Assignment 1 – traffic lights

ASE-9426 Distributed Automation Systems Design

Juha Suntila, 256667
Topi Huttunen, 252791
Matti Savolainen, 274172

Introduction

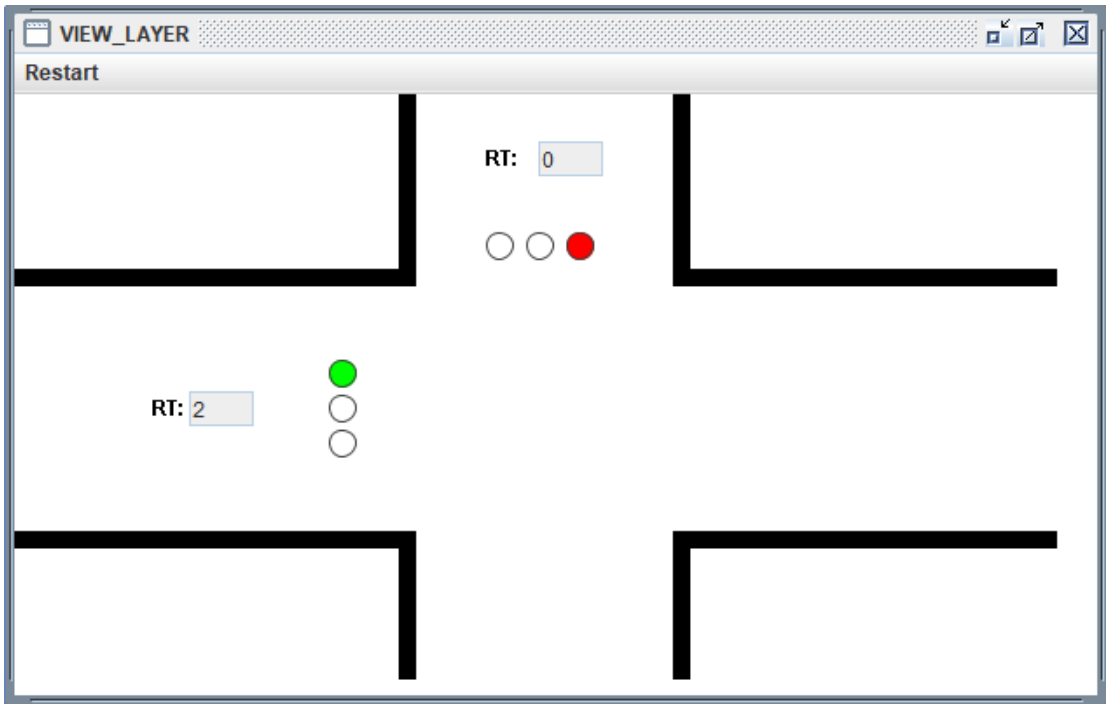
In the assignment students were meant to build a traffic lights control system for two one-way crossing streets, with a pedestrian crossing on both. Objectives for the assignment were to learn build distributed systems based on IEC 61499, design systems at higher level, apply MVC design pattern and learn event-driven architectures. The traffic lights were meant to be build in centralized and distributed ways. A program called Function Block Development Kit (fbdk) was used to build the systems.

Requirements for the system were to show status of each semaphore, the remaining time green or yellow light was to be on, and when pedestrian button were pressed four times, and at least 4 seconds had passed from the corresponding green lights being turned on, the lights would turn to yellow.

Due to misunderstanding of the requirements, the system does fill all the requirements listed above, but it has certain additional features. The additional features include possibility to adjust the times of green and yellow lights on the run for next cycle and pedestrian button lowering the remaining time to a set "minimum green time" if there are more time left than the set value is. In the following chapters the system and how it works is presented piece by piece.

