Traffic Intersection VHDL project

EEE20001 Digital Electronics Design

Lab Day (Wed etc.)	Wednesday	Lab Time e.g. 12:30 pm	10:30
Name of Lab supervisors		Rida Fatima	

Student information (please print)		
Family Name	**Given Name**	**Student ID**
Morad	Marella	103076428
Collins	Jarron	102988098

DECLARATION AND STATEMENT OF AUTHORSHIP

1. We have not impersonated, or allowed ourselves to be impersonated by any person for the purposes of this assessment.
2. This assessment is our original work and no part of it has been copied from any other source except where due acknowledgement is made.
3. No part of this assessment has been written for us by any other person except where such collaboration has been authorised by the lecturer concerned.
4. I have not previously submitted this work for this or any other course/unit.
5. I give permission for my assessment response to be reproduced, communicated, compared and archived for plagiarism detection, benchmarking or educational purposes.

I understand that:

6. Plagiarism is the presentation of the work, idea or creation of another person as though it is your own. It is a form of cheating and is a very serious academic offence that may lead to exclusion from the University. Plagiarised material can be drawn from, and presented in, written, graphic and visual form, including electronic data and oral presentations. Plagiarism occurs when the origin of the material used is not appropriately cited.

Student signature/s

I declare that we have read and understood the declaration and statement of authorship.

Marella Morad – Jarron Collins

Further information relating to the penalties for plagiarism, which range from a formal caution to expulsion from the University is contained on the Current Students website at **www.swin.edu.au/student/**

Design Summary (from ISE Summary report)	
Number of Latches: 0 **Macrocells Used:** 25/64 (40%)	**Pterms Used:** 49/224 (22%) **Registers Used:** 21/64 (33%)

Table of Contents

1 Description of approach

Our approach consists of 3 modules, one top-level module and two lower-level modules. The top-level module is responsible for synchronising all inputs, the lower-level modules are a counter, responsible for delaying the change of state in the second lower-level module which is a state machine that controls the outputs of the traffic light system.

Top-Level (TrafficController):

The top-level entity was designed to synchronise the inputs to the circuit and to map the outputs of the lower-level entity to the inputs of the other lower-level entity. We map the inputs/outputs by creating a signal that acts as a go-between and then set the signal to one entity's output and set the other entity's input to the signal. The top-level entity also sends the global reset and clock to both lower entities and links the whole functionality into one file to be programmed into the CPLD.

Counter:

The counter is clocked at 100hz so it will increase by 100 every second. Thus, we have 3 outputs which are '1' when the counter is equal to 300, 400, and 500 which corresponds to 3, 4, and 5 seconds, respectively. The module takes a 'clear' input which resets the counter whenever it is set to '1', this input is used by the state machine to reset the time with each action. The counter simply uses a range of natural numbers from 0 to 502, it goes up to 502 just to prevent us missing the 500 because the range can loop back around when it reaches the end. Then on every rising clock edge, we increase the count by one and check if it is equal to 300, 400, or 500 setting the respective outputs to '1' if true or '0' if false.

State Machine:

Firstly, the state machine has a process that checks for reset input and sets everything back to default, this process also checks for when a pedestrian button is pressed and then sets a corresponding signal to '1' so that the pedestrian can be enabled by tapping the button at any time, no need to hold it.

Next is the state machine process. This process makes use of a StateType variable and a Case statement. At the beginning of the process, we set both traffic lights to RED("00") and we set an output signal called "clearCounter" to 0. Then using the case statement, we run different code based on which state is currently active. Each 'when' in the case statement will set the lights to the corresponding colours and if the counter is outputting a '1' to indicate that a time delay has passed then the next state will be set. When the time delay is up, the green state checks the value of the pedestrian value and changes the state based on its value. We ensured to set default values for all signals at the previous conditional statements to prevent any latches.

