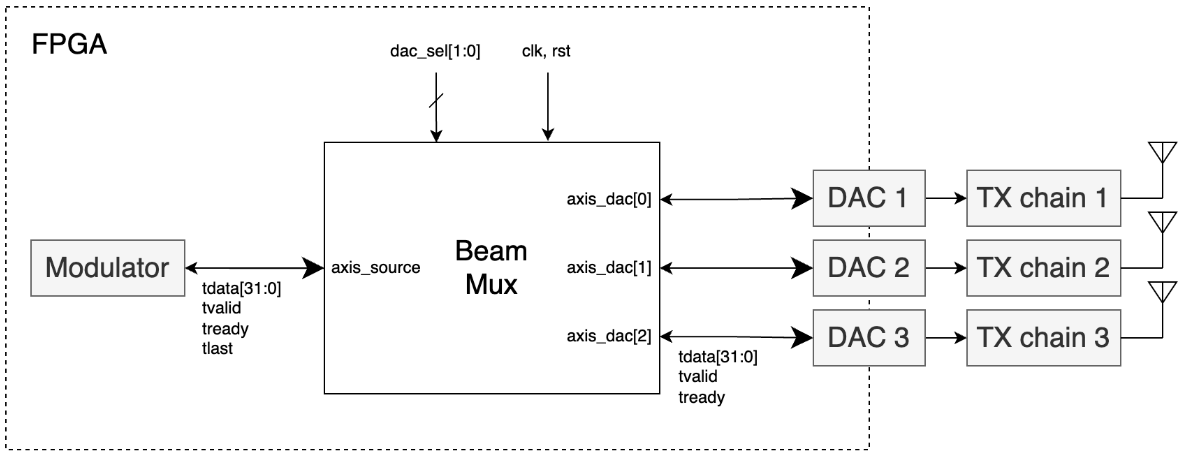
You are tasked with creating a beam multiplexer module, which takes a sample stream from a modulator and routes it to a DAC that feeds a signal to an antenna. The data interfaces use the AXI-Stream protocol.



Each transmission out of the modulator is bursty, and will have a duration between 1024 and 65536 samples. The end of the burst is signified by `axis_source.tlast` being asserted on its final sample.

The modulator will output valid samples at arbitrary rates—it may not always have a valid sample available. The Beam Mux must support arbitrary-rate samples coming in at the `axis_source` interface. The modulator also supports backpressure from the Beam Mux via `tready`.

A DAC should receive an uninterrupted stream of samples; once data is fed into it, its `axis_dac[n].tvalid` should be asserted every clock cycle throughout the duration of the burst. However, the interface to the DAC can provide backpressure using `tready`, so it may not consume a valid sample on every cycle. (Imagine there's a variable interpolator as part of the signal processing chain inside the DAC.)

`dac_sel` is used to select which DAC is being fed samples. When `dac_sel` is updated, the DAC currently receiving samples should finish its burst before samples are fed to the newly selected DAC.

`dac_sel` routes samples as follows:

- `dac_sel = 0`: DAC selection operates in a round-robin fashion; one burst gets output through DAC 1, the next through DAC 2, and so on
- `dac_sel = 1`: all samples flow through DAC 1
- `dac_sel = 2`: all samples flow through DAC 2
- `dac_sel = 3`: all samples flow through DAC 3

DACs that are not currently transmitting or selected should not be fed valid samples.

System latency between samples being output from the modulator and output by the DAC should be minimized, and the downtime of no DAC being fed samples should be kept to a minimum.

The design will be running on a Xilinx Ultrascale+ FPGA. All inputs, outputs, and resets are assumed synchronous; assume `clk` to be running at a 300 MHz rate.

This take-home should be completed in about an evening’s worth of time.

Write your code and test cases in SystemVerilog.

Please provide ample comments, test cases, and a write-up of your design and verification.