Assignment 4

Traffic Light

1 Goal

After gaining experience in coding a given statechart with QP in the previous assignment, students will have a chance to *design* (*refine*) a statechart on their own from given requirements.

2 Readings

a) **Systems of Systems Modeled by a Hierarchical Part-Whole State-Based Formalism**. Luca Pazzi. June 2013.

   (https://www.researchgate.net/publication/236156084_Systems_of_Systems_Modeled_by_a_Hierarchical_Part-Whole_State-Based_Formalism)

   This paper discusses how to partition a system into layers of HSMs. At each layer, an HSM is a ***part*** serving the HSM at the layer above it, while at the same time acts as a ***whole*** representing all the lower-layer HSMs under its control.

   Fig. 3 shows the traffic light system similar to that we use in this assignment. The controller (bottom part – Note it's upside down) is a part to the bigger system while it is a whole representing the two lights it controls.

   Fig. 2 shows the counter-example without using this hierarchical control pattern in which two traffic lights interacting directly to each other without an explicit controller.

b) **From Protocol Specification to Statechart to Implementation**. Kiri L. Wagstaff, Ken Peters, and Lucas Scharenbroich. Jet Propulsion Laboratory. October 2008.

   (http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=81E437D3F3CEAC9F0D1B7FD187AC709D?doi=10.1.1.207.5154&rep=rep1&type=pdf)

   Like (a) above this paper uses a traffic light system to illustrate statecharts and orthogonal regions. The Traffic component in our assignment project is similar to Figure 2 of this paper.

3 Setup

1. Download the compressed project file (platform-stm32l475-disco-assignment4.zip).

2. Backup your existing project folder. Decompress the downloaded project file to a local folder.
3. Launch STM32CubeIDE. Hit F5 to refresh the project content.

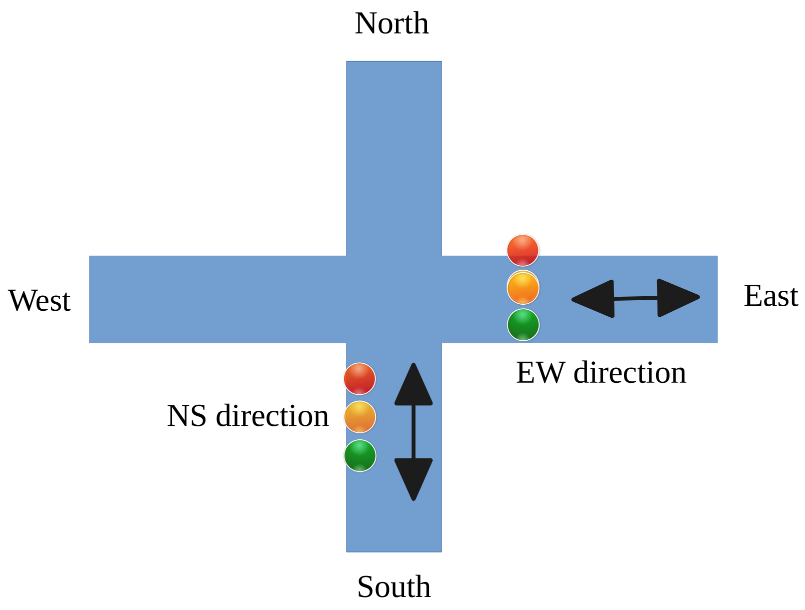
   Or you can right-click on the project in Project Explorer and click "Refresh".
4. Clean and rebuild the project. Download it to the board and make sure it runs.

   In the UART console, type the command "traffic n" or "traffic e". You should see log messages printed out on the console. (Note – the LCD just shows a blank white screen.)
5. If you have not installed UMLet, you will need to do so for this assignment. Please refer to the Setup notes in the previous assignment.

4 Tasks

The design used in this assignment is for a traffic light system at an intersection between North-South (NS) and East-West (EW) roads.

There are two sets of traffic lights, one along the NS direction and the other along the EW direction. They are illustrated in the diagram below. Note that there are two opposite facing traffic lights in each set, which show the same pattern (both RED or both GREEN, etc). For brevity, only one in each set (south facing and east facing) is shown below. Also for simplicity there are no left-turn signals.



1. In the assignment project, the traffic light system is located in the folder **Src/app/Traffic**. It contains the *Traffic* active object (HSM) and the *Lamp* orthogonal regions (under **Src/app/Traffic/Lamp**”). We can think orthogonal regions as parallel HSMs running in the same thread. We will learn more about them in the next class.

The event interface for the Traffic component is defined in TrafficInterface.h. The event TRAFFIC_CAR_NS_REQ is a request from an external component (e.g. smart camera sensor) to request passage along the NS direction. The event is generated each time when a car is detected in front of the north or south facing traffic light. The similar concept applies to the EW direction. For simplicity, we can assume that the camera sensor is smart enough to generate only one event for each specific car it detects.

For simulation, the console command '**traffic n**' or '**traffic s**' generates a TRAFFIC_CAR_NS_REQ event, and '**traffic e**' or '**traffic w**' generates a TRAFFIC_CAR_EW_REQ event.

2. The reference design is described in the statechart named Traffic.uxf. Your are tasked to improve it according to the following new requirements:

   (a) Imposes a minimum duration between each change of direction to avoid the "go" direction from toggling too fast. Note that the NS direction is hypothetically busier and its minimum "go" duration of 20s is intentionally longer than that of the EW direction of 10s. (That is, after the NS direction has switched to a green light, it will keep showing the green light for at least 20s even a car has arrived at the EW direction within that period.)

   (b) Since the NS direction is the main route, we would like to have the "go" direction automatically return to the NS direction when *no cars have been detected along the EW direction for 15 seconds*. Note: You can use the event TRAFFIC_CAR_EW_REQ to check if a car has passed through the intersection along the EW direction.

   (c) Do you see another problem with the original design? What if a TRAFFIC_CAR_EW_REQ event arrives in the EWSlow state? That is a EW-going car is detected while the yellow light is shown along the EW direction. (Assume the driver behaves well and stops in front of the light rather than rushing through it.) Try to fix this issue to avoid the car from being stuck in front of the light forever.

3. Show your design changes in the Traffic statechart. Update the Traffic.uxf file with UMLet and export the finished design to a **pdf** file for submission.

4. Test your code with the console command "traffic ..." and ensure the expected log messages are observed.