2 Block diagram of top-level structure.

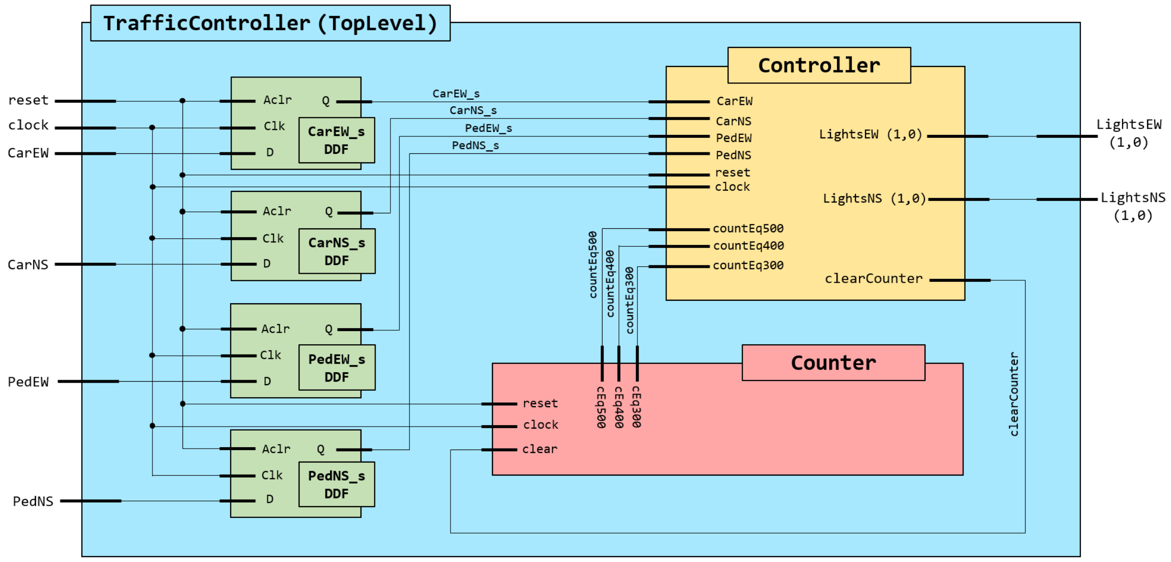

Figure 1 Block Diagram of top-level structure

3 State transition diagram (STD)

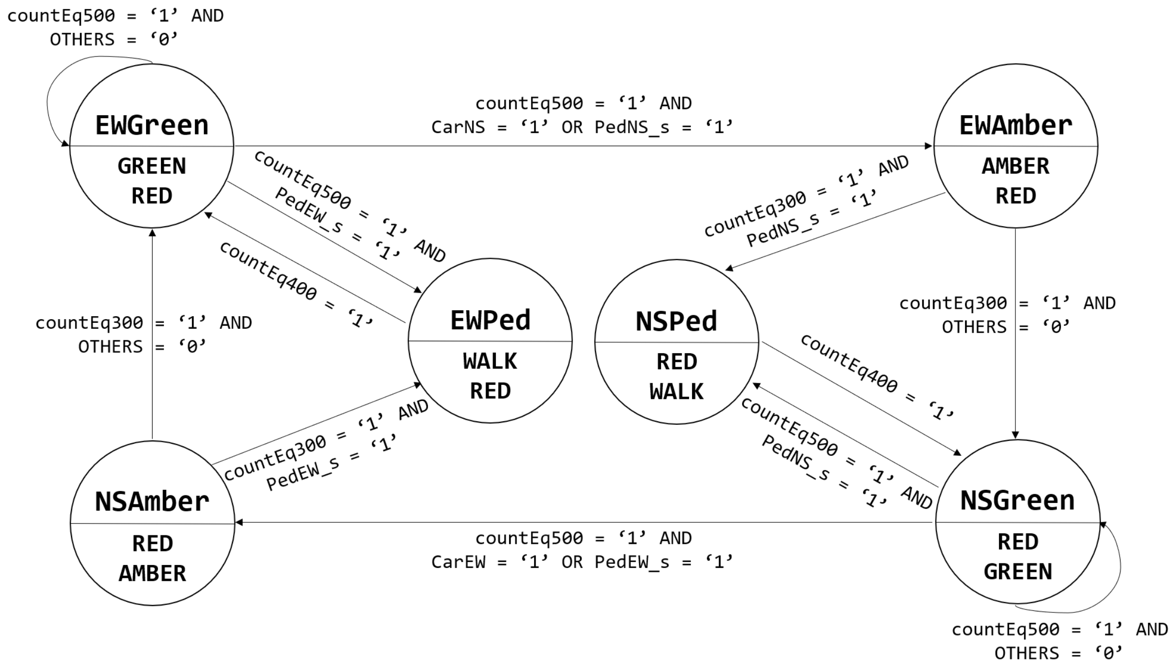

Figure 2 State Transition Diagram of the Traffic Controller

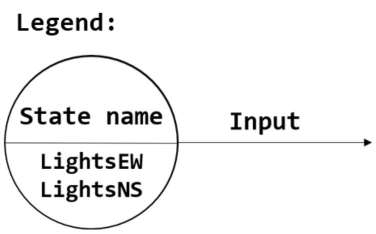

Figure 3 Legend for the State Transition Diagram

4 Testing Table

Current State	Input	Expected Output	Actual Output
EWGreen	CarEW is pressed and held (CarEW = ‘1’)	No change, stay in the EWGreen state	Same
EWGreen	CarNS is pressed and held (CarNS = ‘1’)	Lights cycle from EWGreen to EWAmber then to NSGreen	Same
EWGreen	PedNS is pressed and CarNS is pressed and held (PedNS_s = ‘1’ and CarNS = ‘1’)	Lights cycle from EWGreen to EWAmber then to NSPed stays there for (4s) before going to NSGreen only.	Same
EWGreen	PedEW is pressed (PedEW_s = ‘1’)	Lights cycle to EWPed for a short time (4s) before returning to EWGreen only	Same
EWGreen	PedNS is pressed (PedNS_s = ‘1’)	Lights cycle from EWGreen to EWAmber then to NSPed stays there for (4s) before going to NSGreen only.	Same
EWGreen	PedNS is pressed and CarEW is pressed and held (PedNS_s = ‘1’ and CarEW = ‘1’)	Lights cycle from EWGreen to EWAmber then to NSPed stays there for (4s) before going to NSGreen only.	Same
		If CarEW is held until after reaching NSGreen (up until NSAmber), then the lights will cycle back to NSAmber and then EWGreen	Same
		If CarEW is let go before that, the cycling ends (stays) at NSGreen only.	Same
