EEE20001 Digital Electronics Design
Semester 1, 2014

Traffic Intersection VHDL Project

Lab Supervisor:

Lab Session Day/Time: Thursday 4:30pm

Submitted by:

Kyle Van Berendonck___ 9989773
Name Student ID#

Finn Svendsen _____ 1721755
Name Student ID#

_____
Date

Design Summary (from ISE Summary report)

Number of Latches:	0	
Macrocells Used:	32/64	(50%)
Pterms Used:	73/224	(33%)
Registers Used	25/64	(40%)

Description of Approach

Our design consists of five modules:

- State Machine
- Timer
- North South Pedestrian Memory
- East West Pedestrian Memory
- North South Car Input Synchroniser
- East West Car Input Synchroniser

This implementation utilises four different types of VHDL entities:

- Traffic – the state machine
- Counter – the threshold timer
- DFF – The input synchronisers
- SRFF – The pedestrian synchronisation and memory

We utilize a hybrid Mealy-Moore state machine where;

- The Moore part of the state machine is responsible for controlling the state transitions and the outputs (traffic lights).
- The Mealy part of the state machine is responsible for clearing the timer between state transitions and resetting the pedestrian buttons.

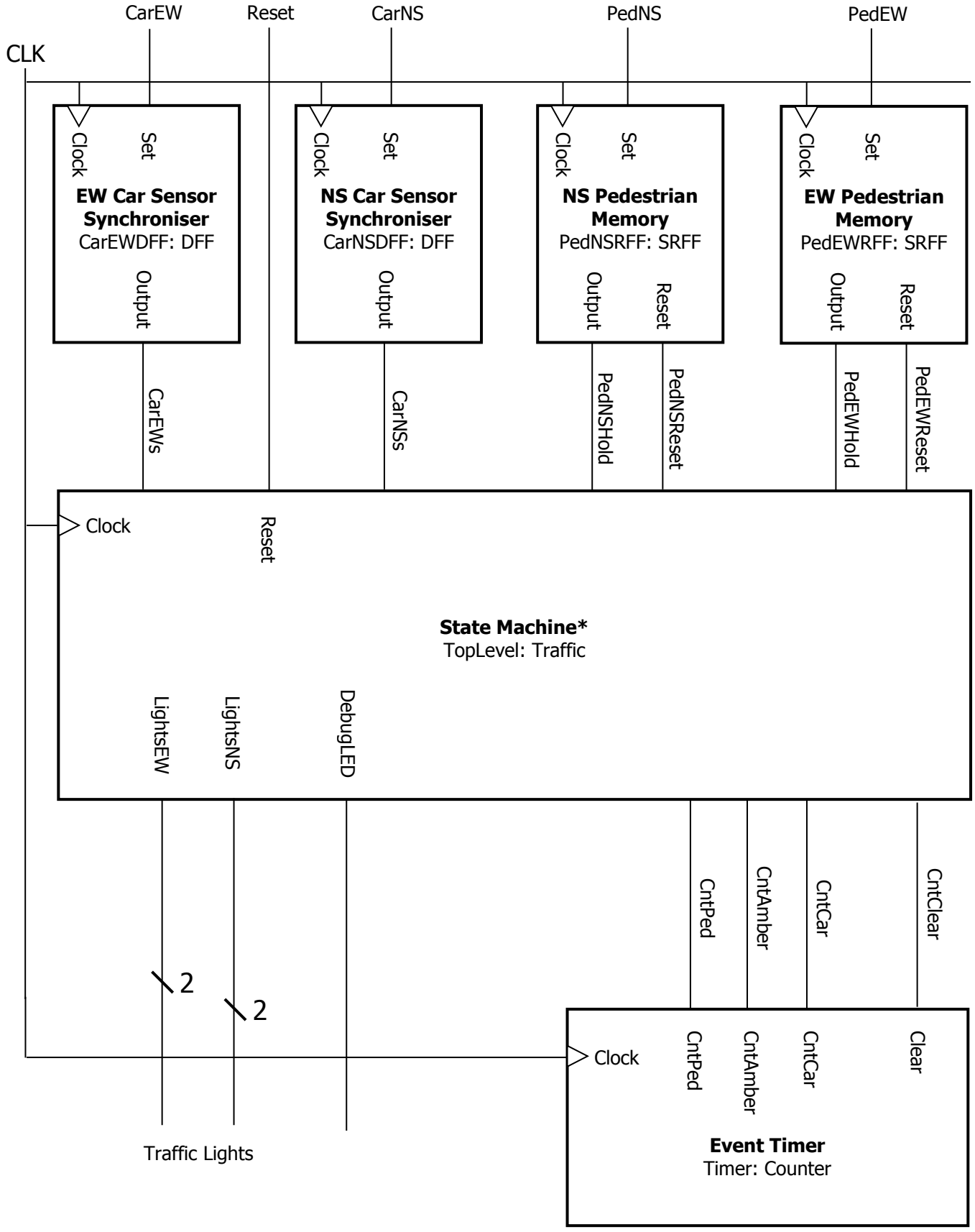
The state machine consists of six states and has:

