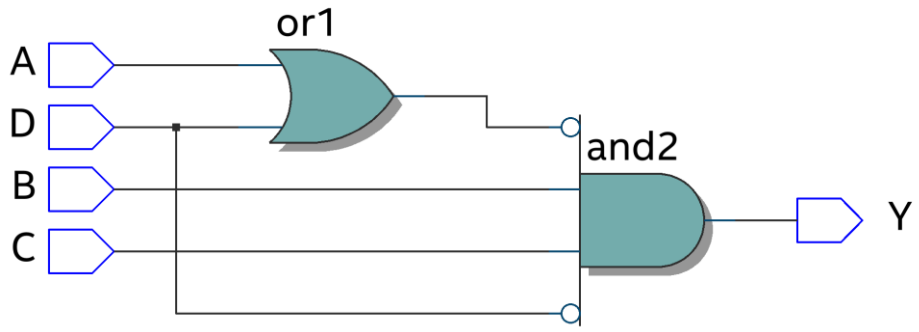
Design 1: Using Verilog gate level primitives, develop a structural model using Verilog code. Use the RTL Viewer to verify the structural design created using the Verilog code.

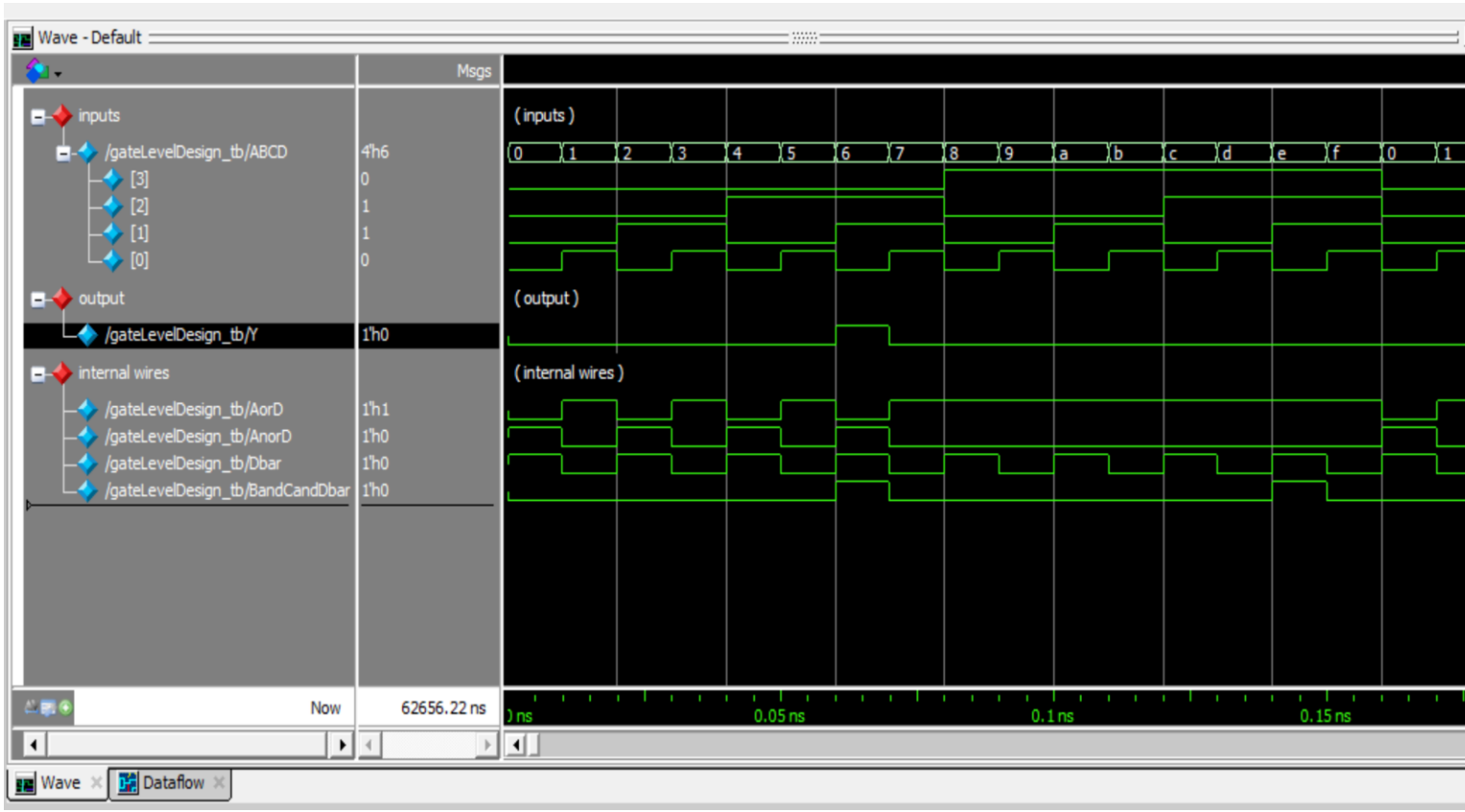
```
/*-----------------------------------------------------------------------
  Name: Nashwa Elaraby
  Lab:  Lab 1 - Structural Design using prinitive gates
 ----------------------------------------------------------------------- */

module gateLevelDesign (Y, A,B,C,D);

output Y;
input A,B,C,D;

wire AorD;
wire AnorD;
wire Dbar;
wire BandCandDbar;

or  or1  (AorD, A, D);
not not1 (AnorD, AorD);
not not2 (Dbar, D);
and and1 (BandCandDbar, B, C, Dbar);
and and2 (Y, AnorD, BandCandDbar);

endmodule
```



