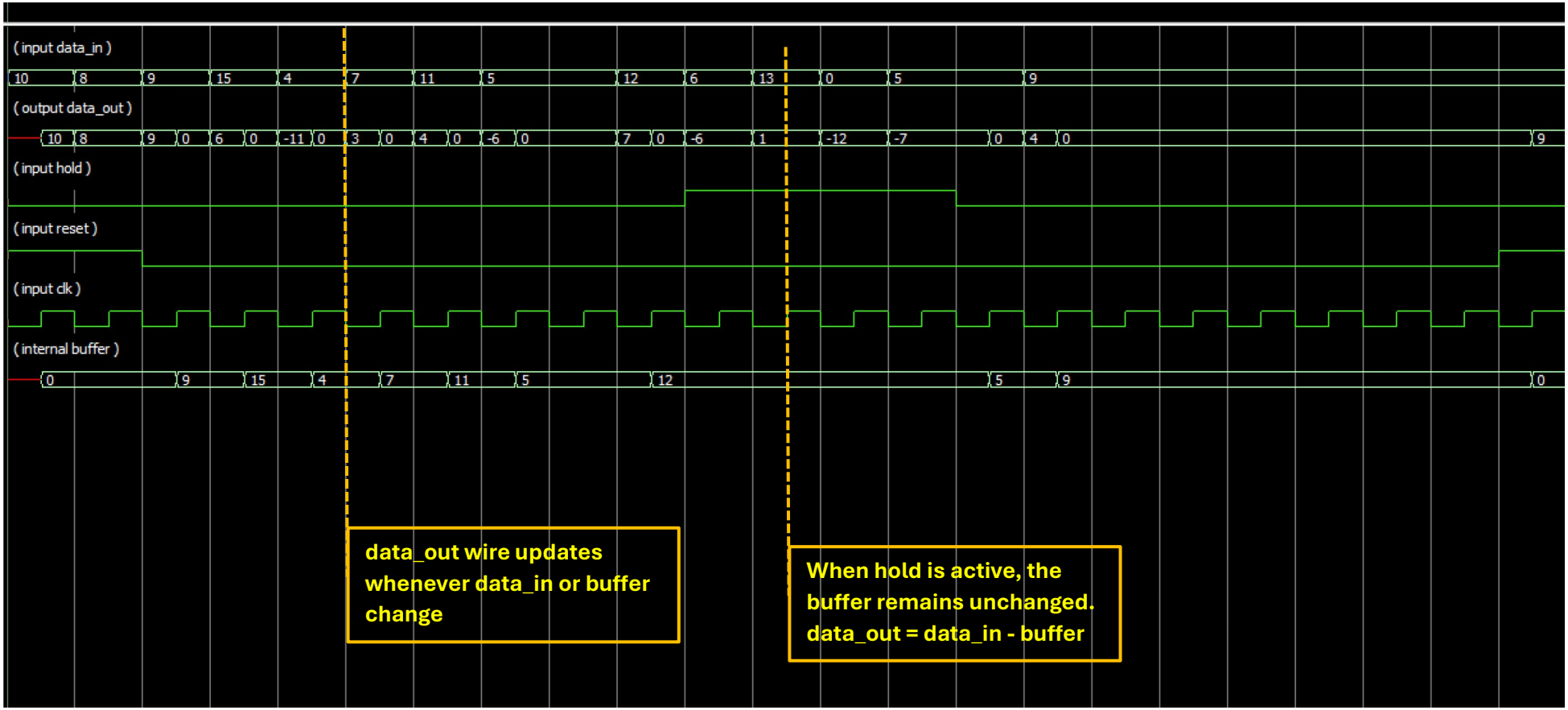
```
/*-----------------------------------------------------------------------------------
       Lesson 9: Digital Signal Processing Applications
-----------------------------------------------------------------------------------
Design a parallel differentiator. The differentiator finds the difference between
adjacent samples. (You can use the code given in the lecture notes). Create a testbench
to test the functionality of the differentiator. In the simulation, use the signed
decimal format to be able to verify whether the differentiator can correctly
report negative differences as well positive ones or not.
Combine your code for the differentiator, the test-bench, and simulation results
in one pdf file.
-----------------------------------------------------------------------------------*/

module Differentiator (data_in, data_out, hold, reset, clk);

parameter word_size = 8;
input    [word_size-1 :0]    data_in;
output   [word_size-1 :0]    data_out;
input                        hold, reset;
input                        clk;

reg      [word_size-1 :0]    buffer;

assign   data_out = data_in - buffer;

always @ (posedge clk)
     if (reset)  buffer <= 0;
else if (hold)   buffer <= buffer;
else             buffer <= data_in;

endmodule
```

```
module Differentiator_tb ();

parameter word_size = 8;
reg    [word_size-1 :0]    data_in;
wire   [word_size-1 :0]    data_out;
reg                        hold, reset;
reg                        clk;

wire   [word_size-1 :0]    buffer;

Differentiator UUT (data_in, data_out, hold, reset, clk);

assign buffer = UUT.buffer;

initial
begin       clk = 0;
forever #5  clk = ~clk; end

initial
begin      reset = 1;
      #20  reset = 0;
      #200 reset = 1;
      #20  reset = 0; end

initial
begin       hold = 0;
      #100  hold = 1;
      #40   hold = 0; end


initial
begin       data_in = 10;
      #10   data_in = 8;
      #10   data_in = 9;
      #10   data_in = 15;
      #10   data_in = 4;
      #10   data_in = 7;
      #10   data_in = 11;
      #10   data_in = 5;
      #10   data_in = 5;
      #10   data_in = 12;
      #10   data_in = 6;
      #10   data_in = 13;
      #10   data_in = 0;
      #10   data_in = 5;
      #10   data_in = 5;
      #10   data_in = 9;
      end

endmodule
```


(input data_in)
10 8 9 15 4 7 11 5 12 6 13 0 5 9
(output data_out)
10 8 9 0 6 0 -11 0 3 0 4 0 -6 0 7 0 -6 1 -12 -7 0 4 0 9
(input hold)
(input reset)
(input clk)
(internal buffer)
0 9 15 4 7 11 5 12 5 9 0
data_out wire updates whenever data_in or buffer change
When hold is active, the buffer remains unchanged. data_out = data_in - buffer