- Synchronous Inputs: North South Pedestrian, North South Car Sensor, East West Pedestrian, East West Car Sensor, Pedestrian Timer Threshold, Amber Timer Threshold, Car Timer Threshold.
- Synchronous Outputs: North South Traffic Lights (Green, Amber, Red), North South Pedestrian Light, East West Traffic Lights (Green, Amber, Red), East West Pedestrian Light, Timer Clear.

The top-level module (Traffic) uses the recommended format; a combinatorial process on synchronous inputs for state machine logic, and a synchronous process with an asynchronous reset.

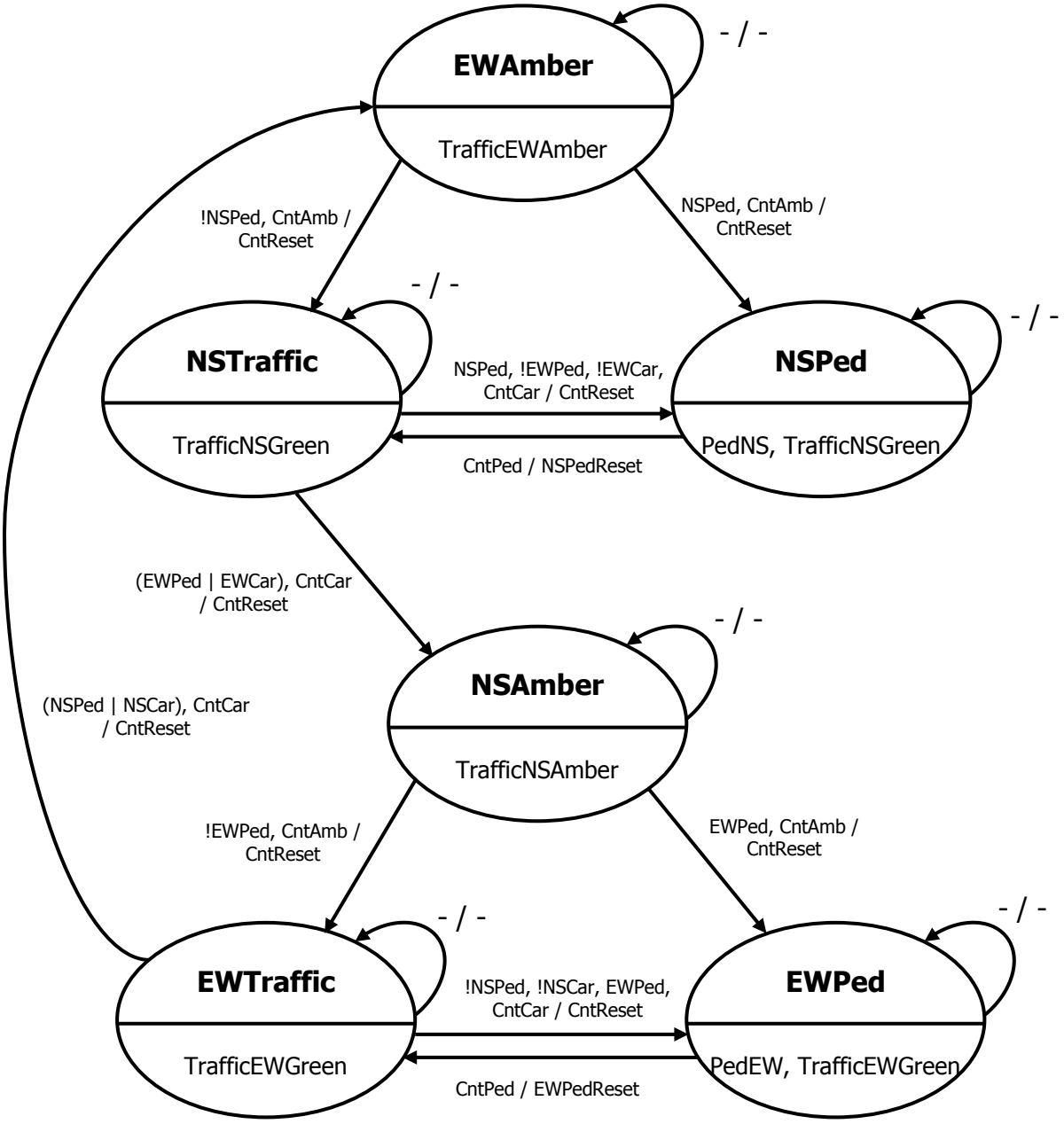
All inputs are synchronised such that they are protected from race conditions. As a design consideration, both the state and timer clear are also explicitly protected from combinatorial race conditions. Our implementation supports adjustable timer thresholds for pedestrian crossing, minimum traffic duration and amber duration.