EWGreen	PedEW and PedNS pressed at the same time	Lights cycle to EWAmber --> NSPed stays there for (4s) before going to NSGreen only (stays for 5s). Then cycles back to NSAmber --> EWPed stays there for 4s before returning to EWGreen only.	Same
Cycling from EWGreen to EWAmber then NSGreen	Reset is pressed (reset = ‘1’)	Terminates the cycling and resets the lights to EWGreen	Same
EWGreen	PedEW is pressed then after 3s PedNS is pressed	Lights cycle to EWPed (for 4s) then return to EWGreen (for 5s) then cycles to EWAmber (3s) then NSPed (4s) before returning to NSGreen only	Same
NSGreen	Nothing No input	Stays in NSGreen until an action button is pressed	Same

Lights just reached NSGreen	CarEW button is pressed and held (CarEW = '1')	Stays in NSGreen for the delay period (5s) then cycles to NSAmber (3s) and then to EWGreen	Same
EWGreen	All buttons are held for 15s	Lights cycle to EWAmber --> NSPed stays there for (4s) before going to NSGreen only (stays for 5s). Then cycles back to NSAmber --> EWPed stays there for 4s before returning to EWGreen only then cycles back again (since the PedNS_s = '1') to EWAmber, NSPed, NSGreen.	Same
EWGreen (for a long time)	PedEW button is pressed and held for 5s	Lights cycle to PedEW (stays 4s) then cycles back to EWGreen (stays 5s) then back to PedEW (stays 4s) then finished on EWGreen	Same
EWGreen	PedEW is pushed 5 times repeatedly	(Only the first one is counted) Lights cycle to PedEW (stays 4s) then cycles back to EWGreen	Same

