Lab 8: Complex Control

TA: Andrew White
Nov 3, Nov 5, Nov 10

1 Overview

This lab is a further continuation of the FSM/Controller design, where we will add to the traffic controller created in the last lab. This lab will also look at the *generic statement* in an entity declaration, and utilize this to create generic width devices.

2 VHDL Constructs

2.1 Generic Statement

Up to this point, we have only considered fixed sized elements and the VHDL necessary to create these elements. Recreating the same circuit element, a different size every time to target the current design being developed, becomes unnecessarily redundant. VHDL allows for the description of circuit elements in terms of variable sizes. For example, rather then creating a 16-bit register, a N-bit register can be described. When that register component is used in a design, the actual width of the device will be specified.

VHDL uses a *generic statement* inside the entity declaration, which allows parameters that *will be fixed at compile time* to be passed to the design. This generic statement will allow for a parameter like the width of the inputs and outputs to be specified. Attached to this lab is an example of a generic width register and the test bench to test the register. When instantiating a component that uses generic statements in the entity, a generic map is used. This is similar to the way the *port map* of a component instantiation relates to the port statement of the entity. This generic statement can also be used to specify delays, bounds for loops, or any other parameter or constant that the designer wishes to be unique to each instance of use.

3 Lab

3.1 Description

In this lab you will create a **Synchronous, Structural, Generic Width Binary Up-Counter**. This counter will be used by the traffic light con-

troller created in the previous lab. The functionality of the traffic controller will be extended to include the use of counters to delay switching the flow of traffic from the main street to the cross street as well as to only give the cross street a green light for a predefined amount of time; after which the flow of traffic will be switched back to the main street. For each instance where a counter is needed, it will be necessary create states for reseting and/or enabling the counter, and also waiting for the counter to reach the predetermined terminating value. The key to integrating the counters is to enumerate all needed control sequences as states.

3.2 Procedure

The counter you design must be described structurally in terms of flip-flops and any basic gates ((N)AND, (N)OR, INV). These base components can be described using behavioral VHDL and can be of generic width if needed. The type of flip-flop used as well as the design of the actual flip-flop will be entirely up to you, but should have an enable and an asynchronous reset. The counter must be of a variable width so that the same counter can be used to realize several different sizes. The counter should have the following properties:

- Generic Width, where *bit size* of the counter is specified through the generic interface.
- Synchronous design (all flip-flops must use the common clock).
- Asynchronous reset.
- Described **structurally** using flip-flops (of your choice) and basic gates.
- Counting in ascending binary order.

The counter created will be used in **two separate instances of different sizes** in the traffic controller. Though this is not optimal, it illustrates that the same counter component can be instantiated twice in the same design using differing generic parameters at the same time. The behavior of the FSM will be modified as follows:

- When the controller starts, the main road's traffic lights display a green light and the cross road displays a red light.

- When the sensor for the cross road is triggered, a counter starts counting and continues for 50 clock cycles. After which, the main road's traffic lights will cycle from green to yellow and then to red.
- Once (after) the main road's traffic lights have turned red, the cross road's traffic lights will turn green.
- Once the cross road's traffic light is green, another counter starts counting and continues for 200 clock cycles. After which, the cross road's traffic lights will cycle from green to yellow, and then to red.
- Once (after) the traffic lights for the cross road have turned red, the main road's traffic lights will turn green.
- This behavior will repeat for as long as the FSM is running.

To test the design, you will need to use the test bench template given last lab to generate the clock signal necessary. Also, it will be necessary to use a **wait until** statement instead of waiting for some time period. An example of how to use this in the test bench is to wait until a green light becomes 1 until asserting a sensor input. Here is an example of this:

< **BEGIN VHDL CODE SEGMENT** >

```
-- Set sensor for East/West traffic light
sensor_EW <= '1';
sensor_NS <= '0';

-- Now wait for East/West traffic light to turn green
wait until green_EWLight = '1';

-- Now assert the North/South sensor
sensor_EW <= '0';
sensor_NS <= '1';

-- Now wait for North/South traffic light to turn green
wait until green_NSLight = '1';

-- Halt test bench
sim_end <= '1';
wait;
```

< **END VHDL CODE SEGMENT** >

3.3 Report

For the lab report, you will need to create a block diagram of the structural layout of the counter. This block diagram can be created either **neatly and clearly** by hand or (preferably) using a software drawing program (visio, paint, xfig, etc.). Credit will be deducted if the bock diagram is illegible and unreadable. The following items need to be submitted along with the cover sheet:

- Block diagram of structural layout for counter
- VHDL code for the flip-flop, counter, traffic controller, and test bench
- Waveform for simulation