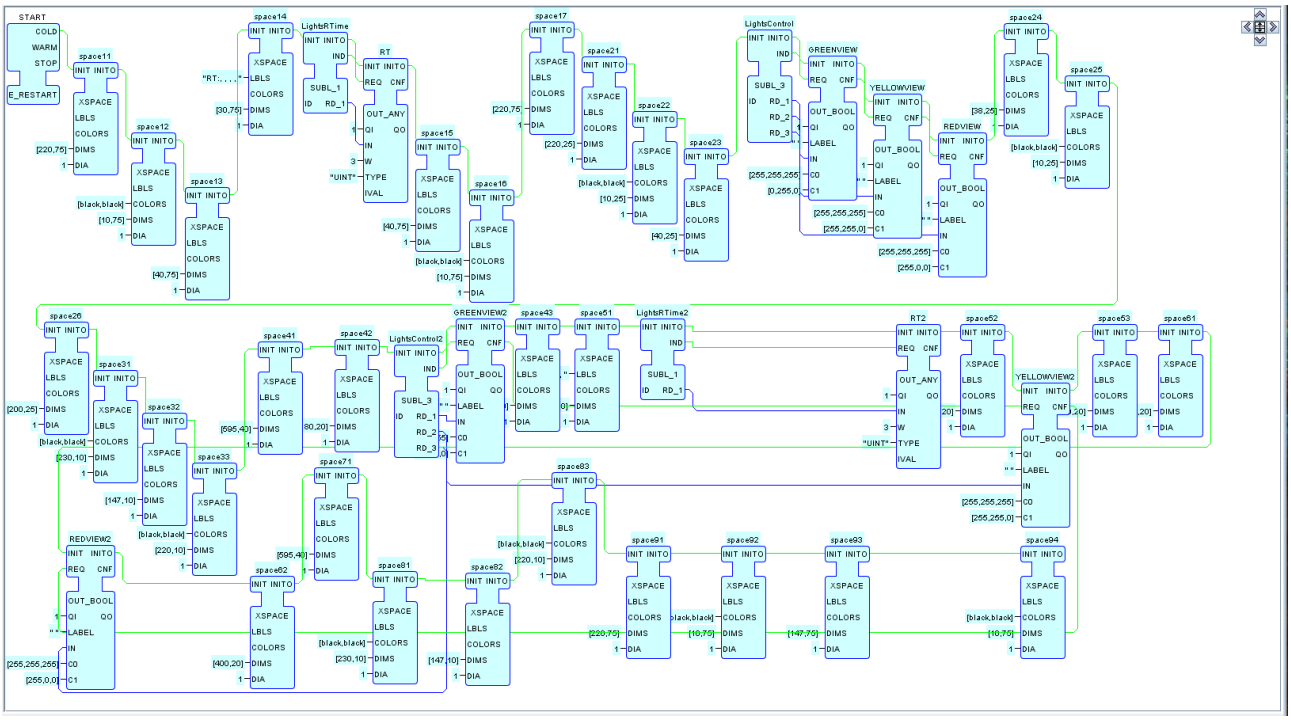
VIEW LAYER

The designed view layer is presented in the picture 1 below. The view layer was build based on the picture given on the presentation of the assignment.



Picture 1: View layer

The code for this is presented in picture 2.

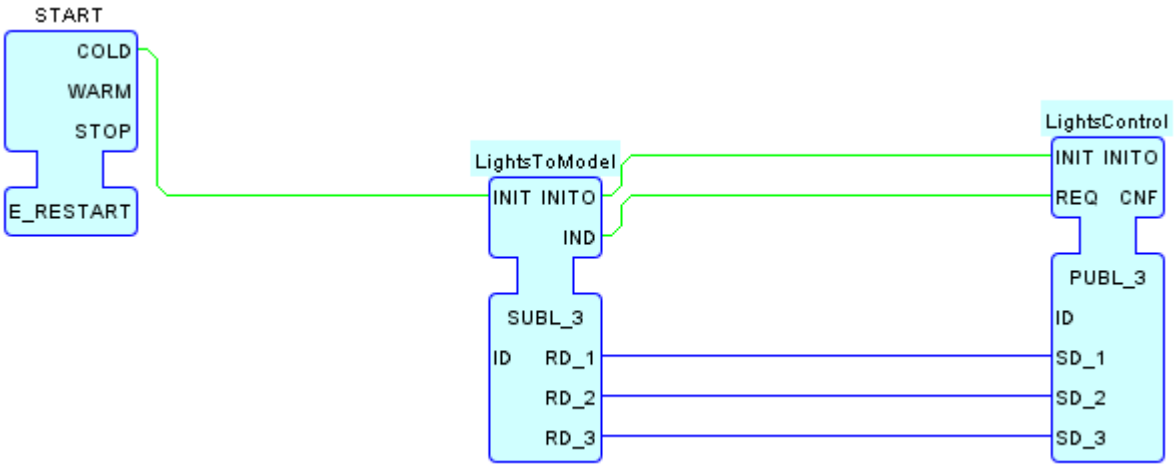


Picture 2: View layer code

Basically, it is made of the xspace -blocks to fill out the white and black areas of the picture, forming the streets, OUT_ANY -blocks for the remaining times, OUT_BOOL -blocks for the lights and subscribers to the required variables and events. The texts are a part of the xspace -blocks.

MODEL

The model for each traffic light included one subscriber and one publisher block shown in picture 3.

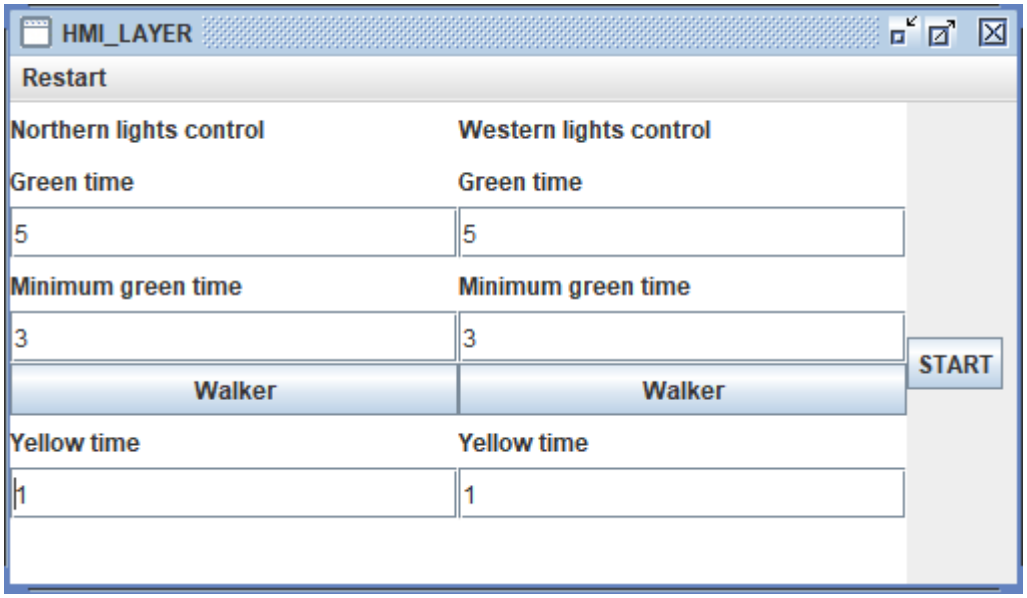


Picture 3

Nothing fancy in here due to the way the traffic lights are.