5 VHDL modules

5.1 TopLevel Module (Traffic Controller Module)

```
-------------------------------------------------------------------
-- TrafficController.vhd
-- TopLevel of this project that includes the syncronising of inputs
-- and outputs from the other two modules into this one
-- i.e. from Controller and Counter into TrafficController
-------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- TopLevel module
entity TrafficController is
      Port(       -- Clock and Reset inputs
                  reset       :     in    STD_LOGIC;
                  clock       :     in    STD_LOGIC;

                  -- Car and pedestrian buttons
                  CarEW       :     in    STD_LOGIC;
                  CarNS       :     in    STD_LOGIC;
                  PedEW       :     in    STD_LOGIC;
                  PedNS       :     in    STD_LOGIC;

                  -- Light control
                  LightsEW    :     out   STD_LOGIC_VECTOR (1 downto 0);
                  LightsNS    :     out   STD_LOGIC_VECTOR (1 downto 0)
            );
end TrafficController;

architecture Behavioral of TrafficController is

-- Synchronized inputs
```

```vhdl
	signal CarEW_s		:	STD_LOGIC;
	signal CarNS_s		:	STD_LOGIC;
	signal PedEW_s		:	STD_LOGIC;
	signal PedNS_s		:	STD_LOGIC;


-- Internal connetions (coming from counter)
	signal clearCounter	:	STD_LOGIC;
	signal countEq500	:	STD_LOGIC;
	signal countEq400	:	STD_LOGIC;
	signal countEq300	:	STD_LOGIC;

begin
	--==========================================
	-- Synchronizes all inputs to sync. logic
	--==========================================
	synchronizer:
	process (reset, clock)
	begin
		if(reset = '1') then
			CarEW_s <= '0';
			CarNS_s <= '0';
			PedEW_s <= '0';
			PedNS_s <= '0';
		elsif rising_edge(clock) then
			CarEW_s <= CarEW;
			CarNS_s <= CarNS;
			PedEW_s <= PedEW;
			PedNS_s <= PedNS;
		end if;
	end process synchronizer;

	--==========================================
	-- Counter used for timing
	--==========================================
	--First level sync
	theCounter:
	entity work.Counter
		Port Map(
				reset	=> reset,
				clock	=> clock,

				clear	=> clearCounter, -- Clears the counter

				cEq300	=> countEq300, -- Count equals 400
				cEq400	=> countEq400, -- Count equals 400
				cEq500	=> countEq500 -- Count equals 500
			);

	--==========================================
	-- Contoller to implement state machine
	--==========================================
	--Second level sync
	theController:
	entity work.controller
	    Port Map(
				reset		=> reset,
				clock		=> clock,

				-- Car and pedestrian buttons
				CarEW		=> CarEW_s,
```

```
                        CarNS        => CarNS_s,
                        PedEW        => PedEW_s,
                        PedNS        => PedNS_s,

                        -- Light control
                        LightsEW     => LightsEW,
                        LightsNS     => LightsNS,

                        -- Counter control
                        clearCounter => clearCounter,
                        countEq300   => countEq300,
                        countEq400   => countEq400,
                        countEq500   => countEq500
                  );
end Behavioral;
```