Block Diagram of Top Level Structure



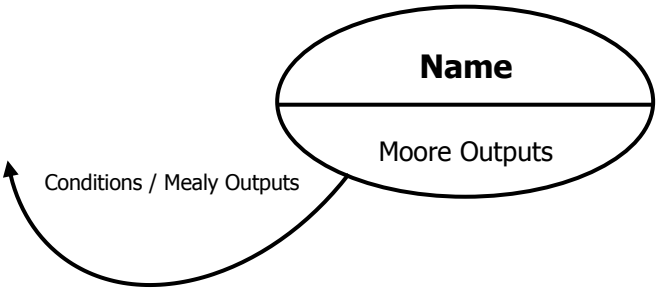
*To simplify the flow of the diagram, the State Machine is shown as a modular component (but without port names). In the VHDL implementation, the state machine is actually the top level module and the interconnected modules are instantiated within.

State Transition Diagram



Legend

Conditions:
NSPed (North South Pedestrian), NSCar (North South Car Sensor), EWPed (East West Pedestrian), EWCar (East West Car Sensor), CntPed (Timer Pedestrian Threshold exceeded), CntAmber (Timer Amber Threshold exceeded), CntCar (Timer Car Threshold exceeded)
Mealy Outputs:
NSPedReset, EWPedReset (Memory resets), CntReset (Timer reset)
Moore Outputs:
TrafficNSGreen, TrafficNSAmber, PedNS, TrafficEWGreen, TrafficEWAmber, PedEW (Light controls)



Testing

We produced testing procedures based on the dotpoints found in page 211 of the Unit Manual:

