Gillian Kearney

Lab Report 6

ECE 2031 L09

3 March 2022

Diagram, engineering drawing

Description automatically generated

Diagram, engineering drawing

Description automatically generated

Diagram, engineering drawing

Description automatically generated

Diagram, engineering drawing

Description automatically generated

**Figure 1.** Series of worksheets that outline the states of two trains along a rail. Note: rails assume straight unless colored differently.

**Table

Description automatically generated with low confidence**

**Table 1.** Table that outlines the possible states of a state machine and the outputs defined for each outlined state.

**Figure 2.** State machine diagram that outlines each state, the transition to the next state and the outputs that are activated upon entry into the state.

APPENDIX A

VHDL CODE IMPLEMENTING MOORE STATE MACHINE

-- TrainControler.vhd

-- Four-State Moore State Machine

-- Gillian Kearney

-- ECE 2031 L09

-- 3/3/2022

-- State machine to control trains

-- This template implements the same general path as the example

-- covered in lecture, but does not "release" trains from their

-- stopped state until the other train is completely clear of the

-- relevant sensor. This is to account for long trains, but that

-- does not affect your assignment this semester.

--

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.all;

USE IEEE.STD\_LOGIC\_ARITH.all;

USE IEEE.STD\_LOGIC\_UNSIGNED.all;

ENTITY TrainController IS

PORT(

reset, clock, sensor1, sensor2 : IN std\_logic;

sensor3, sensor4, sensor5, sensor6 : IN std\_logic;

switch1, switch2, switch3, switch4 : OUT std\_logic;

dirA, dirB : OUT std\_logic\_vector(1 DOWNTO 0)

);

END TrainController;

ARCHITECTURE a OF TrainController IS

-- Create a new TYPE called STATE\_TYPE that is only allowed

-- to have the values specified here. This

-- 1) enables using helpful names for values instead of

-- arbitrary values

-- 2) ensures that signals of this type can only have valid values, and

-- 3) helps the synthesis software create efficient hardware for the design.

TYPE STATE\_TYPE IS (

Start,

Ain,

AinBrev,

Bin,

StartBrev,

BinRev,

StartArevBrev,

AinRevBrev,

ArevBinRev,

ArevBin,

StartArev,

AinRev

);

-- Create a signal of the new type. Note that there is

-- nothing special about the names "state" or "state\_type", but it

-- makes

-- sense to use these names because those names fit their purpose.

SIGNAL state : STATE\_TYPE;

-- This creates some new internal signals which will be concatenations

-- of some of the sensor signals. This will make CASE statements

-- easier.

-- Note that the names are \*not\* what makes them concatenations of the -- relevant

-- signals; all of these signals need to be assigned values in the

-- architecture.

SIGNAL sensor12, sensor13, sensor24, sensor34: std\_logic\_vector(1 DOWNTO 0);

BEGIN

-- A process statement is required for clocked logic, such as a state

-- machine.

PROCESS (clock, reset)

BEGIN

IF reset = '1' THEN -- This state machine uses an active-high reset.

-- Reset to this state

state <= Start;

ELSIF clock'EVENT AND clock = '1' THEN

-- Case statement to determine next state.

-- Case statements are a nice, clean way to make

-- decisions

-- based on different values of a signal.

CASE state IS

WHEN Start =>

CASE Sensor12 IS

WHEN "00" => state <= Start;

WHEN "01" => state <= Bin;

WHEN "10" => state <= Ain;

WHEN "11" => state <= AinBrev;

WHEN OTHERS => state <= Start;

END CASE;

WHEN Ain =>

CASE Sensor24 IS

WHEN "00" => state <= Ain;

WHEN "01" => state <= Start;

WHEN "10" => state <= AinBrev;

WHEN "11" => state <= Bin;

WHEN OTHERS => state <= Ain;

END CASE;

WHEN AinBrev =>

CASE Sensor34 IS

WHEN "00" => state <= AinBrev;

WHEN "01" => state <= StartBrev;

WHEN "10" => state <= Ain;

WHEN "11" => state <= BinRev;

WHEN OTHERS => state <= AinBrev;

END CASE;

WHEN Bin =>

CASE Sensor13 IS

WHEN "00" => state <= Bin;

WHEN "01" => state <= Start;

WHEN "10" => state <= ArevBin;

WHEN "11" => state <= Ain;

WHEN OTHERS => state <= Bin;

END CASE;

WHEN StartBrev =>

CASE Sensor13 IS

WHEN "00" => state <= StartBrev;

WHEN "01" => state <= BinRev;

WHEN "10" => state <= ArevBin;

WHEN "11" => state <= Ain;

WHEN OTHERS => state <= StartBrev;

END CASE;