5.2 Controller Module

```
-------------------------------------------------------------------
-- Controller.vhd
-- Includes the setting of the state machine based on the state
-- machine diagram
-- It includes two process:
-- SynchronousProcess:
--    to sync asynchronous inputs to the clock
--    including the pedestrians buttons
-- CombinationalProcess:
--          to format the state machine
-------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Controller module
entity Controller is
    Port ( -- Clock and Reset inputs
           reset         : in   STD_LOGIC;
           clock         : in   STD_LOGIC;

           -- Car and pedestrian buttons
           CarEW         : in   STD_LOGIC; -- Car on EW road
           CarNS         : in   STD_LOGIC; -- Car on NS road
                     -- Pedestrian moving EW (crossing NS road)
           PedEW         : in   STD_LOGIC;
                     -- Pedestrian moving NS (crossing EW road)
           PedNS         : in   STD_LOGIC;

           -- Light control
           -- controls EW lights
           LightsEW      : out  STD_LOGIC_VECTOR (1 downto 0);
           -- controls NS lights
           LightsNS      : out  STD_LOGIC_VECTOR (1 downto 0);

           -- Counter control
           clearCounter : out  STD_LOGIC; -- to clear counter
           -- to check for right timing
           -- count = 300, 400, 500 (will be synced from counter)
           countEq300    : in   STD_LOGIC; -- 3 seconds delay (Amber)
           countEq400    : in   STD_LOGIC; -- 4 seconds delay (Walk)
           countEq500    : in   STD_LOGIC  -- 5 seconds delay (Green)
```

```
           );
end Controller;

architecture Behavioral_CONT of Controller is

type  StateType is (NSGreen, NSAmber, EWGreen, EWAmber, NSPed, EWPed);
signal      state, nextState : StateType;
signal      PedNS_s, PedEW_s : STD_LOGIC;

-- Encoding for lights
constant RED   : std_logic_vector(1 downto 0) := "00";
constant AMBER : std_logic_vector(1 downto 0) := "01";
constant GREEN : std_logic_vector(1 downto 0) := "10";
constant WALK  : std_logic_vector(1 downto 0) := "11";

begin
      SynchronousProcess:
      process(reset, clock, PedNS_s, PedEW_s, PedNS, PedEW)
      begin
            if(reset = '1') then
                  state <= EWGreen; --default state
                  -- clearing PedNS_s and PedEW_s singals when reset is
                  -- pressed (clearing the flip-flops)
                  PedNS_s <= '0';
                  PedEW_s <= '0';
            elsif rising_edge(clock) then
                  state <= nextState;
                  if state = NSPed then
                  -- setting PedNS_s to 0 after reaching the NSPed state
                  -- (i.e. after turing the walk light for a short time)
                        PedNS_s <= '0';
                  elsif state = EWPed then
                  -- setting PedEW_s to 0 after reaching the NSPed state
                  -- (i.e. after turing the walk light for a short time)
                        PedEW_s <= '0';
                  end if;
                  --Ped buttons syncronising with rising_edge
                  -- Setting the PedNS_s and PedEW_s to 1, so that the
                  -- PedNS amd PedEW buttons don't need to be held
                  if(PedNS = '1') then
                        PedNS_s <= '1';
                  end if;
                  if(PedEW = '1') then
                        PedEW_s <= '1';
                  end if;
            end if;
      end process SynchronousProcess;

      CombinationalProcess: --state machince process
      process(state, countEq300, countEq400, countEq500, CarEW, CarNS,
      PedEW_s, PedNS_s)
      begin
            -- default values for outputs
            nextState <= state;
            LightsEW <= RED;
            LightsNS <= RED;
            clearCounter <= '0';
            case state is
                  when EWGreen  =>
                        -- setting the EW lights to green
                        LightsEW <= GREEN;
```

```
                -- Green lights delay (5s)
                if(countEq500 = '1') then
                -- waiting for the CarNS button to be pressed
                --(and held)
                        if(CarNS = '1') then
                                nextState <= EWAmber;
                                clearCounter <= '1';
                        -- PedNS_s is detected
                        -- cycle to EWAmber and then to NSPed
                        --(and NSGreen)
                        elsif (PedNS_s = '1') then
                                nextState <= EWAmber;
                                clearCounter <= '1';
                        -- PedEW_s is detected
                        -- cycle to EWPed (and stay in EWGreen)
                        elsif (PedEW_s = '1') then
                                nextState <= EWPed;
                                clearCounter <= '1';
                        end if;
                end if;

        when EWAmber  =>
                -- Setting the EW lights to amber
                LightsEW <= AMBER;
                -- Amber lights delay (3s)
                if(countEq300 = '1') then
                        -- PedNS button is pressed
                        -- before finishing cycling to green
                        if(PedNS_s = '1') then
                                nextState <= NSPed;
                                clearCounter <= '1';
                        -- PedNS button is not pressed
                        -- before finishing cycling to green
                        elsif(PedNS_s = '0') then
                                nextState <= NSGreen;
                                clearCounter <= '1';
                        end if;
                end if;

        when NSPed  =>
                -- moving to NSGreen regardless of input
                LightsNS <= WALK;
                -- Walk light delay (4s)
                if(countEq400 = '1') then
                        nextState <= NSGreen;
                        clearCounter <= '1';
                end if;

        when NSGreen  =>
                -- setting the NS lights to green
                LightsNS <= GREEN;
                -- Green lights delay (5s)
                if(countEq500 = '1') then
                -- waiting for the CarEW button to be pressed
                --(and held)
                        if(CarEW = '1') then
                                nextState <= NSAmber;
                                clearCounter <= '1';
                        -- PedEW_s is detected
                        -- cycle to NSAmber and then to EWPed
                        --(and EWGreen)
```

```
                                        elsif (PedEW_s = '1') then
                                                nextState <= NSAmber;
                                                clearCounter <= '1';
                                        -- PedNS_s is detected
                                        -- cycle to NSPed (and stay in NSGreen)
                                        elsif (PedNS_s = '1') then
                                                nextState <= NSPed;
                                                clearCounter <= '1';
                                        end if;
                                end if;

                        when NSAmber  =>
                                -- Setting NS lights to amber
                                LightsNS <= AMBER;
                                -- Amber lights delay (3s)
                                if(countEq300 = '1') then
                                    -- PedEW button is pressed
                                        -- before finishing cycling to green
                                        if(PedEW_s = '1') then
                                                nextState <= EWPed;
                                                clearCounter <= '1';
                                        -- PedEW button is not pressed
                                        -- before finishing cycling to green
                                        elsif(PedEW_s = '0') then
                                                nextState <= EWGreen;
                                                clearCounter <= '1';
                                        end if;
                                end if;
                        when EWPed  =>
                                -- moving to EWGreen regardless of input
                                LightsEW <= WALK;
                                -- Walk light delay (4s)
                                if(countEq400 = '1') then
                                        nextState <= EWGreen;
                                        clearCounter <= '1';
                                end if;
                        when others =>
                                -- To avoid any latches
                                nextState <= EWGreen;
                end case;
        end process CombinationalProcess;
end Behavioral_CONT;
```

