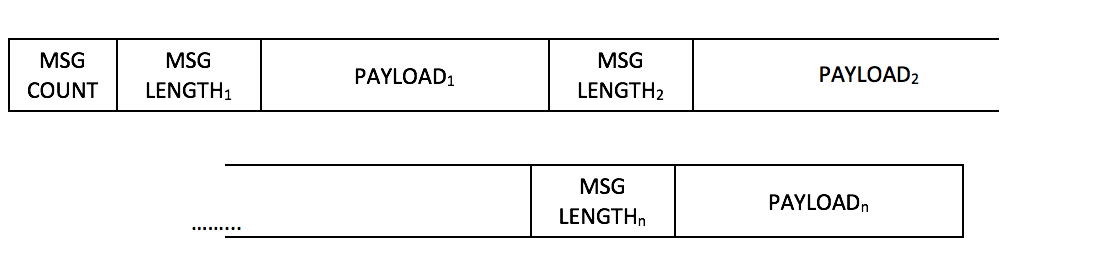
**IMC FPGA Engineer Assignment**

**Message Extractor**

All exchanges have a custom protocol in which they disseminate data and accept data from their customers. The objective of the assignment is to design and implement a message extractor.

The format of the incoming data stream is given below.

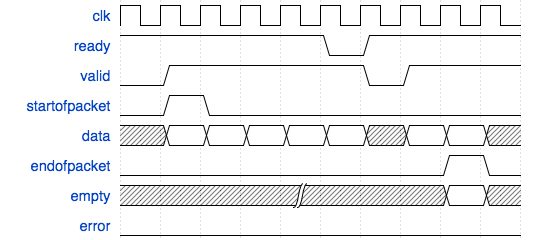


|  |  |  |
| --- | --- | --- |
| Field name | Length | Description |
| Message Count | 2 bytes | Number of messages in the packet |
| Message Length | 2 bytes | Length of the following message (excluding this field) |
| Payload | Variable | Payload data |

The expected output of the block is the payload of these messages.

**Input setup**

1. The input of the module is a 64 bit Avalon Streaming interface. The I/O signals are given below.



|  |  |  |  |
| --- | --- | --- | --- |
| Signal Name | Direction | Width (bits) | Description |
| clk | Input | 1 | Clock |
| reset\_n | Input | 1 | Active low reset |
| in\_ready | Output | 1 | Indicates when the sink module (module being designed) is ready to accept data. Read Latency =1 |
| in\_valid | Input | 1 | High when incoming data is valid, 0 otherwise |
| in\_startofpacket | Input | 1 | High for the 1st clock cycle of incoming valid data, 0 otherwise |
| in\_endofpacket | Input | 1 | High for the last clock cycle of incoming valid data, 0 otherwise |
| in\_data | Input | 64 | Incoming payload |
| in\_empty | Input | 3 | Indicates the number of bytes that are empty during cycles that contain the end of a payload. Should only be qualified with incoming end of packet. |
| in\_error | Input | 1 | A bit mask used to mark errors affecting the incoming data being transferred in the current cycle. |

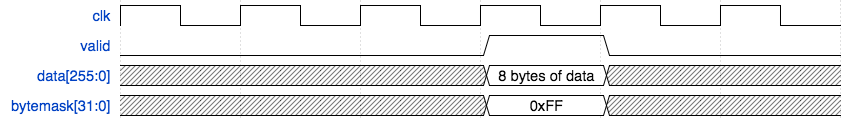
1. Assume that the minimum message length for any message is **8 bytes** and the maximum is **32 bytes**. The total size of the entire stream can be a maximum of **1,500 bytes**.
2. Assume **in\_error is always 1’b0**.

**Output setup**

1. The output signals of the module are given below.

|  |  |  |  |
| --- | --- | --- | --- |
| Signal Name | Direction | Width (bits) | Description |
| clk | Input | 1 | Clock |
| reset\_n | Input | 1 | Active low reset |
| out\_valid | Output | 1 | High when out going data is valid, 0 otherwise |
| out\_data | output | 256 | Outgoing payload |
| out\_bytemask | Output | 32 | Indicates the number of bytes valid in the payload. |

For example, if the message length of a message reads 8 bytes, the expected output would be the 8 bytes of the payload in out\_data bus with an out\_bytemask of 32’hFF qualified by an out valid.



**Questions**

1. Draw the finite state machine for the chosen design.
2. Write an elegant, synthesizable solution for the message extractor in Verilog/SystemVerilog or VHDL and verify it against the given reference file.
3. In your opinion, what is the maximum frequency that the design can run at?
4. Please explain how would your design change if the range of message lengths change from [8,32] bytes:
5. To [1,32] bytes?
6. To [8,256] bytes?
7. What are the trade-offs for the chosen approach?

Please write down all other assumptions that you make.

**Example Packet**

**Sample Input**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **in\_data [63:0] (hex)** | **in\_startof-packet** | **in\_endof-packet** | **in\_valid** | **in\_empty** | **in\_error** |
| 0008000862626262 | 1 | 0 | 1 | X | 0 |
| 62626262000c6868 | 0 | 0 | 1 | X | 0 |
| 6868686868686868 | 0 | 0 | 1 | X | 0 |
| 6868000a70707070 | 0 | 0 | 1 | X | 0 |
| 707070707070000f | 0 | 0 | 1 | X | 0 |
| 7a7a7a7a7a7a7a7a | 0 | 0 | 1 | X | 0 |
| 7a7a7a7a7a7a7a00 | 0 | 0 | 1 | X | 0 |
| 0e4d4d4d4d4d4d4d | 0 | 0 | 1 | X | 0 |
| 4d4d4d4d4d4d4d00 | 0 | 0 | 1 | X | 0 |
| 1138383838383838 | 0 | 0 | 1 | X | 0 |
| 3838383838383838 | 0 | 0 | 1 | X | 0 |
| 3838000b31313131 | 0 | 0 | 1 | X | 0 |
| 3131313131313100 | 0 | 0 | 1 | X | 0 |
| 095a5a5a5a5a5a5a | 0 | 0 | 1 | X | 0 |
| 5a5a | 0 | 1 | 1 | 6 | 0 |

**Note:** in\_valid can be de-asserted at any time after data starts streaming in.

**Sample Output**

|  |  |  |
| --- | --- | --- |
| **out\_data (hex)** | **out\_bytemask (binary)** | **out\_valid** |
| 6262626262626262 | 32’b00000000\_00000000\_00000000\_11111111 | 1 |
| 686868686868686868686868 | 32’b00000000\_00000000\_00001111\_11111111 | 1 |
| 70707070707070707070 | 32’b00000000\_00000000\_00000011\_11111111 | 1 |
| 7a7a7a7a7a7a7a7a7a7a7a7a7a7a7a | 32’b00000000\_00000000\_01111111\_11111111 | 1 |
| 4d4d4d4d4d4d4d4d4d4d4d4d4d4d | 32’b00000000\_00000000\_00111111\_11111111 | 1 |
| 3838383838383838383838383838383838 | 32’b00000000\_00000001\_11111111\_11111111 | 1 |
| 3131313131313131313131 | 32’b00000000\_00000000\_00000111\_11111111 | 1 |
| 5a5a5a5a5a5a5a5a5a | 32’b00000000\_00000000\_00000001\_11111111 | 1 |