EE 417 - Integrator DataPath Controller Design Example

Assignment:

Consider the following code of an integrator data-path.

```
module integrator_parallel # (parameter word_size = 3) (
                        output   reg     [word_size +1: 0]   data_out,
                        input            [word_size -1: 0]   data_in,
                        input                                hold,
                        input                                clock, reset, buffer_data_in  );
always @ (posedge clock)  begin
      if (reset)    data_out  <= 0;
else if (buffer)  data_out  <= data_in;
else if (hold)    data_out  <= data_out;
else              data_out  <= data_out + data_in;
end
endmodule
```

1. Write down the code for a controller that would control the operation of the integrator, based on the input from the ports of a top module to give the following operation. The top module, if enabled, should accumulate groups of 4 consecutive samples. If not enabled, it should hold the last value it reached.

   Test data for enabled operation:

   Data_in: 1 2 3 4 5 6 7 1 4 6 1 0 ....

   Data_out: 1 3 6 10 5 11 18 19 4 10 11 11 ....

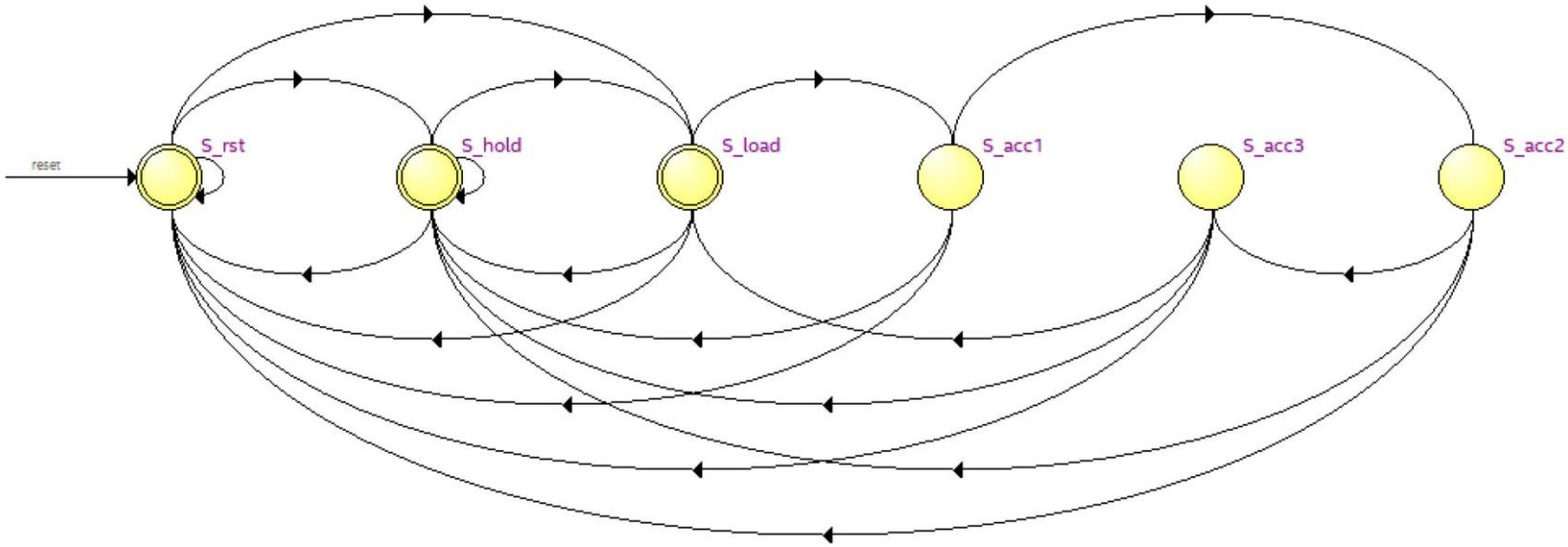
2. Write the down the code for the top module and include the block diagram that shows the data path, the controller and the signals.