5.3 Counter Module

```
-------------------------------------------------------------------
-- Counter.vhd
-- Here are the time delays defined based on the light used
-- This counter is based on 100Hz as the clock frequency
-- Based on this freqnecy, the time delay for:
-- Amber is 3s, Walk is 4s, Green is 5s
-------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Counter module
entity Counter is
        Port(
                        Reset      : in    STD_LOGIC;
```

```
                   clock      : in    STD_LOGIC;

                   clear      : in    STD_LOGIC; -- Clear counter to zero

                   cEq300     : out   STD_LOGIC; -- 3 seconds delay (Amber)
                   cEq400     : out   STD_LOGIC;   -- 4 seconds delay (Walk)
                   cEq500     : out   STD_LOGIC  -- 5 seconds delay (Green)
             );
end Counter;

architecture Behavioral of Counter is

signal count : natural range 0 to 502;
-- give the range of the count values as whole numbers
-- 502 to make sure that we do not miss 500
-- time in seconds = counts/100Hz
begin

      process (Reset, clock, count)
      begin
            if (Reset = '1') then
                  -- Reset count to 1
                  count <= 1;
            elsif rising_edge(clock) then
                  if (clear = '1') then
                        -- reset count to 1 only at rising edge
                        count <= 1;
                  else
                        -- count (increment count by 1)
                        count <= count + 1;
                  end if;
            end if;
            -- Amber light delay
            if (count = 300) then
                  cEq300 <= '1';
            else
                  cEq300 <= '0';
            end if;
            -- Walk light delay
            if (count = 400) then
                  cEq400 <= '1';
            else
                  cEq400 <= '0';
            end if;
            -- Green light delay
            if (count = 500) then
                  cEq500 <= '1';
            else
                  cEq500 <= '0';
            end if;
   end process;
end architecture Behavioral;
```