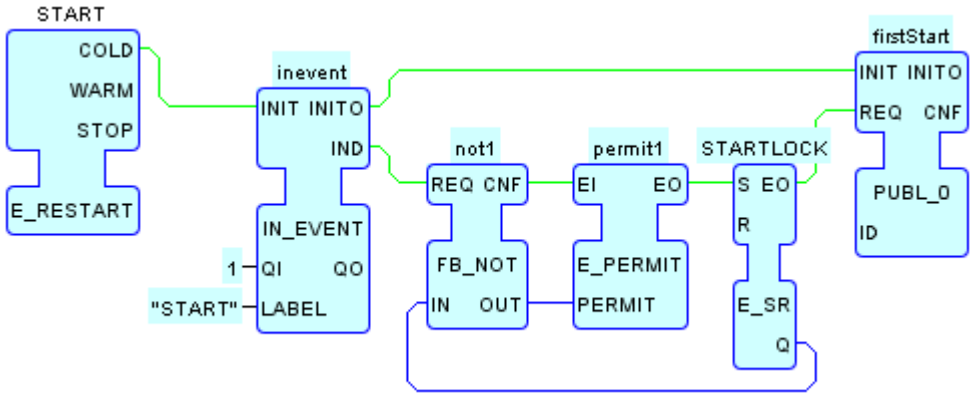
HMI LAYER

HMI layer, same as view layer, were both used as they were in both distributed and centralized approaches. HMI layer included three resources, start, HMI_NORTH and HMI_WEST.



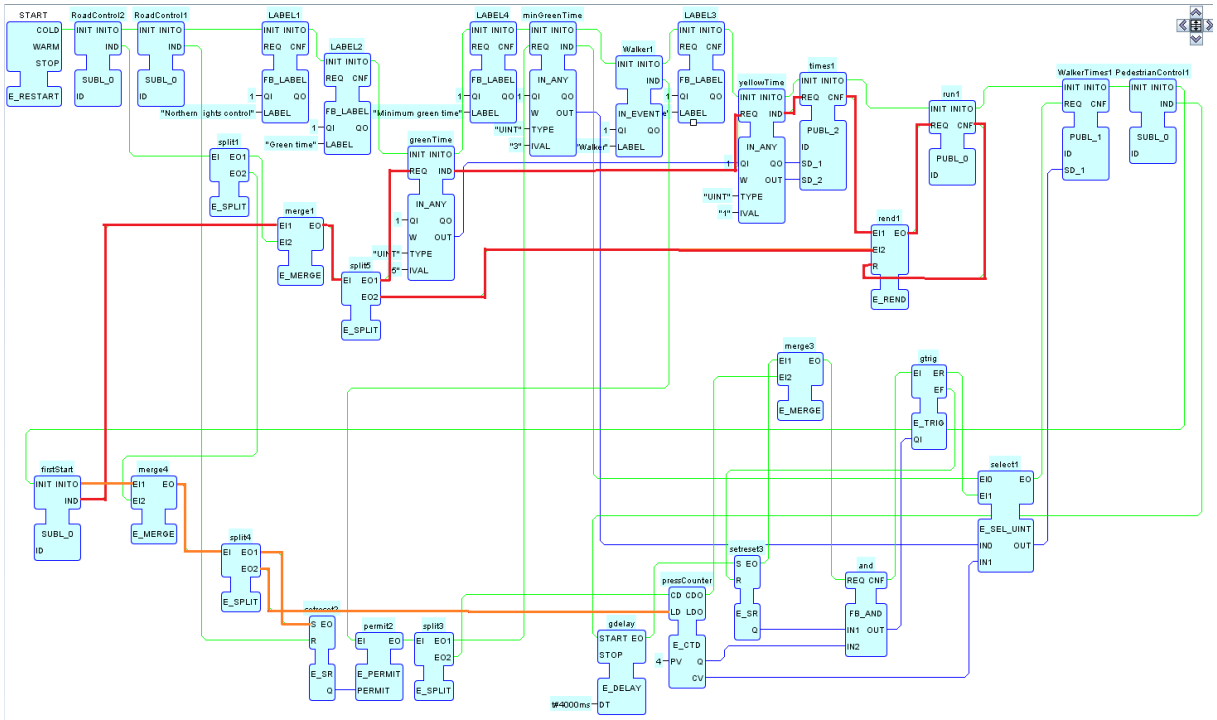
Picture 4: HMI layer

Picture 4 shows the HMI for the user. The user can adjust the times for green and yellow lights and has the pedestrian buttons available. The start button does the initial start of the system. The code for it is shown below in picture 5. Once the button is pressed it locks itself through not permitting any more events from it to the rest of the system.



