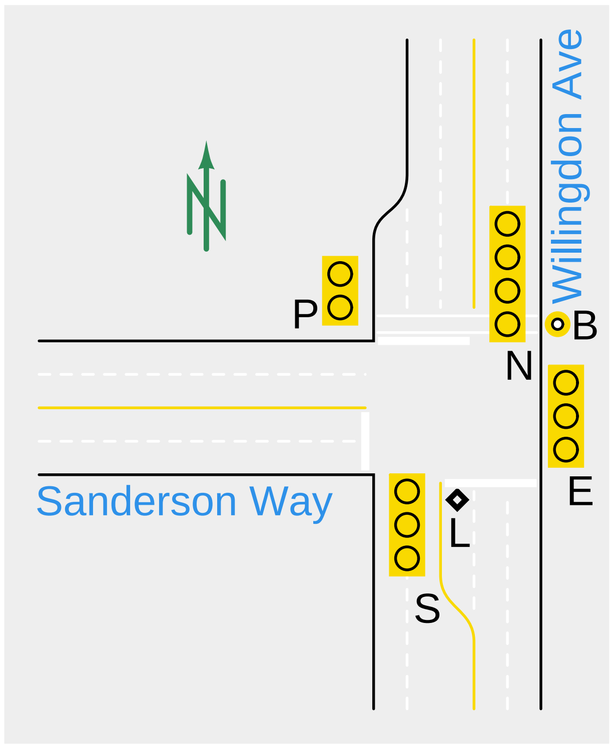
Traffic Light Controller

Introduction

In this lab you will design a traffic light controller for the intersection at Willingdon Avenue and Sanderson Way. This intersection has a left-turn lane and pedestrian crossings.

Specifications

The diagram below shows the layout of the intersection. We will only include the westbound pedestrian crossing across Willingdon.



Inputs and Outputs

The four signals are labelled for the traffic flow they control: N(orth), S(outh), E(ast) and P(edestrian). Each signal has two, three or four lamps labelled by colour or direction (R(ed), Y(ellow), G(reen) and L(eft)). There are two sensors: an induction loop (L) in the road that detects cars waiting to turn left northbound on Willingdon and a pushbutton (B) that pedestrians push to indicate they want to cross the road.

Your design will control the lamps through a 12-bit output declared `std_logic_vector (11 downto 0)` with active-high signals ordered (left-to-right): NR NY NG NL ER EY EG SR SY SG PR PG .

Your design will sense the button and loop through a 2-bit input declared `std_logic_vector (1 downto 0)`: with active-high signals ordered (left-to-right): B L. The loop signal is active when there is a car stopped above the loop. The button signal is active only when the button is being pushed (your design will have to "remember" if it was pushed).

Operation

To avoid collisions only some combinations of lights are allowed. The combinations and the lit lamps are:

- NB - allows northbound traffic to go left or north: NG, NL, ER, SR, PR.
- NSB - allows north/southbound traffic: NG, ER, SG, PR.
- EB - allows eastbound traffic: NR, EG, SR, PG.

Signals must transition from green to yellow before turning red to allow cars time to stop. Thus there are three more combinations where Y is substituted for G (and NL is off and PR is on).

The intersection normally transitions from EB to NSB and back with yellow signals in-between. However, when a northbound car is waiting to turn left the intersection will transition from EB to NB and then to NSB.

The NSB condition ends after 6 seconds. NB ends after 3 seconds. EB ends after 6 seconds but if B was pushed then it should end after 10 seconds (this allows pedestrians enough time to cross the road). Yellow signals end after 1 second.

Pre-Lab

State Machine Design

Determine the number of states required to implement the intersection controller and the values of the output signals for each state. As per course guidelines, all outputs must be the outputs of registers (be "registered"). Refer to the state machine design instructions in the lecture notes if necessary.

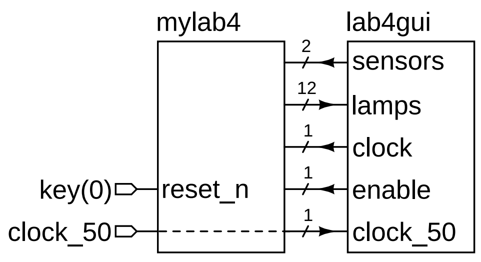
Assign names to the states and prepare a table showing showing the `lamps` output values for each state.