Test	Method	Assertions
Idle test Testing whether the circuit fulfils the requirement that it must not alter the traffic direction until an event in the perpendicular direction requires attention.	1. Power and reset the board 2. Wait 30 seconds	• The traffic should not change direction.
Parallel pedestrian test Testing whether pedestrians can cross the road in parallel to the traffic when the traffic is already going.	1. Power and reset the board 2. Wait 6 seconds 3. Press the east-west pedestrian button	• After a short delay, the east-west pedestrian walk lights should enable for a small time period and then disable. The east-west traffic should not be interrupted.
Direction switch test Testing whether the lights will change direction given the correct input.	1. Power and reset the board 2. Immediately press the north south pedestrian buttons 3. Wait a short while	• After a medium delay, the east-west traffic lights should cycle through amber to red. When this is complete, the north-south lights (including the walk light) should be green.
Priority test Testing whether the intersection can be stalled by a malicious pedestrian.	1. Power and reset the board 2. For 6 seconds, repeatedly press the east-west pedestrian button 3. Press the north-south pedestrian button 4. Immediately continue to press the east-west pedestrian button	• The traffic direction should cycle to north-south shortly after pressing the north-south pedestrian button despite also pressing the east-west button. A short time after the traffic swaps to north-south, it should swap back to east-west in reaction to the pedestrian present.
Amber stall test A regression test for an issue we found in the state machine which caused it to erroneously stall.	1. Power and reset the board 2. Wait 6 seconds 3. For an extremely briefly period of time, press the north-south car button and then let go	• The east-west light should do a full cycle from green to amber and finally stopping at red. Observe carefully for a regression where the lights would become frozen in amber state until more input is received from either intersection. The north south lights should eventually turn green.