Picture 5: Start resource

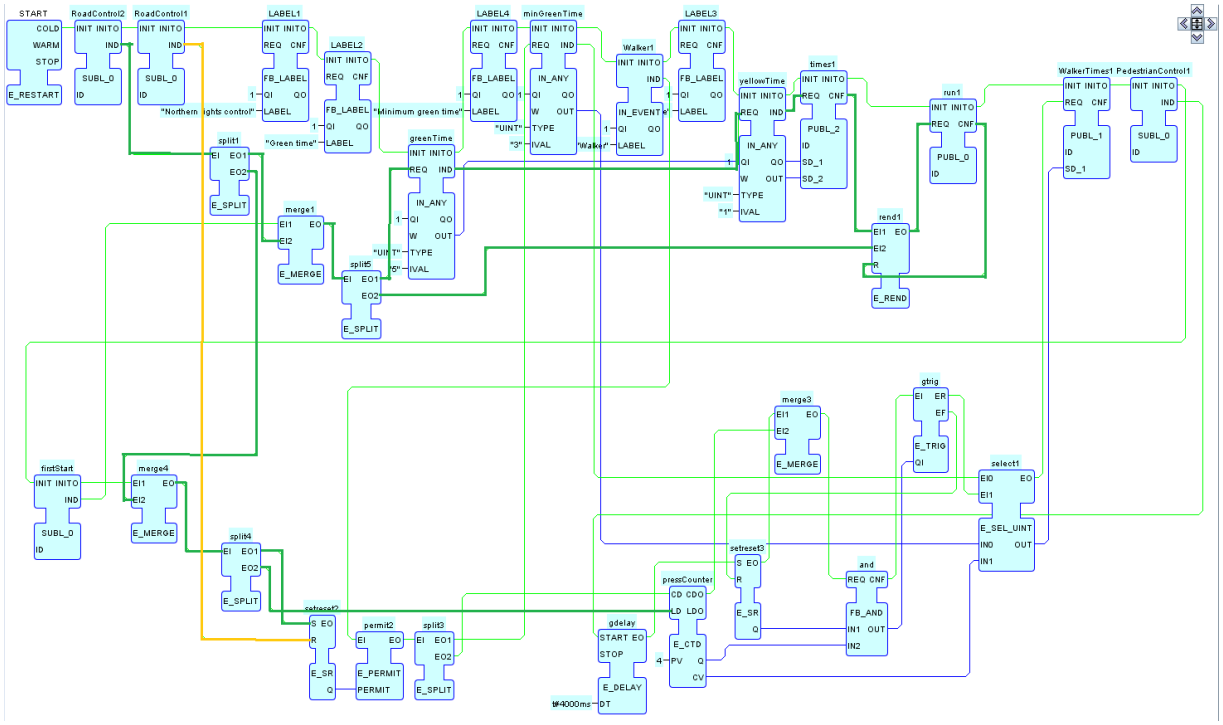
Both HMI_NORTH and -WEST resources are identical except for the initial start. The cycle of the lights starts from the northern lights. The initial first runs events and how they emerge are shown in picture 6.



Picture 6: firstStart

With the firstStart -publisher of picture 5, the firstStart -subscriber in picture 6 on the bottom left. The firstStart -block is at the end of the initialization events. Shown in orange line, the initialization also sets the permit bit for pedestrian buttons counter and loads the minimum required four presses to the counter.

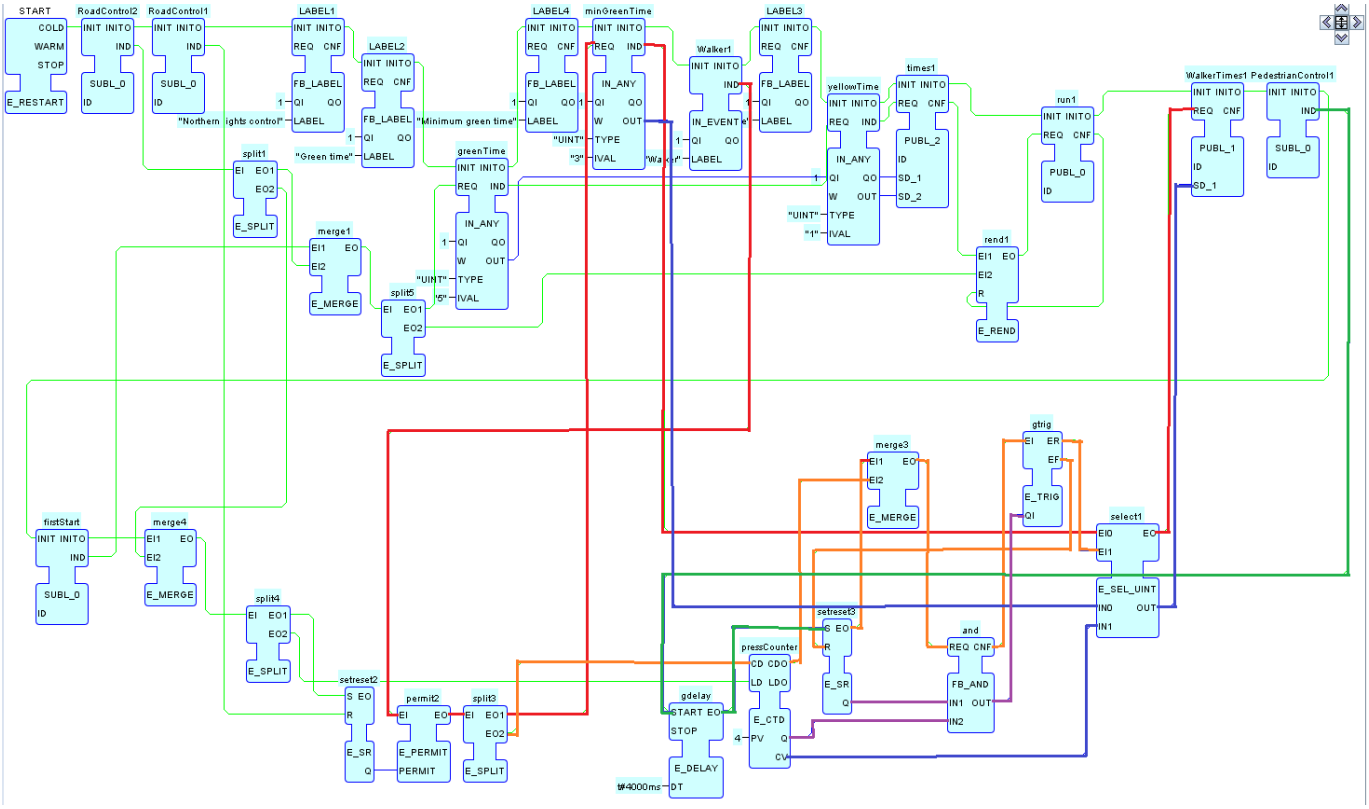
The redline shows how the firstStart -requests the set times from the fields of HMI and loads them to times1 -publisher. The event rendezvous block is placed before the run1 -publisher because changing the time parameters during the run won't trigger the lights to start over instantly.