Draw a state transition diagram labelled with state names. The state transitions should be labelled with the conditions that must be met to cause the transition. These could be combinations of: active inputs, other state machine states or time elapsed time in the current state.

You may use another state machine to capture button pushes. This state machine may have an output (e.g. W(aiting)) that may be used as state transition condition in the above state machine.

VHDL Implementation

Your design will interface to a virtual control panel running on the PC by instantiating a component supplied by the instructor:



where `mylab4` is your top-level entity and the `lab4gui` component is declared as follows:

```
-- interface to the virtual hardware and clocks
component lab4gui is
   port (
      sensors : out std_logic_vector(1 downto 0);
      lamps : in std_logic_vector (11 downto 0);
      clock : out std_logic ; -- 10 kHz clock
      enable : out std_logic ; -- 1 pulse-per-second
      clock_50 : in std_logic -- 50 MHz clock
   );
end component ;
```

This component is declared in a package called `lab4pkg`. To use the component it is sufficient to put:

```
use work.lab4pkg.all ;
```

before your entity declaration.

The component has a 50 MHz clock input that must be supplied from the `clock_50` input to your top-level entity. You *must* use the component's 10 kHz `clock` output as your clock signal. You *may* use the `enable` signal, which is high for one clock cycle every second, to simplify your design.

Your design should use the `key(0)` input as an active-low reset input, `reset_n` that forces the controller to EB.

Submit a report in PDF format to the appropriate dropbox on the course web site containing the following: the usual identification information, the state ouputs table, the state transition diagram described above, and VHDL code that implements your design.

Procedure

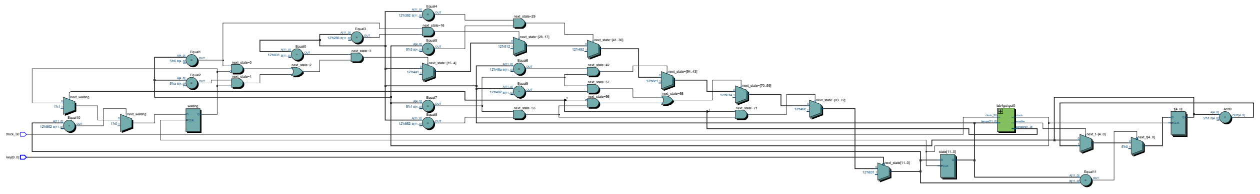
Follow the instructions in Lab 1 but add the following step before compiling your code:

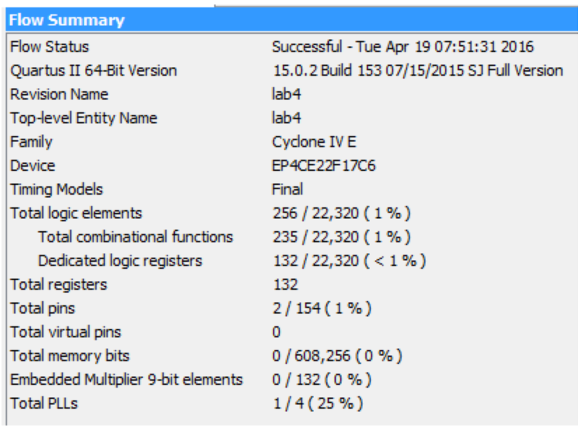
- download `lab4.zip` from the course web site and extract the contents into your project folder. Then select Project > Add/Remove Files in Project... and add the file `lab4gui.vhd` to your project.

After the FPGA is programmed, double-click on the `lab4gui .bat` file to start the user interface and test your design. Check: (1) normal cycling through the states, (2) that the green light duration is extended when you push the 'walk' button, and (3) that the "advanced green" state is used when cars are waiting to turn left northbound on Willingdon.

Lab Report

Use the report instructions in the previous lab. Sample flow summaries and schematics are shown below.





Flow Summary	
Flow Status	Successful - Tue Apr 19 07:51:31 2016
Quartus II 64-Bit Version	15.0.2 Build 153 07/15/2015 SJ Full Version
Revision Name	lab4
Top-level Entity Name	lab4
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	256 / 22,320 (1 %)
Total combinational functions	235 / 22,320 (1 %)
Dedicated logic registers	132 / 22,320 (< 1 %)
Total registers	132
Total pins	2 / 154 (1 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	1 / 4 (25 %)

Hints and FAQs

Q: What types should I use for my signals?