Quartus has optimized the design by using a 4-input AND gate. The bubbles represent a complemented input and have replaced the not gates in the structural design.

Not required for this assignment, but to verify the functionality, a testbench is included here.

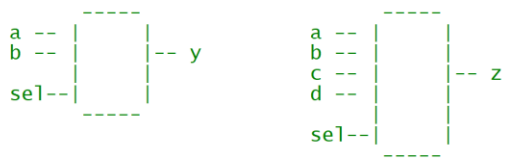
```
module gateLevelDesign_tb ();

reg [3:0] ABCD;
wire Y;

wire AorD;
wire AnorD;
wire Dbar;
wire BandCandDbar;

// instantiating the Unit Under Test

gateLevelDesign UUT (Y, ABCD[3], ABCD[2], ABCD[1], ABCD[0]);

// accessing internal signals for testing

assign AorD = UUT.AorD;
assign AnorD = UUT.AnorD;
assign Dbar = UUT.Dbar;
assign BandCandDbar = UUT.BandCandDbar;


initial
  begin
        ABCD = 4'b0000;
     forever
     #10 ABCD = ABCD + 1;
  end

 endmodule
```



We notice in the waveform figure that the output Y is only 1 when ABCD is equivalent to 4'd6 or 4'b0110. This can be verified by testing the switching function of the structural design given in this problem. The internal signals were included in the testbench to simplify debugging and troubleshooting if needed.

Design 2:

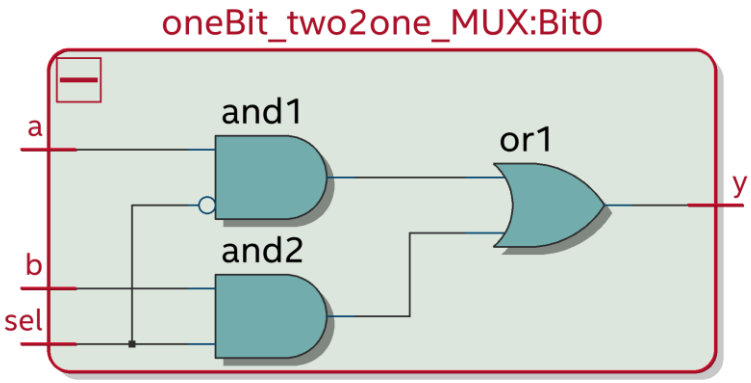
```
/*-------------------------------------------------------------------------------------------------

This file has 3 nested MUX designs. It is an example of structural
design applications. It includes 3 modules:

 oneBit_two2one_MUX (y,a,b,sel)

 fourBit_two2one_MUX (y4,a4,b4,sel)

 four2oneMUX (z4, a4,b4,c4,d4, sel)
```




```
-------------------------------------------------------------------------------------------------*/
```