WHEN BinRev =>

CASE Sensor12 IS

WHEN "00" => state <= BinRev;

WHEN "01" => state <= StartBrev;

WHEN "10" => state <= ArevBinRev;

WHEN "11" => state <= AinBrev;

WHEN OTHERS => state <= BinRev;

END CASE;

WHEN StartArevBrev =>

CASE Sensor34 IS

WHEN "00" => state <= StartArevBrev;

WHEN "01" => state <= AinRevBrev;

WHEN "10" => state <= ArevBinRev;

WHEN "11" => state <= BinRev;

WHEN OTHERS => state <= StartArevBrev;

END CASE;

WHEN AinRevBrev =>

CASE Sensor13 IS

WHEN "00" => state <= AinRevBrev;

WHEN "01" => state <= AinRev;

WHEN "10" => state <= StartArevBrev;

WHEN "11" => state <= ArevBinRev;

WHEN OTHERS => state <= AinRevBrev;

END CASE;

WHEN ArevBinRev =>

CASE Sensor24 IS

WHEN "00" => state <= ArevBinRev;

WHEN "01" => state <= BinRev;

WHEN "10" => state <= StartArevBrev;

WHEN "11" => state <= AinRevBrev;

WHEN OTHERS => state <= ArevBinRev;

END CASE;

WHEN ArevBin =>

CASE Sensor34 IS

WHEN "00" => state <= ArevBin;

WHEN "01" => state <= Bin;

WHEN "10" => state <= StartArev;

WHEN "11" => state <= AinRev;

WHEN OTHERS => state <= ArevBin;

END CASE;

WHEN StartArev =>

CASE Sensor24 IS

WHEN "00" => state <= StartArev;

WHEN "01" => state <= AinRev;

WHEN "10" => state <= ArevBin;

WHEN "11" => state <= Bin;

WHEN OTHERS => state <= StartArev;

END CASE;

WHEN AinRev =>

CASE Sensor12 IS

WHEN "00" => state <= AinRev;

WHEN "01" => state <= AinBrev;

WHEN "10" => state <= StartArev;

WHEN "11" => state <= ArevBin;

WHEN OTHERS => state <= AinRev;

END CASE;

END CASE;

END IF;

END PROCESS;

-- Notice that all of the following logic is NOT in a process

-- block,

-- and thus does not depend on any clock. Everything here is

-- pure combinational

-- logic, and exists in parallel with everything else.

-- Combine bits for the internal signals declared above.

-- ("&" operator is concatenation)

sensor12 <= sensor1 & sensor2;

sensor13 <= sensor1 & sensor3;

sensor24 <= sensor2 & sensor4;

sensor34 <= sensor3 & sensor4;

-- The following outputs depend on the state. This is a Moore

-- state machine.

WITH state SELECT Switch1 <=

'0' WHEN Start,

'0' WHEN Ain,

'0' WHEN AinBrev,

'1' WHEN Bin,

'0' WHEN StartBrev,

'1' WHEN BinRev,

'0' WHEN StartArevBrev,

'0' WHEN AinRevBrev,

'1' WHEN ArevBinRev,

'1' WHEN ArevBin,

'0' WHEN StartArev,

'0' WHEN AinRev;

WITH state SELECT Switch2 <=

'0' WHEN Start,

'0' WHEN Ain,

'0' WHEN AinBrev,

'1' WHEN Bin,

'0' WHEN StartBrev,

'1' WHEN BinRev,

'0' WHEN StartArevBrev,

'0' WHEN AinRevBrev,

'1' WHEN ArevBinRev,

'1' WHEN ArevBin,

'0' WHEN StartArev,

'0' WHEN AinRev;

WITH state SELECT DirA <=

"01" WHEN Start,

"01" WHEN Ain,

"01" WHEN AinBrev,

"01" WHEN Bin,

"01" WHEN StartBrev,

"01" WHEN BinRev,

"10" WHEN StartArevBrev,

"10" WHEN AinRevBrev,

"10" WHEN ArevBinRev,

"10" WHEN ArevBin,

"10" WHEN StartArev,

"10" WHEN AinRev;

WITH state SELECT DirB <=

"01" WHEN Start,

"01" WHEN Ain,

"10" WHEN AinBrev,

"01" WHEN Bin,

"10" WHEN StartBrev,

"10" WHEN BinRev,

"10" WHEN StartArevBrev,

"10" WHEN AinRevBrev,

"10" WHEN ArevBinRev,

"01" WHEN ArevBin,

"01" WHEN StartArev,

"01" WHEN AinRev;

-- These outputs happen to be constant values for this solution;

-- they do not depend on the state.

Switch3 <= '0';

Switch4 <= '0';

END a;