CSEE 3827: Fundamentals of Computer Systems

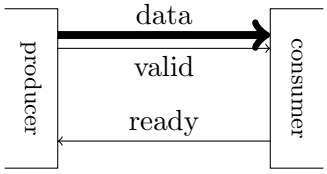
Project #2

Token Replicator

Due 10/24/17 at 11:59pm

1 Background

Latency insensitive interface. This machine transfers input and output tokens using latency-insensitive interfaces, as depicted below.



This unidirectional interface transfers one data token per cycle when both ready and valid are high. Token "flow" is controlled by the valid and ready signals from the producer and consumer respectively. A producer may stop sending data by pulling its valid signal low, and a consumer may stop receiving data by pulling its ready signal low. A token is transferred only when both ready and valid are high. Until a token is successfully transferred, it is the responsibility of the producer to continue to proffer the data, lest the value be dropped.

2 Function

The REPLICATOR machine accepts sequences of two, two-bit tokens. The first token specifies a data value. The second specifies the number of copies to create. The machine should emit the specified number of copies to create. So, if the machine receives a 2 followed by a 3, it should emit 2, 2, 2. If it receives a 0 followed by a 2 it should emit 0, 0. And if it receives 1 followed by 0, it should emit nothing. Upon reset the machine should stop immediately and return to a state where it is expecting the first command token.

3 Quality

Try to minimize the total gate cost. As in the first assignment, each gate has unit cost regardless of number and negation of inputs. Splitters and pins are free. Each D flip-flop has a cost of 9.

4 Scaffolding

The test harness works largely as in the first assignment. The harness feeds input commands from a ROM to the REPLICATOR module and outputs are compared against the expected ones. Matching and not-matching totals are accumulated in NUM_PASS and NUM_FAIL.

There are a couple new additions:

- The harness will timeout after a specified number of clock cycles. If your design hangs or does not produce the expected outputs within this the simulation will halt.

- The harness speaks the latency insensitive protocol and will randomly offer/accept data to/from the REPLICATOR module.
- The test cases include all 16 possible commands, for a total of 32 input tokens and 24 output tokens.

5 Rules and Regulations

- Your design must use ony AND, OR, NOT, NOR, NAND, XOR, XNOR, or D-flip-flops.
- Do not change the name or appearance of the REPLICATOR module.
- Simulation of the test harness must halt without error (e.g., oscillations or undefined wire values). Incorrect results ($NUM\_FAIL > 0$) are fine.
- Submissions must be made via courseworks.
- Upload a *single* .circ file, whose names contains only alpha-numeric characters, hyphens, or underscores (no spaces, parenthesis, etc.). Do not upload lib3827.circ.
- [Update 10/6/17: The uploaded file should have no external libraries loaded (i.e., external dependencies) except for lib3827.]

6 Scoring Rubric

To be graded, your submission must adhere to all rules and regulations. We will test your module on all 16 possible commands **in a different random order from the scaffolding**.

	Total	Formula
Function	70 pts	$\lceil 70 \times \frac{NUM\_PASS}{24} \rceil$
Quality	30 pts	If fully functional, $30 - (GateCost - 60)$ with a max of 30 min of 0.
Style	10 pts	If fully functional, extra credit to ten (or more) most beautiful schematics.

7 Hints

- There is no automatic process to arrive at a solution.
- Search for ways to break the problem into smaller pieces. Tackling it as one FSM is unlikely to be tractable.
- When decomposing, ask yourself questions such as "When should X be true?" or "When should Y happen?" From the answers to these questions, you can build your expressions and then simplify before building the circuit. Doing this can also provide insights into which values are useful/reusable.
- Test harness distributed with the scaffolding randomly proffers input tokens and accepts output tokens, assigning random values to VAL_IN and RDY_IN. For initial tests you may find it helpful to modify the harness to *always* offer and accept input and output tokens (VAL_IN = RDY_IN = 1).