Picture 7: Normal run cycle

Picture 7 shows the normal run cycle of the HMI resource. RoadControl2 -subscriber starts the run from top left corner (dark green line) by permitting the events from pedestrian button, loads the counter and requests the times for run1 -publisher. In orange color the RoadControl1 -subscriber resets the permit bit for pedestrian button events.

Picture 8 shows how the pedestrian button and the features around it works.



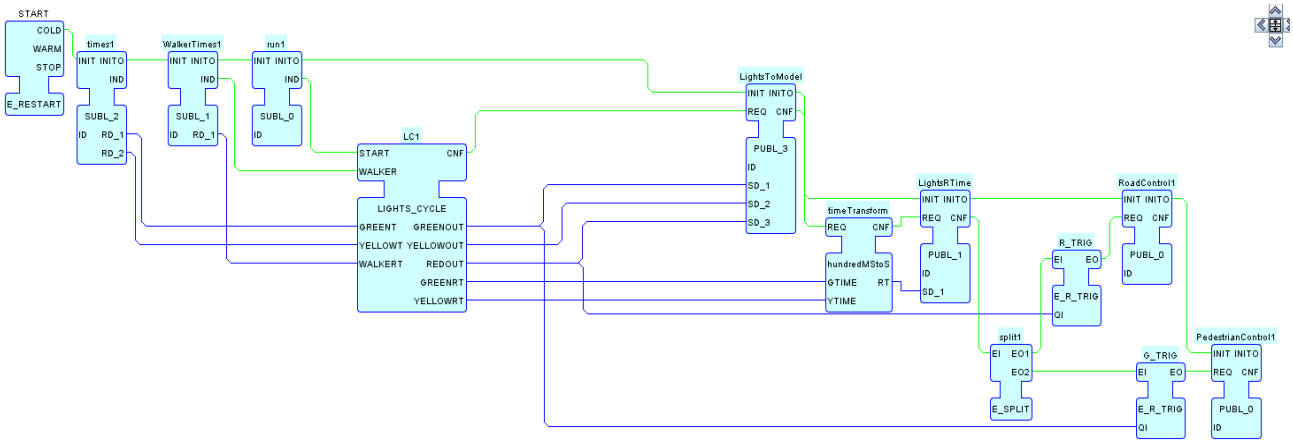
Picture 8: Pedestrian functionalities

If the pedestrian buttons events are permitted, as described in previous pictures, pressing the button request the set minimum green time (red event line) and sends it to WalkerTimes1 -publisher. Pedestrian button also counts down the pressCounter -counter on each press.

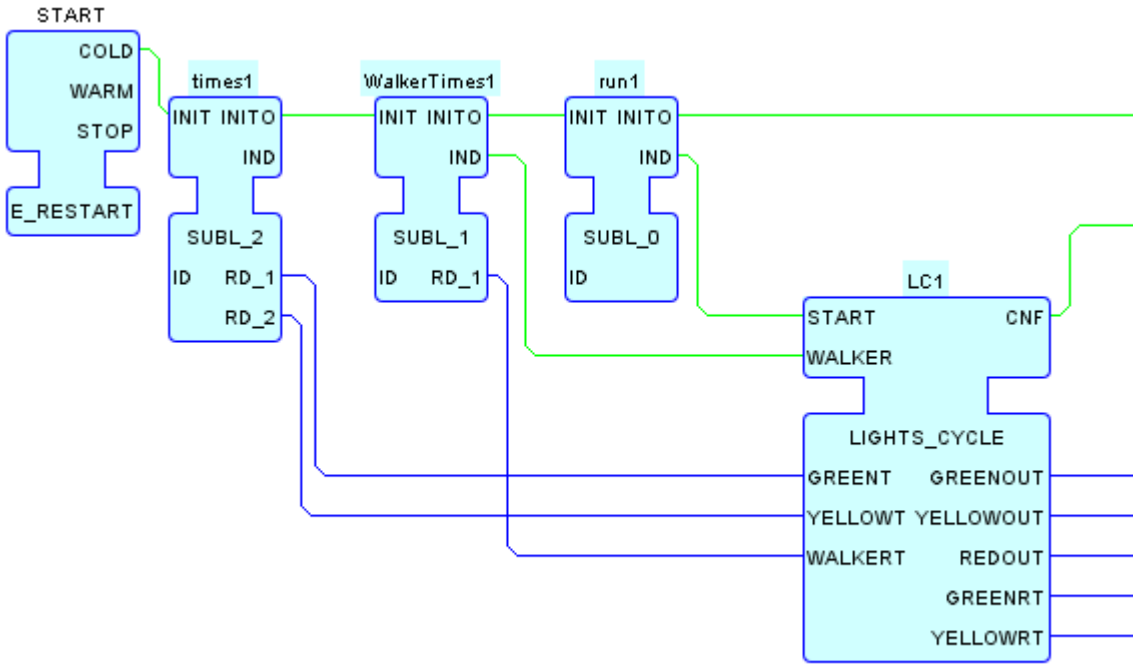
Top right corner the subscriber PedestrianControl1 gives a signal when the green light has turned on and starts a delayed event of 4 seconds (dark green line). Once the time has passed, it sets a bit. This bit and the one coming out of the counter (purple line) after the 4 presses are done, goes into a logical AND-gate which checks if the conditions are met to turn lights instantly to green. On the rising edge a value of “0” is sent to the WalkerTimes1 -publisher with an event signal.

CONTROL LAYER

Created control layer is shown in picture 9. Both, northern and western lights, have the same looking control layer.



