1. Combinational Logic Practice

The last major building block you haven't implemented yet are the shifters. Implement **at least one of the following**, though I hope you'll find that once you've got one working, the other two are trivial.

- Shift Left Logical, or SLL:
  - Takes an input, and then shifts it to the left by shamt (shift_amount). This is also how you multiply numbers by powers of two.
- Shift Right Logical, or SRL:
  - Takes an input, and then shifts it to the right by shamt (shift_amount). This is also how you divide numbers by powers of two.
- Shift Right Arithmetic, or SRA:
  - Takes a **signed** input and shifts it to the right by shamt. Since this is division, we should be preserving the sign (msb) of the input. The way we do this is make sure that we fill in the missing bits with whatever the input's original sign (msb) bit was.

1.

	in				Out			
shamt	3	2	1	0	3	2	1	0
1	0	0	1	0	0	1	0	0
2	0	0	1	0	1	0	0	0
3	0	0	1	0	0	0	0	0
4	0	0	1	0	0	0	0	0

input wire [$clog2(N)-1:0] shamt
input wire [N-1:0] in;
output wire [N-1:0] out

comb logic out = (in << shamt)

shamt = 2

0	0
1	1
2	2
3	3

og index + shamt >= N → 0+2 %4 = 2
if ≥N discard
1+2 %4 = 3
2+2 %4 = 0
3+2 %4 = 1

2. Parallel to Serial Practice

To get data out of our FPGA we often have to take our busses (parallel wires) and convert them into a serial stream of bits. There is however always a point of contention between sending the MSB first, or sending the LSB first! It can vary from device to device, so we're going to modify a working MSB first parallel to serial converter to be able to work as either using a direction input.

- The parallel_to_serial_converter.sv file contains mostly implemented logic and a description of the io.
- The testbench runs, and if you do make waves_parallel_to_serial_converter you can start with a populated gtkwave view to make debugging easier.
- This is a more realistic assignment than designing from scratch, most of the world is modifying existing (mostly) working designs.
- Bonus: add some more test cases to the testbench and keep visualizing! Can you add in random testing? How would you implement a checker for this module?

inputs: clk, rst, i_valid, direction, i_data
outputs: out, o

N bit wide or i_data
if i