
    // Calculate even parity bit using XOR reduction operator
    wire parity_bit;
    assign parity_bit = ~^A; // XOR reduction, then complement for even parity

    // Concatenate parity bit with input A
    assign out_with_parity = {A, parity_bit};

endmodule
```



5. Implement a comparator that compares 2 inputs (A, B) and has 3 outputs using conditional operator.

Inputs A and B are 4-bit bus while the 3 outputs are single bits.

```
module comparator (
    input [3:0] A, B,         // 4-bit inputs A and B
    output A_greaterthan_B,   // Output high if A > B
    output A_equals_B,        // Output high if A == B
    output A_lessthan_B       // Output high if A < B
);

    // Using conditional operator
    assign A_greaterthan_B = (A > B) ? 1'b1 : 1'b0;
    assign A_equals_B = (A == B) ? 1'b1 : 1'b0;
    assign A_lessthan_B = (A < B) ? 1'b1 : 1'b0;

endmodule
```