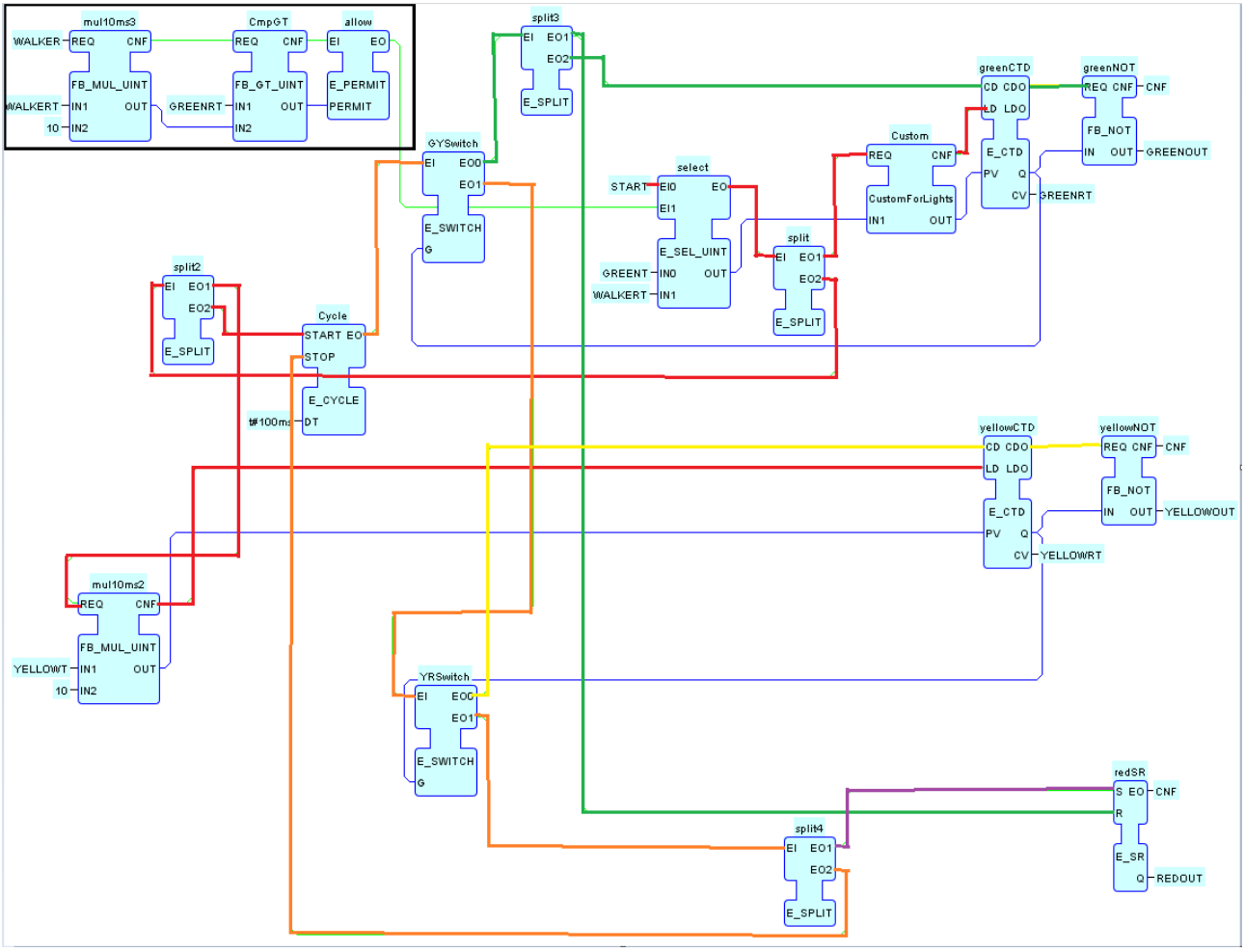
Picture 9: Control layer



Picture 10: start of control layer

Starting from top left; subscriber block times1 writes the set times form HMI to the LIGHTS_CYCLE composite block. WalkerTimes1 block writes the set minimum green time and triggers the WALKER event on the LIGHTS_CYCLE block. Run1 subscriber block triggers the normal cycle.

The LIGHTS_CYCLE composite block is shown in picture 11.



Picture 11: LIGHTS_CYCLE

On top left side, inside the black square, is the pedestrian time check so to say. If the pedestrian buttons event gets to the control layer, these three blocks first multiplies the minimum green time by ten because the cycle of the system is 100ms. After that, it compares it to the remaining green time. If the current remaining time is greater than the minimum green time, the event is allowed out. This brings us to the initial start block. If START event is brought, this loads the green time to the system, if the pedestrian part is permitted, it loads the minimum green time to the system.

