ATOMOS FPGA Engineering Technical Interview Questions

Question 1:

Write Verilog/VHDL code for a single-cycle pulse synchronizer, which transfers a single-cycle pulse from one clock domain to another.

The module has the following ports:

- `clk1 (input)`
- `clk2 (input)`
- `pulse_clk1 (input)`
- `pulse_clk2 (output)`

The purpose of the module is to synchronise the `pulse_clk1` signal from the `clk1` domain to the `clk2` domain. The `pulse_clk1` input is asserted for a single cycle, and should be synchronised as a single-cycle pulse on the `pulse_clk2` output, synchronous to `clk2`.

The relationship between `clk1` and `clk2` is arbitrary: i.e. clk1 could be slower or faster than clk2 – the module should work either way.

If there are multiple pulses on the `clk1` side which are 'close' together, only the first pulse needs to be transferred.

Question 2:

Write VHDL/Verilog for a module that packs variable length data symbols into a 32-bit output word.

The module has the following ports:

- `clk (input)`
- `reset (input)`
- `din[31:0] (input)`
- `din_width[4:0] (input)`
- `din_valid (input)`
- `flush (input)`
- `dout (output)`
- `dout_valid (output)`

Data input (`din`) is valid whenever `din_valid` is high – and the number of bits in the symbol is indicated by `din_width`. When 32-bits of data have been accumulated, `dout_valid` is driven high, and the 32-bits of accumulated data are diven on `dout`. The `flush` input causes a valid data word to be output regardless of whether a full 32-bits have been accumulated. The value `din_width` is one less than the actual number of valid bits of `din` – a value of `0x0` indicates that `din` has a single valid bit and a value of `0x1f` indicates a full compliment of 32 valid bits.

Example input / output sequence. Latency for `dout` here is shown as 1 cycle (illustrative only), but this is not important – the latency can be any value.

din_valid	**din_width**	**din**	**flush**	**dout_valid**	**dout**
1	0x7	0x000000aa	0	0	0x00000000
1	0x7	0x000000bb	0	0	0x00000000
1	0x7	0x000000cc	0	0	0x00000000
1	0x7	0x000000dd	0	0	0x00000000
1	0x9	0x00000012	0	1	0xddccbbaa
1	0x9	0x00000345	0	0	0xddccbbaa
1	0x11	0x00000678	0	0	0xddccbbaa
1	0xf	0x0000abcd	0	1	0x678d1412
1	0x9	0x00000345	0	0	0x678d1412
0	0x0	0x00000000	1	0	0x678d1412
0	0x0	0x00000000	0	1	0x0345abcd

Question 3:

A memory arbiter on an FPGA has the following simple interface, designed to multiple requesters. Each requester connection to the arbiter has the following signals:

Signal Name	Description	Direction	Width
a_valid	Request address and length valid	input	1
a_addr	Request address	Input	32
a_len	Request length	Input	8
a_ready	Request ready. Request accepted when this signal is high. Ignored when low.	output	1
d_wvalid	Write data valid	input	1
d_wdata	Write data	input	32
d_wready	Write data ready. Data accepted when high, ignored when low.	output	1
d_rvalid	Read data valid	output	1
d_rdata	Read data	output	32
d_rready	Read data ready. Data accepted when high, ignored when low.	input	1

Request channels (address and burst length) and data channels are independent and can be stalled at the requester using the "ready" signals. Multiple requests can be made before any data is present.

The implemented design has the multiple requesters placed far away from the arbiter, resulting in long nets, which causes timing problems. Design a scheme to connect the requesters to the arbiter which eliminates this problem.

IMPORTANT: The scheme must guarantee that back-to-back requests can be made, and accepted, so that there are no gaps in the flow of requests or data. This is to ensure that the chosen scheme does not reduce memory data throughput.

Question 4:

Write RTL code (either VHDL or Verilog) for a module which manages the following signals for a 1024 entry asynchronous FIFO – these are all outputs from the module:

```
read_pointer
write_pointer
read_empty
write_full
read_fill
write_fill
```

The FIFO as the following input signals:

```
read_clk
write_clk
read_enable
write_enable
reset_n
```

All `read_` signals are synchronous to `read_clk`, and all `write_` signals are synchronous to `write_clk`. The `reset_n` input is an asynchronous reset.

Question 5:

The following code implements a 3 input adder block, where the output should be the sum of three inputs one cycle after the inputs are valid. The code has some errors – what are they?

```
entity adder_three_input is
    port (
       clk : in  std_logic;
       a   : in  unsigned(7 downto 0);
       b   : in  unsigned(7 downto 0);
       c   : in  unsigned(7 downto 0);
       z   : out unsigned(7 downto 0)
    );
end  adder_three_input;

architecture rtl of  adder_three_input is

    signal a_plus_b : unsigned(7 downto 0);

begin
    process (clk)
    begin
       if rising_edge(clk) then
           a_plus_b <= a + b;
           z <= c + a_plus_b;
       end if;
    end process;
end rtl;
```