(a) Write a gate-level model describing the operation of a One-bit 2-to-1 MUX. The module will have 3 inputs: **a**, **b** and **sel**, and one output **y**. If **sel** = 0, then **a** passes to the output **y**. If **sel** = 1, then **b** passes to the output **y**. Create the Karnaugh map and find the expression for the switching function for **y**.

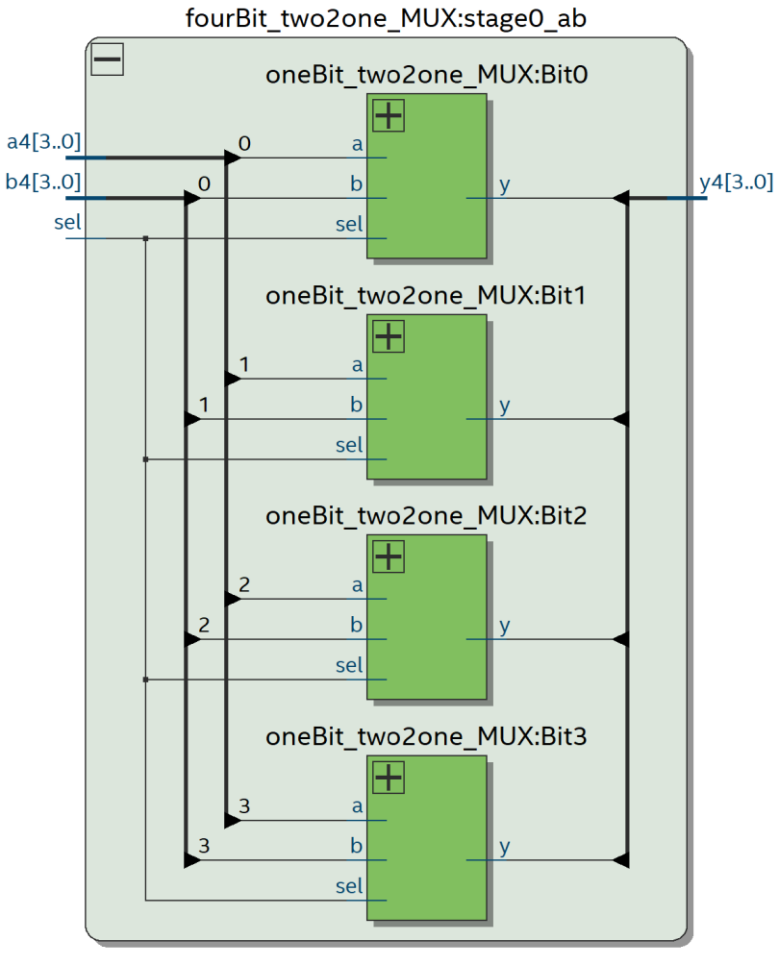
```
module oneBit_two2one_MUX (y,a,b,sel);

output  y;
input   a,b,sel;

//assign y = (a&(~sel)) | (b&sel);

wire selbar, a_and_selbar, b_and_sel;

not not1 (selbar, sel);
and and1 (a_and_selbar, a, selbar);
and and2 (b_and_sel, b, sel);
or  or1  (y, a_and_selbar, b_and_sel);

endmodule
```



Notice that in the code, the assign operator can be replaced to specify the switching function used to determine the output logic value. This is a simplified method that could replace the instantiation of primitive logic gates. The switching function was obtained using the K-map for the output y.