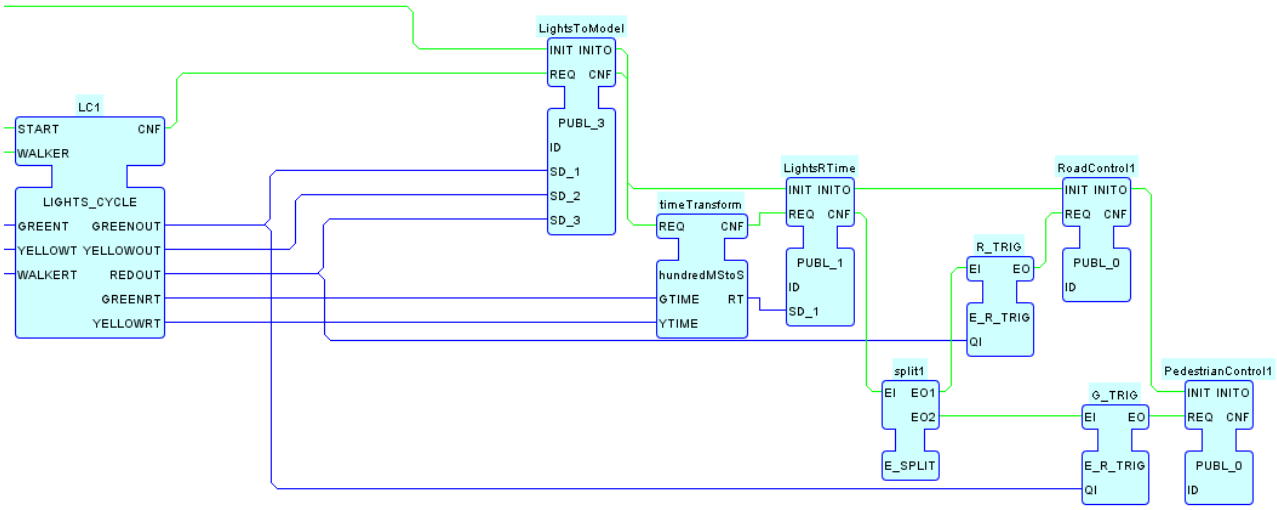
This event (red line) goes through a custom-made block, which checks the output of the select -block. As mentioned before, the 4 presses of the pedestrian button write “0” at the WALKERT variable. Because the counters do not set their output to “1” if you load a “0” in it, the Custom -block checks if the input value equals to zero. If it does, it writes “1” to the output, otherwise it multiplies the value by 10 and writes it to the output This ST code is presented in picture 12.

```
1 IF IN1 = 0 THEN
2 OUT := 1;
3 ELSE
4 OUT := IN1 * 10;
5 END_IF;
```

Picture 12: ST code

Continuing with the red event flow, the split block after select block goes to another split which one branch starts the clock of the system. Other branch multiplies the yellow time by 10 and loads it to yellowCTD.

From the cycle -block, the clock of the system the orange line goes to the first switch block. The 100ms event pulses (green line) first resets the REDOUT bit (bottom right) and then goes to the green lights counter. As long as the counter is not set, the logical NOT behind it writes “1” to the GREENOUT variable. Once the counter runs out, its output turn to “1”, GREENOUT turns to “0” and the GYSwitch turns the event pulses (orange line) to go through the YRSwitch to the yellow counter (yellow line). Same as with the green light, once the counter sets, the YRSwitch changes the event pulses to go and set the REDOUT bit (purple line) and with the same pulse stops the E_CYCLE block, the clock of the system. Both counters send out the remaining time in 100ms during the process.



Picture 13: end of control layer

After the LIGHT_CYCLE block, the states of each light is written to the LightsToModel -publisher block and the remaining times of green and yellow semaphores are written to theTimeTransform -basic block, which rounds the remaining times to seconds and publishes them with LightsRTime -block after. The ST -code of the TimeTranform -block is presented in picture 14.

```
1  IF GTIME>=10 THEN
2  RT := ((GTIME - 1 - ((GTIME - 1) MOD 10))/10) + 1;
3  ELSIF (GTIME<10 AND GTIME>0) THEN
4  RT := 1;
5  ELSIF (GTIME=0 AND YTIME>=10) THEN
6  RT := ((YTIME - 1 - ((YTIME - 1) MOD 10))/10) + 1;
7  ELSIF (GTIME=0 AND YTIME<10 AND YTIME>0) THEN
8  RT := 1;
9  ELSE
10 RT := 0;
11 END_IF;
```

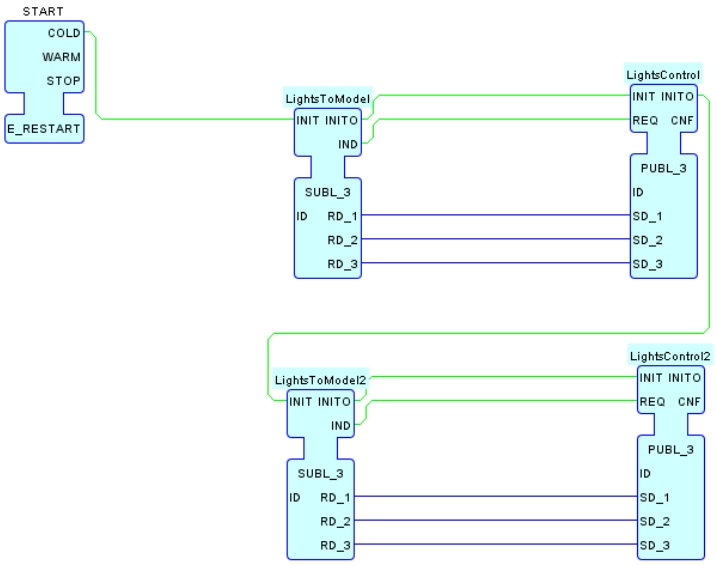
Picture 14: hundredMStoS -block

R_TRIG and G_TRIG event trigger blocks are checking when the lights turn and sends an event to the RoadControl1 and PedetsrianControl1 publishers when that happens.