A: If it's something you count (like seconds), use `unsigned`. If it's something that's true or false (like sensor and lamp values), use `std_logic[_vector]`. If it's something that can be one of several choices, use an enumerated type. Don't use `bit[_vector]` any more. For example:

```
signal t, next_t : unsigned (4 downto 0) ;
signal waiting, next_waiting : std_logic ;
...
subtype states is std_logic_vector (11 downto 0) ;
signal state, next_state : states ;
...
constant EBY : states := "100001010010" ;
...
alias L is sensors(0);
alias reset_n is key(0);
```

Note that in this case I've used `std_logic_vector` rather than an enumerated type for the states because the state values are also the output values (a rule in this course). Aliases and the 'subtype' are a convenient way to specify shorter or more meaningful names for other signals.

Q: Give me a hint, please.

A: A simple solution consists of one component instantiation (`lab4gui`) and three state machines:

- one to keep track of whether the pedestrian button has been pushed
- one to keep track of how many seconds have elapsed in the current state
- one to select the next state (which is the same as the lamp output value) based on: (i) the current state, (ii) the elapsed time in the current state, (iii) the current loop sensor input and (iv) whether a pedestrian has pushed the button

Each state machine requires one conditional assignment statement and one unconditional assignment inside a process statement. See below.

Q: How do I "remember" that the pedestrian button has been pushed?

A: Use a flip-flop that is set when the button is pushed and reset when leaving the pedestrian crossing state. For example:

```
-- state update:
 next_waiting <=
   '0' when next_state = EBY else
   '1' when B = '1' else
   waiting ;
...
-- in your process:
     waiting <= next_waiting ;
```

Q: How do I keep track of how much time has elapsed in a state?

A: Use a register that is set to zero when the state changes and is incremented once per second (e.g. when a 1 pps[1] enable is active) otherwise. For example:

```
-- state update:
```

[1]pulse per second

```
next_t <=
  to_unsigned(0,t'length) when next_state /= state else
  t + 1 when enable = '1' else
  t ;
 ...
-- in your process:
     t <= next_t ;
```

Q: How can I code the state transition rules?

A: You could use a conditional assignment such as:

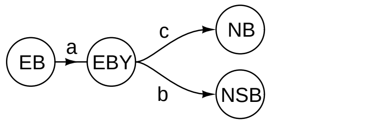
```
  next_state <=
    NB   when state = EBY  and t = 1 and L = '1' else
    ... else
    state ;
```

Q: What signals do I need to include in my entity declaration's `port` clause? What signal names should I use?

A: For this lab, the 50 MHz signal from the oscillator on the DE0-Nano board and the `key(0)` button. Connect the `clock_50` signal directly to the `lab4gui`'s `clock_50` input and use `key(0)` as an active-low reset.
You must use the same signal names (`clock_50`, `key`) that are used in the settings file (`DE0_NanoX3330.qsf`). Using the wrong signal name or not importing the settings file will result in dimly-lit LEDs and nothing else.

Q: Can you give me an example of a state transition diagram?

A: Different styles may be appropriate for different readers. In the following (partial) example the state names and transition conditions are taken from the VHDL code:



a: (W='1' and T=10) or (W='0' and T=6)
b: T=1 and L='0'
c: T=1 and L='1'

Using descriptive state names ("Northbound Traffic") and transition conditions ("walk button not pushed and 10s timeout") is also acceptable. You do not have to show that transitions happen on clock edges (always the case in this course) or that the default transition is remain in the same state (this can be assumed).

Q: Should I use the code above into my solution?

A: These are just examples meant to help you figure out your own solution.

Q: Why are none of the lights on? All of my states have at least one light turned on for each signal!

A: Did you reset your circuit? If not, the state register powers up with value 0 resulting in all of the lamps being off.

Q: Why does the FPGA programmer say 'Failed'? It worked the first time!

A: The JTAG interface used to program the FPGA is also used by the GUI software. You must quit the lab4gui program before (re-)programming the FPGA.

Q: How could the design be improved?

A: The RTL netlist shows many 5-bit comparators to test the elapsed time and 12-bit comparators to test the current state. The design might be improved by using a down-counter that is initialized on entry to the state and a different state encoding (for example, you could use 'one-hot' state variables that do not appear in the output).

Q: What should I do before the lab to maximize my chances of completing it?

A: You should: (a) ask about anything in the Lab instructions that is not clear, (b) complete and submit the Pre-Lab report and (c) have VHDL code that compiles without syntax errors.