VHDL

```
--------------------------------------------------------------------------------
-- Traffic.vhd
--
-- Toplevel module for the traffic light/intersection controller.
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity Traffic is port
   ( Reset              : in  std_logic
   ; Clock              : in  std_logic

   -- CPLD board LED
   ; DebugLED           : out std_logic

   -- Car/Ped sensors
   ; 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 Traffic;

architecture Behavioral of Traffic is
   -- 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";

   -- State machine
   type StateType       is ( EWAmber, EWTraffic, EWPed, NSAmber, NSTraffic, NSPed );
   signal State         : StateType;
   signal NextState     : StateType;

   -- Counter flags
   signal CntPed        : std_logic;
   signal CntAmber      : std_logic;
   signal CntCar        : std_logic;
   signal CntClear      : std_logic;
   signal CntClearNext  : std_logic;

   -- NS synchronous inputs
   signal CarNSs        : std_logic;
   signal PedNSHold     : std_logic;
```

```
   signal PedNSReset    : std_logic;

   -- EW synchronous inputs
   signal CarEWs        : std_logic;
   signal PedEWHold     : std_logic;
   signal PedEWReset    : std_logic;
begin
   DebugLED <= Reset; -- Show reset status on FPGA LED

   Timer : entity Counter port map
      ( Clear  => CntClear
      , Clock  => Clock
      , CntPed => CntPed
      , CntAmb => CntAmber
      , CntCar => CntCar );

   CarEWDFF : entity DFF port map
      ( Set    => CarEW
      , Clock  => Clock
      , Output => CarEWs );

   CarNSDFF : entity DFF port map
      ( Set    => CarNS
      , Clock  => Clock
      , Output => CarNSs );

   PedNSSRFF : entity SRFF port map
      ( Set    => PedNS
      , Reset  => PedNSReset
      , Clock  => Clock
      , Output => PedNSHold );

   PedEWSRFF : entity SRFF port map
      ( Set    => PedEW
      , Reset  => PedEWReset
      , Clock  => Clock
      , Output => PedEWHold );

   SynchronousProcess:
   process (Reset, Clock)
   begin
      if (Reset = '1') then
         State <= EWTraffic;
         CntClear <= '1';
      elsif (rising_edge(Clock)) then
         State <= NextState;
         CntClear <= CntClearNext;
      end if;
   end process SynchronousProcess;

   CombinatorialProcess:
```

```
process (State, CarEWs, CarNSs, PedEWHold, PedNSHold, CntAmber, CntCar, CntPed)
begin
   LightsEW <= RED;
   LightsNS <= RED;
   NextState <= State;
   CntClearNext <= '0';
   PedNSReset <= '0';
   PedEWReset <= '0';

   case State is
      when EWAmber =>
         LightsEW <= AMBER;
         if (CntAmber = '1') then
            if (PedNSHold = '1') then
               NextState <= NSPed;
            else
               NextState <= NSTraffic;
            end if;
         end if;

      when EWTraffic =>
         LightsEW <= GREEN;
         if (CntCar = '1') then
            if (PedNSHold = '1' or CarNSs = '1') then
               NextState <= EWAmber;
               CntClearNext <= '1';
            elsif (PedEWHold = '1') then
               NextState <= EWPed;
               CntClearNext <= '1';
            end if;
         end if;

      when EWPed =>
         LightsEW <= WALK;
         if (CntPed = '1') then
            PedEWReset <= '1';
            NextState <= EWTraffic;
         end if;

      when NSAmber =>
         LightsNS <= AMBER;
         if (CntAmber = '1') then
            if (PedEWHold = '1') then
               NextState <= EWPed;
            else
               NextState <= EWTraffic;
            end if;
         end if;

      when NSTraffic =>
         LightsNS <= GREEN;
```

```
            if (CntCar = '1') then
               if (PedEWHold = '1' or CarEWs = '1') then
                  NextState <= NSAmber;
                  CntClearNext <= '1';
               elsif (PedNSHold = '1') then
                  NextState <= NSPed;
                  CntClearNext <= '1';
               end if;
            end if;

         when NSPed =>
            LightsNS <= WALK;
            if (CntPed = '1') then
               PedNSReset <= '1';
               NextState <= NSTraffic;
            end if;
      end case;
   end process CombinatorialProcess;
end Behavioral;


----------------------------------------------------------------------------------
-- Counter.vhd
--
-- An event timer based on a counter with thresholds.
----------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity Counter is port
   ( Clear              : in  std_logic
   ; Clock              : in  std_logic

   -- Count threshold outputs
   ; CntPed             : out std_logic
   ; CntAmb             : out std_logic
   ; CntCar             : out   std_logic );
end Counter;

architecture Behavioral of Counter is
   signal Count         : natural range 0 to 10000;
begin
   Counter:
   process (Clear, Clock)
   begin
      if (Clear = '1') then
         Count <= 0;
      elsif (rising_edge(Clock)) then
         Count <= Count + 1;
      end if;
   end process;
```

```
   Flags:
   process (Count)
   begin
      CntPed <= '0';
      CntAmb <= '0';
      CntCar <= '0';
      -- Pedestrian time threshold
      if (Count > 450) then
         CntPed <= '1';
      end if;
      -- Amber time threshold
      if (Count > 150) then
         CntAmb <= '1';
      end if;
      -- Traffic time threshold
      if (Count > 600) then
         CntCar <= '1';
      end if;
   end process;
end Behavioral;


--------------------------------------------------------------------------------
-- DFF.vhd
--
-- D Flip-Flop implementation.
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity DFF is port
   ( Clock              : in  std_logic
   ; Set                : in  std_logic
   ; Output             : out std_logic );
end DFF;

architecture Behavioral of DFF is
   signal Q             : std_logic;
begin
   Logic:
   process (Clock)
   begin
      Q <= Q;
      if rising_edge(Clock) then
         Q <= Set;
      end if;
      Output <= Q;
   end process;
end Behavioral;
```

```
--------------------------------------------------------------------------------
-- SRFF.vhd
--
-- SR Flip-Flop implementation.
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity SRFF is port
   ( Set                : in  std_logic
   ; Reset              : in  std_logic
   ; Clock              : in  std_logic
   ; Output             : out std_logic );
end SRFF;

architecture Behavioral of SRFF is
   signal Q             : std_logic;
begin
   Logic:
   process (Clock, Reset)
   begin
      if Reset = '1' then
         Q <= '0';
      elsif rising_edge(Clock) then
         if Set = '1' then
            Q <= '1';
         end if;
      end if;
      Output <= Q;
   end process;
end Behavioral;
```