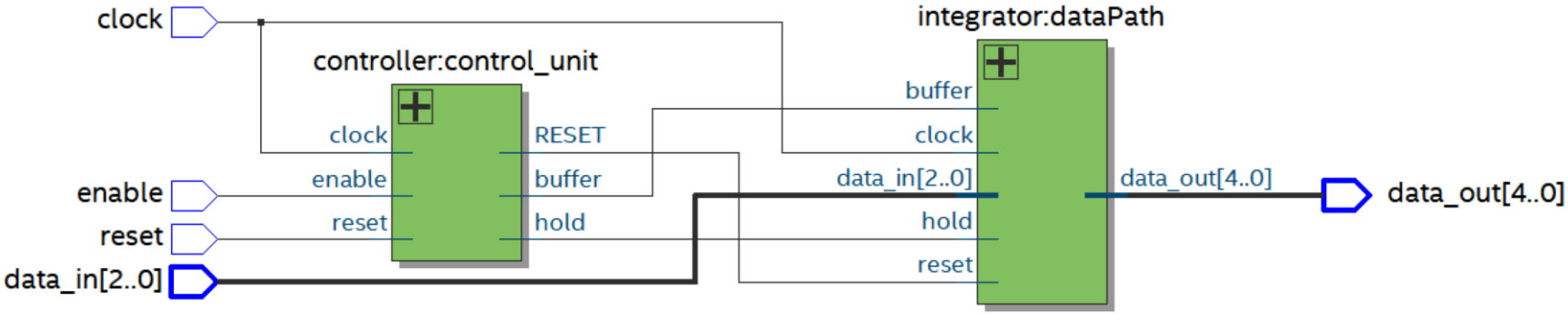
3. Create a testbench that verifies the functionality of your design.