(b) Use the One-bit 2-to-1 MUX to create a 4Bit 2-to-1 MUX and verify the nested structure.

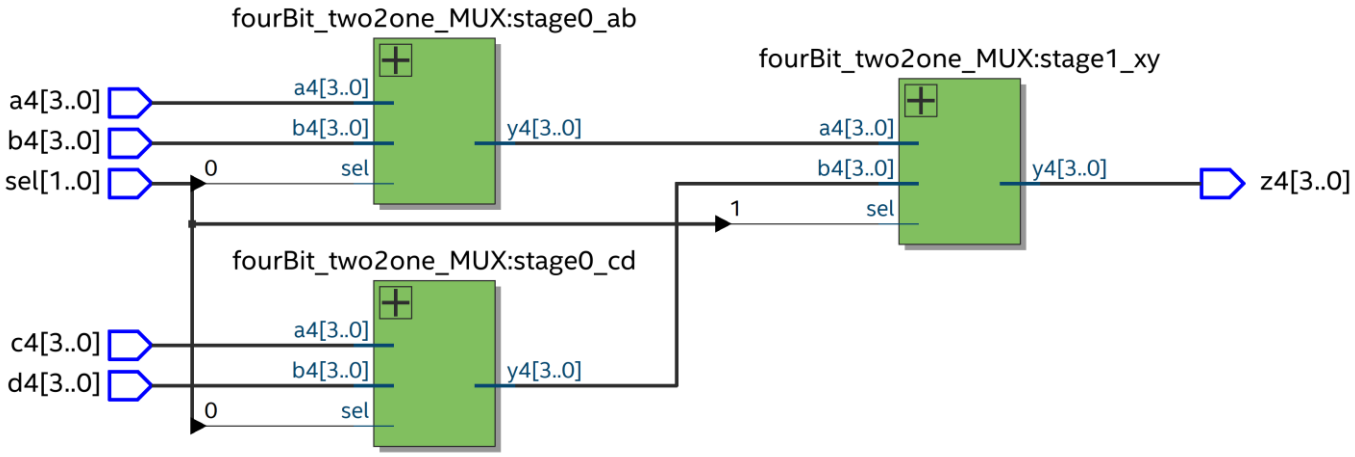
```
module fourBit_two2one_MUX (y4,a4,b4,sel);

output [3:0] y4;
input  [3:0] a4, b4;
input        sel;

// instantiating a singleBit Mux for every bit of the input pair

oneBit_two2one_MUX  Bit0 (y4[0],a4[0],b4[0],sel);
oneBit_two2one_MUX  Bit1 (y4[1],a4[1],b4[1],sel);
oneBit_two2one_MUX  Bit2 (y4[2],a4[2],b4[2],sel);
oneBit_two2one_MUX  Bit3 (y4[3],a4[3],b4[3],sel);

endmodule
```



One simple singleBit 2to1 multiplexer was instantiated to handle one bit of the 4Bit inputs of a and b.

(c) Using a nested structure create a 4Bit 4-to-1 MUX. Verify the connection using the RTL Viewer, and verify the interconnections of the instantiated MUXs.

```
/*      sel2    |   output y4
      -------------------------
          00     |     a4
          01     |     b4
          10     |     c4
          11     |     d4
      -------------------------  */

module four2oneMUX (z4, a4,b4,c4,d4, sel);

output [3:0] z4;
input  [3:0] a4, b4, c4, d4;
input  [1:0] sel;

wire   [3:0] y4, x4;

fourBit_two2one_MUX  stage0_ab (x4,a4,b4,sel[0]);
fourBit_two2one_MUX  stage0_cd (y4,c4,d4,sel[0]);

fourBit_two2one_MUX  stage1_xy (z4,x4,y4,sel[1]);

endmodule
```



The RTL viewer shows that three instances were created, and the data flow shows two stages for the input selection to pass to the output.