Liberty Exercise Set 3

The default LSE delay module uses the standard control semantics and is thus only controlled by back pressure. In this exercise you will hierarchically wrap the delay to create a pipe_register which can be stalled via an external control signal in addition to back pressure. You will then extend the pipe_register to have a clear port so that you can flush the pipeline registers.

To accomplish this, you will need to use the gate module. The gate module has data inputs and control inputs and a user function which determines, based on the control inputs, whether the data inputs should pass to the outputs. You can think of the gate module as a “transistor” that can control the flow of data.

The gate module has two input ports, in and control and a single output port, out. The gate defines a user function called gate_control that will be invoked for each input connection (remember, all the ports are multiports). The user function is provided the port index number, port status, data, and dynID for the connection as well as the data and port status for all the connections to the control port. The function must return 1 if the data is to be passed to the output or 0 otherwise. The function can return -1 if it cannot yet make a decision. For reference, the signature of the user point is:

\[
\text{parameter gate\_control : userpoint(<int porti, LSE\_signal\_t status, LSE\_dynid\_t id, LSE\_port\_type(in) *data, LSE\_signal\_t *cstatus, LSE\_dynid\_t *cid, LSE\_port\_type(control) **cdata, int cwidth>>> => <<<int>>>);
\]

The control flow semantics of the gate are fairly simple. With the default parameterization, data and enable pass from in to out if the user function for the particular port instance returns 1. Otherwise the output is set to LSE\_signal\_nothing | LSE\_signal\_disabled. Similarly, the acknowledge signal flows from out to in if the user function returns 1. Otherwise, the value of the acknowledge signal is controlled by the parameter gate_style. If this parameter is set to gate::blocker then the input is negatively acknowledged (LSE\_signal\_nack). If the parameter is set to gate::filter, then the input is acknowledged (LSE\_signal\_ack). The control port is always acknowledged on all port instances.

With this information, you should now be ready to start the exercise.

1. Create a module called pipe_register. Give the module two input ports: in and stall. Give the module a single output port named out. The types of input and output should be polymorphic but should be equal. The type of stall should be int. If the stall signal is 1, then the pipe register should hold its current value and should not output or enable its contents. For this part of the exercise, we will disallow multiple input connections to the in and control port. Add checks to your module to ensure this is the case.

   In your module, instantiate a delay module and a gate module. Connect the in port to the in port of the delay instance. Connect the out port of the delay instance to the in port of the
gate. Connect the out port of the gate to the out port of the module. Connect the stall port to the control port of the gate.

Fill in the gate.control user function on the gate such that it passes the data through if the stall input is 0 and such that it prevents data from flowing otherwise. Make sure that your user function handles cases where cstatus[0] has unknown data and enable signals. Your user function will be reinvoked after the data and enable is known. Your user function can return -1 to indicate that it cannot yet make a decision.

Finally decide whether the instance of the gate module should be configured as a gate::filter or gate::blocker to create the correct stall behavior.

2. Create a small configuration to test to see if your pipe.register module works as you’d expect.

3. Now add multiport behavior to the input and output ports of your module. Its behavior should be similar to the multiport behavior of the delay instance.

4. Finally, add multiport behavior to the stall input. You can decide the specific semantics. For example, you could make any asserted stall signal stall the pipe register. Alternatively, you could make it so that they must all agree in order for the stall to occur.

5. Add a second gate module. Connect this module after the first gate. Connect this instance’s control inputs to a new port called drop. Configure the gate so that when drop is asserted, the delay loses its current value and does not output it.