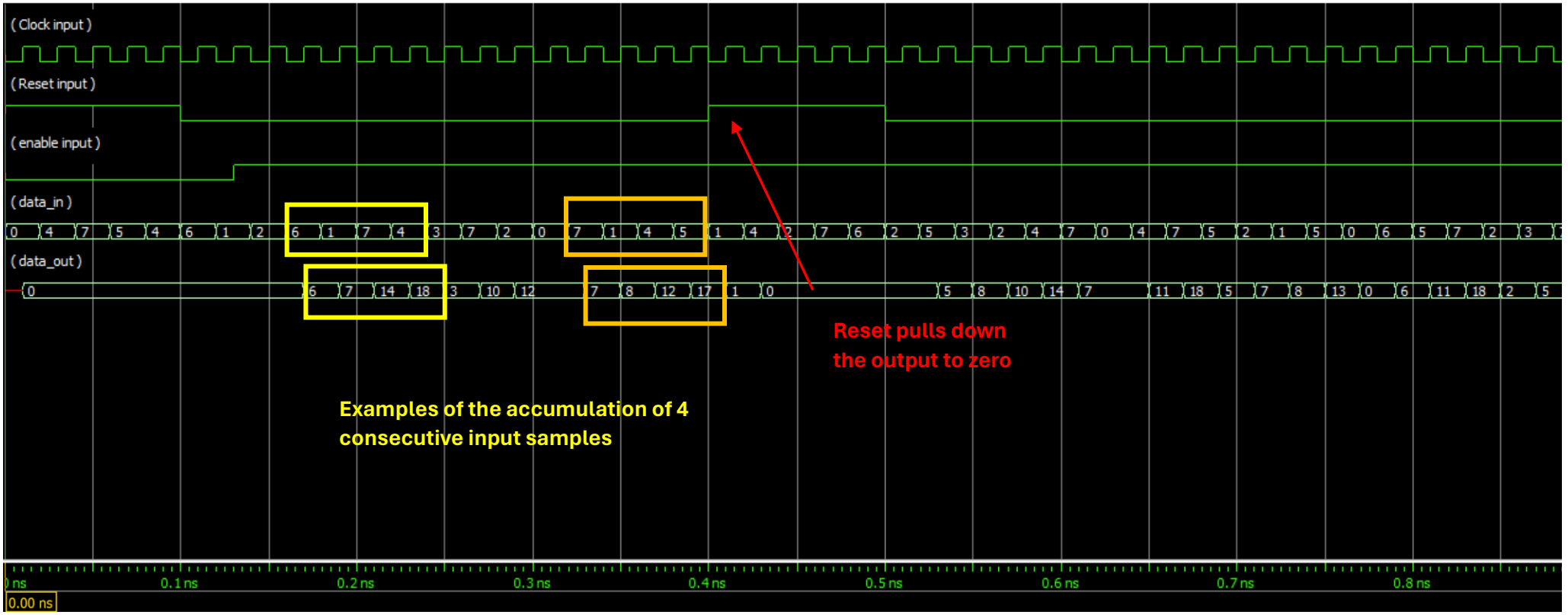
```
/*-------------------------------------------------------------------------
  The top module, if enabled, should accumulate sets of 4 consecutive
  samples. If the module is not enabled, it should just hold the last
  value it has reached. If the module is reset it will output a zero.

  After the hold is removed the controller will load a new sample as
  the first value in the set of 4 samples to accumulate.
-------------------------------------------------------------------------*/

module Integrator_DataPath_Controller #(parameter word_size = 3) (
         output     [word_size+1:0] data_out,
         input      [word_size-1:0] data_in,
         input                      clock, reset, enable);

wire  hold, buffer, RESET;

// Instantiation of the dataPath module
integrator dataPath (
         .data_out (data_out),  // output port
         .data_in  (data_in),   // input port
         .hold     (hold),      // internal input
         .clock    (clock),     // input port
         .reset    (RESET),     // input reset
         .buffer   (buffer));   // internal input

controller control_unit (
         .hold   (hold),        // internal output
         .buffer (buffer),      // internal output
         .RESET  (RESET),       // internal output
         .enable (enable),      // input port
         .clock  (clock),       // input port
         .reset  (reset) );     // input port

endmodule
```

```
module integrator #( parameter word_size = 3) (
         output reg [word_size+1:0] data_out,
         input      [word_size-1:0] data_in,
         input                      hold,
         input                      clock, reset,
         input                      buffer);

always @ (posedge clock)
     begin
          if (reset)  data_out  <= 0;
     else if (buffer) data_out  <= data_in;
     else if (hold)   data_out  <= data_out;
     else             data_out  <= data_out + data_in; end

endmodule


module controller  (
         output reg     hold,         // internal output
         output reg     buffer,       // internal output
         output reg     RESET,        // internal output
         input          enable,       // input port
         input          clock,        // input port
         input          reset );      // input port

reg [2:0] state, next_state;

parameter   S_rst  = 3'b000;  // resets the data_out and the integrator
parameter   S_load = 3'b001;  // enables loading the data-in onto the dataout
parameter   S_acc1 = 3'b010;  // Accumulates 2 samples
parameter   S_acc2 = 3'b011;  // accumulates 3 samples
parameter   S_acc3 = 3'b100;  // accumulates 4 samples
parameter   S_hold = 3'b101;  // holds the last value reached at the output

always @ (posedge clock)
  if (reset) state <= S_rst;
  else state <= next_state;
```

```
always @ *
  begin
     RESET  = 1'b0;    // default value is a zero unless changed for a state
     buffer = 1'b0;    // We could also list all the outputs for every state
     hold   = 1'b0;    // Same effect
        case (state)
             S_rst  :  begin
                       RESET = 1'b1;
                       if (enable) next_state = S_load;
                          else     next_state = S_hold;    end
             S_load :  begin
                       buffer = 1'b1;
                       if (enable) next_state = S_acc1;
                          else     next_state = S_hold;    end
             S_acc1 :  begin
                       if (enable) next_state = S_acc2;
                          else     next_state = S_hold;    end
             S_acc2 :  begin
                       if (enable) next_state = S_acc3;
                          else     next_state = S_hold;    end
             S_acc3 :  begin
                       if (enable) next_state = S_load;
                          else     next_state = S_hold;    end
             S_hold :  begin
                       hold = 1'b1;
                       if (enable) next_state = S_load;
                          else     next_state = S_hold;    end
           default :  begin
                       RESET = 1'b1;              // reset the output as the default case
                       next_state = S_rst; end    // reset the FSM
         endcase
   end
endmodule
```


reset
S_rst
S_hold
S_load
S_acc1
S_acc3
S_acc2


clock
enable
reset
data_in[2..0]
controller:control_unit
clock
enable
reset
RESET
buffer
hold
integrator:dataPath
buffer
clock
data_in[2..0]
hold
reset
data_out[4..0]
data_out[4..0]


(Clock input)
(Reset input)
(enable input)
(data_in)
(data_out)
Examples of the accumulation of 4 consecutive input samples
Reset pulls down the output to zero
0.1 ns
0.2 ns
0.3 ns
0.4 ns
0.5 ns
0.6 ns
0.7 ns
0.8 ns
0.00 ns