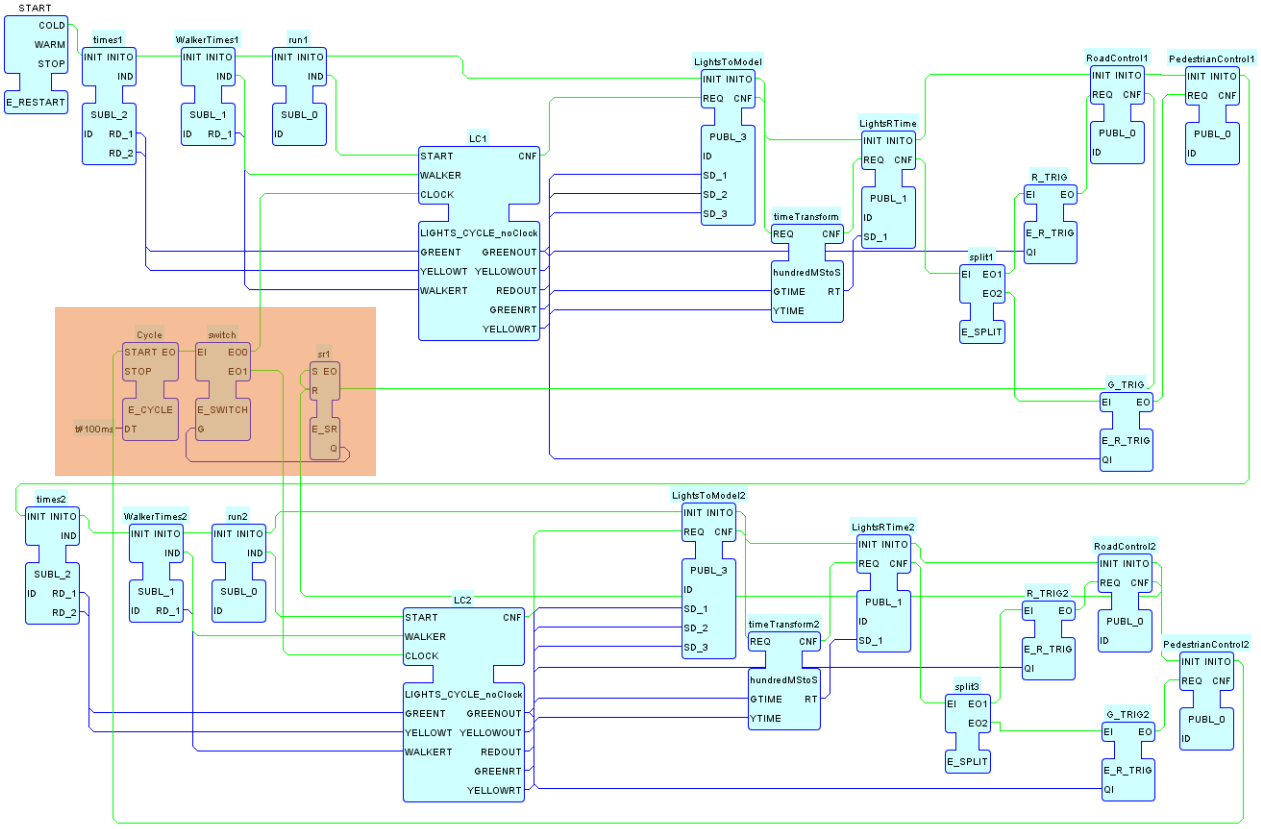
The control layer for the other lights are identical, except the names of different publishers etc. have a different number at the end.

CENTRALIZED APPROACH

For the centralized approach it was determined that the key element between distributed and centralized systems if that in centralized system the subsystems have their internal clocks whereas centralized works with a single clock. The distributed system was modified so that the models were merged (picture 15) and the event pulses give to LIGHT_CYCLE block were given from the outside. (picture 16). HMI and VIEW layer stayed the same.

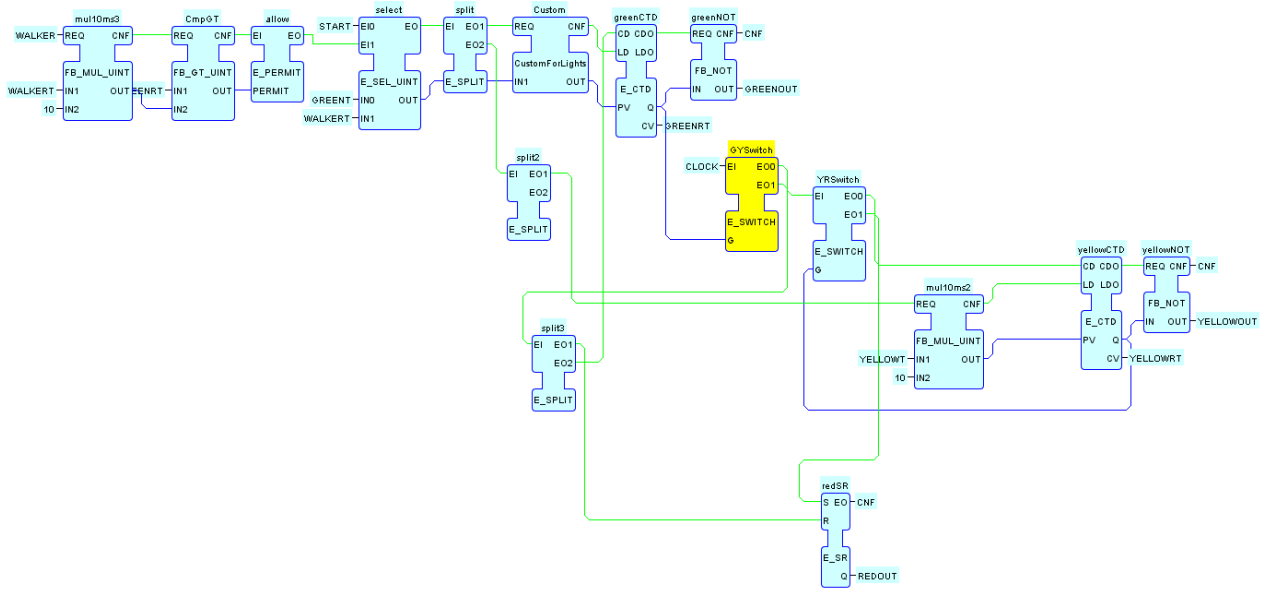


Picture 15: model layer of centralized approach



Picture 16: Control layer of centralized approach, red area shows the position of clock.

In the picture 17 the modified LIGHTS_CYCLE block is shown. The block highlighted in yellow takes in the clock pulses.



Picture 17: LIGHTS_CYCLE_noClock