DSLX

1. Update the `mul4` function below to use the DSLX standard library functions to implement a 4-bit multiplier (don't forget the `std::` prefix).
2. Generate the verilog for the design.
3. Run the OpenLane flow up until synthesis.
4. Observe the change in the complexity of the graph.
5. Compare to the results w/ the previous adder design.

```
[ ] %%bash -c 'cat > user_module.x; interpreter_main user_module.x'
    import std

    fn mul4(a: u4, b: u4) -> u8 {
      u8:0 // TODO(YOU) implement mul4
    }

    fn user_module(io_in: u8) -> u8 {
      mul4(io_in[0:4], io_in[4:8]) as u8
    }

    #[test]
    fn test() {
      let _ = assert_eq(mul4(u4:8, u4:8), u8:64);
      let _ = assert_eq(user_module(u8:0b1000_1000), u8:0b0100_0000);
      _
    }
```