Basic RTL Design

Synchronous (positive-edge triggered) Register Transfer Level (RTL) designs implements a given functionality by using registers (Flip-Flops) and combinational logic. The registers store the system state, which is updated on every positive clock edge. The next state is determined by the combinational logic from the current state (register outputs) and the primary inputs.

In this exercise, you implement a few basic functions which are repeatedly used in the class and serve as a basis for many RTL designs. Specify the RTL architecture for these functions as RTL block diagrams, i.e., **you do not have to write any VHDL code for this exercise.**

Hand-in instructions: Prepare a small report with your solutions (detailed). Submit your report as a PDF through the lecture moodle per the moodle submission deadline.

Task 1: Pulse Generator

Your block in this task has the following ports with directions:

- `CLKxCI` (input): $100\,\text{MHz}=10\,\text{ns}$ clock.
- `RSTxRI` (input): Active-high aynchronous reset that is only asserted once during power up (HIGH='1') and inactive (LOW='0') after.
- `X` (output): Pulse which is HIGH ('1') for $10\,\text{ns}$ every microsecond.

Draw the RTL block diagram describing this circuit. Note that you can measure pulse length in different ways, i.e., you can use the start of the pulses or the gap between the pulses to measure 1 microsecond. Picking one or the other will just result in a slightly different timing diagram compared to Figure 1 where we have also added a register before output `X`.

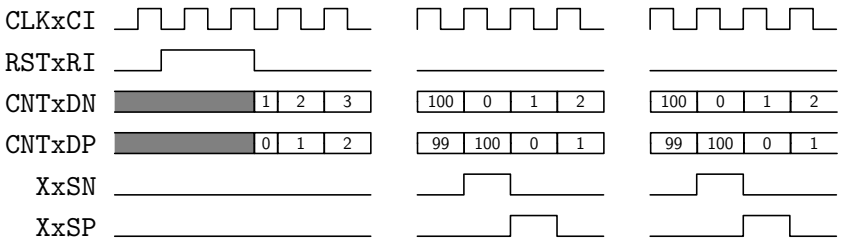

Figure 1: Pulse generator timing diagram with a pulse every microsecond ($1000\,\text{ns}$).

Hints and common errors

Some important things to remember in this class:

- Remember the rules of synchronous design at all times!
- Do not make your life unnecessarily complicated! Implement the circuit with standard components like adders, boolean gates, multiplexers, D flip-flops (no T flip-flops!) and so forth. Do not implement standard components like adders with logic gates, you should just directly use an adder as a standard component.

Circuit designer's toolbox

We often find that many students will immediately start drawing schematics without having fully understood the problem they are trying to solve. **A clear, concise problem statement is always the first step to solve any programming or hardware design problem.**

As the class progresses, you will have to deal with problem statements becoming less detailed and you have to fill in the gaps. Examples of elaborating the problem description include:

- Writing down a list of ports with their directions and purpose as shown in Task 1.
- Drawing a timing diagram to show the circuit behavior as done in Figure 1. For a task where counters are used, you can also add the counter value to relate this to signal changes, i.e., at what counter value should other signals do something particular.
- If you have a larger circuit with many signals, you can do several small timing diagrams where you only include the subset of signals relevant for describing certain cases.

These practices, while simple, are a good way to better understand the circuit you are implementing and it also serves as a form of documentation.

Task 2: Up-Counter with Conditional Enable

The input of your block in this task are two 1-bit signals `AxSI` and `BxSI` as well as a clock `CLKxCI` and an asynchronous reset `RSTxRI` that is only asserted once during power up (HIGH) and inactive (LOW) after. Implement a circuit that counts and outputs the number of clock cycles in which both inputs are '1'.

Draw the RTL block diagram describing this circuit. You may consider elaborating on our task description by listing the ports and their purpose as done in Task 1.

Task 3: Rising Edge-Detector

The input of your block in this task is a 1-bit signals `AxSI` as well as a clock `CLKxCI` and an asynchronous reset `RSTxRI` that is only asserted once during power up (HIGH) and inactive (LOW) after. The input signal `AxSI` can change from '0' to '1' and from '1' to '0' at random moments in time. Implement a circuit that outputs (signal `X`) a '1' for only a single clock cycle every time the input transitions from '0' to '1'.

Draw the RTL block diagram describing this circuit.