RTL Design Project

Consider the following module which unpacks 7-bit values from a 32-bit input stream of data.

```
//
// Input interface:
// ================
// ready_out     Output; Set ready_out to true when there is room to accept a new word, else false.
// valid_in      Input; When true AND ready_out is true, a word is deemed received by this module
// data_in[31:0] Input; LSB-aligned 32-bit data word
// sop_in        Input; True when this is the first data word for a packet
// eop_in        Input; True when this is the last data word for a packet
//
// Output interface:
// =================
// valid_out    Output; True for each cycle where a value is presented
// data_out     Output; Output value, only valid when valid_out == 1
// sop_out      Output; Present along with the first valid value for a packet
// eop_out      Output; Present along with the last valid value for a packet
//
// Requirements:
// =============
// 1) Values extracted from data_in are LSB-first.
// 2) Valid Packets start with start of packet (sop_in) flag and end with an end of packet flag (eop_in).
//    Values received after an eop_in but before sop_in should be discarded.
// 3) When eop_in is received, residual state should be cleared after the last output value is presented
//    so the next packet can be processed cleanly.
// 4) The general use case would be expected to have a packet length which is a multiple of 7-bit output
//    words. The upper bits of the last sample should be zeroed out if the packet is not a multiple of
//    7-bit words.
// 5) If valid data is ready at the input there must be a continuous stream of output values (no gaps).
//    Some latency between the first word in and the first value out is to be expected.
// 6) If the Input stream is sending data, minimize the # of dead cycles (ideally 0) between packets
//    on the output.
//
// Example:
// ========
//       sop_in eop_in  value_in                                          data_out     sop_out  eop_out
// 0:      1      0     32'b1111_0000_0000_1100_1100_0000_0101_1010
//                                                                     7'b101_1010     1        0
//                                                                     7'b000_0000     0        0
```

```
//                                                                    7'b011_0011     0         0
//                                                                    7'b000_0000     0         0
// 1:      0      0     32'b0111_1101_0000_0000_0000_0000_0000_0111
//                                                                    7'b111_1111     0         0
//                                                                    7'b000_0000     0         0
//                                                                    7'b000_0000     0         0
//                                                                    7'b000_0000     0         0
//                                                                    7'b111_1101     0         0
// 2:      0      0     32'b0000_0000_0000_0000_0000_0000_0010_0000
//                                                                    7'b100_0000     0         0
//                                                                    7'b000_0000     0         0
//                                                                    7'b000_0000     0         0
//                                                                    7'b000_0000     0         0
//       .....     .....     .....     .....     .....     .....     .....     .....     .....
// 6:      0      1     32'b1111_1110_0000_0000_0000_0000_0000_0000
//                                                                    7'b000_0000     0         0
//                                                                    7'b000_0000     0         0
//                                                                    7'b000_0000     0         0
//                                                                    7'b000_0000     0         0
//                                                                    7'b111_1111     0         1
//
module data_unpack
(
  input wire         clk,
  input wire         rst,

  output logic       ready_out, // Can be used to back pressure the input data stream
  input wire         valid_in,
  input wire [31:0]  data_in,
  input wire         sop_in,
  input wire         eop_in,

  output logic       valid_out,
  output logic [6:0] data_out,
  output logic       sop_out,
  output logic       eop_out
);

// Code Body (to be filled…)

endmodule
```

Please provide (deliverables):

1) **RTL Design & Verification**
   a. System Verilog (*preferred*), Verilog or VHDL implementation of the block.
   b. Timing closure considerations and constraints.
   c. Follow the best design practices considering modularity, readability and maintainability of code.
   d. RTL Code can be tested, compiled using free version of the simulator available or EDA playground
      i. Modelsim: https://fpgasoftware.intel.com/?edition=pro&product=modelsim_ae
      ii. EDA Playground: https://www.edaplayground.com/
   e. Develop a simple test Environment to verify the DUT described above

2) **Power-point Presentation**
   a. Your solution (Walk-through of design and verification strategy).
   b. Assumptions (Explanation for design decisions).
   c. Architecture, Micro-Architecture diagrams.
   d. Possible Error Scenarios and mitigation techniques.
   e. Timing Diagrams, Waveform Snapshots (if applicable).