Lab 7: More FSM Design

TA: Andrew White
Oct 22, Oct 29, Oct 31

1 Overview

This lab is a continuation of FSM design using VHDL.

2 VHDL Constructs

2.1 Enumerated Types

As the name describes, VHDL–*VHSIC Hardware Description Language* is used to describe the hardware or circuit in terms of either behavior, structure, or in terms of the data-flow through the system. In FSM design, it is often convenient to abstract the behavior of the circuit from the implementation. User specified enumerated types are one way to aide in this. Rather than use arrays of type *std_logic* or *bit*, new types can be created whose values are symbolic of the FSM they are describing. The syntax for this is given in Figure 1 and an example in Figure 2.

type *type_name* **is** (*type_definition*);

Figure 1: Syntax for a type definition.

2.2 State Encoding With Enumerated Types

When defining new enumerated types, it is important to note that the encoding of each successive value will be in *binary*. For example, in Figure 2

–Define the type
type fsm_states **is** (stInit, st1, st2, st3)
– Declare a signal of the new type
signal curstate : fsm_states

Figure 2: Example of type used in four state FSM.

attribute enum_encoding : string;
attribute enum_encoding **of** *type_name* **: type is**
"*value1_encode value2_encode ...*";

Figure 3: Syntax for enumerated type value encoding.

where there are four states, *stInit* will be encoded as value **00** and *st3* will be encoded as **11**. As discussed in the last lab, binary state encoding might not be desirable. VHDL allows us to define an attribute that can be set to allow for the encoding of the values of an enumerated type to be changed. This attribute is defined and set in the declarative region of the architecture with the type definition. The syntax is given in Figure 3 and an example for the type example in Figure 2 is given in Figure 4. State encoding is most often unspecified and left to the synthesis tool to handle. For the Xilinx tools, this is set under Synthesis/Implementation Options.

3 Lab

3.1 Description

For this lab, you are to design a FSM that controls a four-way intersection. The FSM will control traffic for an intersection of two streets, the *main road* running north-south (**NS**) and the *cross road* running east-west (**EW**). The traffic controller will give the outputs (*green, yellow,* and *red*)for one traffic light for each street (the light outputs for a street will be used for the traffic lights of both directions of that street). The intersection uses two sensors to indicate when a car is present on a particular street; **sensor_NS** for cars on the street running north-south, and **sensor_EW** for cars on the street running east-west.

The behavior of the traffic controller is as follows:

attribute enum_encoding : string;
attribute enum_encoding **of** fsm_states **: type is** "00 01 11 10";

Figure 4: Example of type used in four state FSM.

- 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, 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.
- The cross road's traffic lights will remain green until the sensor for the main road is triggered. Then the cross road's traffic lights will cycle 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.

The functionality of this traffic controller is very limited and will be added to in future labs. Do not discard this FSM.

3.2 Procedure

Create a Moore FSM designed to implement the traffic controller described above. Define and use an enumerted state type, where you define your own values that are suffecient to cover all states in your FSM. In this lab you may leave the state encoding for your defined type as the default binary encoding.

Your FSM will have four inputs: clock, reset, sensor_NS, and sensor_EW. The FSM will have six outputs: green_NS, yellow_NS, red_NS, green_EW, yellow_EW, and red_EW. The sensor inputs will be asserted to signify that a car has reached the intersection. Only the sensor of the road whose traffic is stopped needs to be monitored. When a road's traffic light turns green, it's sensor should be deasserted after a short period to signify that there are no cars waiting. For this lab, the sensor inputs will be generated in the test bench, so it is important that you plan when to assert and deassert the sensor inputs. To test the circuit, use the VHDL test bench included as a template to create a test bench for your FSM as done in previous labs. In following labs we will make this traffic sensor more robust and handle traffic in a more effecient manner.

3.3 Report

The lab report will consist of a state diagram either neatly hand-drawn or created using a drawing program (visio, paint, xfig, etc), the VHDL code of

the FSM, the test bench used, and the simulation waveform. The waveform should be clearly labeled where a car arrives and noted that the appropriate traffic light is triggered. Test the FSM for several transitions of traffic from the two streets to show proper functionality.