Lab 1

TDDI08
Embedded Systems Design

Johan Olsson, johol842
Chung-Han Chiang, chuch234

IDA - Department of Computer and Information Science
Linköping University
17-02-2022
Version 1

Contents

1. Introduction

The laboratory is about a traffic light controller modeled and implemented in SystemC, and its simulated results. The controller controls four lights. Each light is responsible for one direction North–South(NS), South-North(SN), West–East(WE) and East-West(EW). For short we will only use the first letter: N, S, W and E of specifying directions. There are also four sensors detecting vehicles in the corresponding directions. The controller should satisfy following properties:

1. The lights should work independently.
2. Safety constraints: lights in conflicting directions should not be on at the same time.
3. If a vehicle arrives at the crossing (as detected by the respective sensor), it will eventually be granted the green light.

Throughout this report, we use the word *on* synonymous with *green*. Also *off* is synonymous with *red*. This is natural as we use boolean variables to represent the signals.
Within the code, arrays are used to describe signals, inputs or outputs. The order of lights and sensors when enumerated are *always* in the following order: N, S, W and E.

2. Our Design

The design of our model in SystemC uses different modules which are coupled together in a testbench. The input generator and monitor draw inspiration from given example code. The main design problem is the Traffic light controller itself.

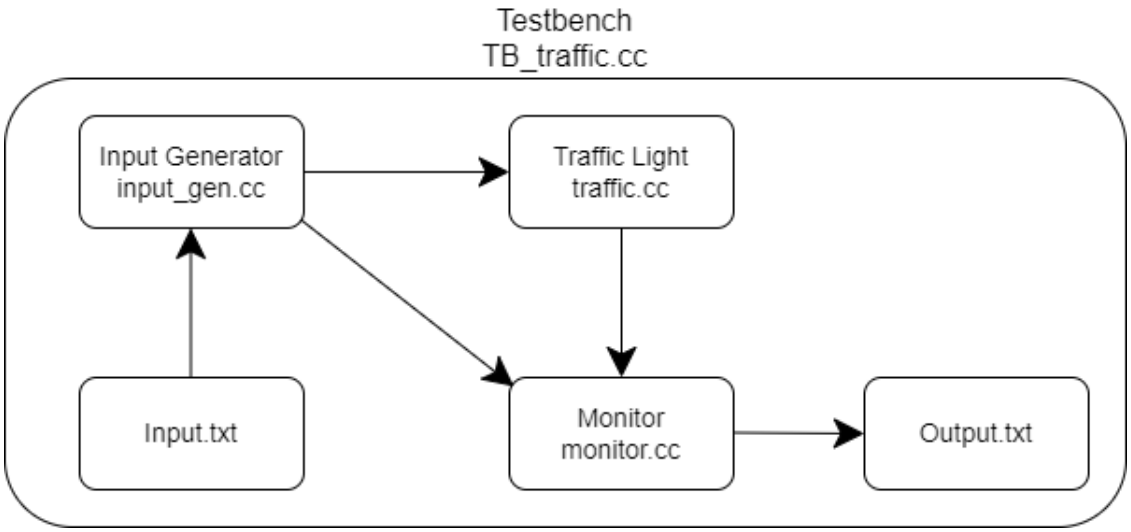

Figure 2.1: Overview of the system structure and how data flows between modules

2.1 Compiling and using the code

The code is compiled through a Make-file and must then be run with 3 commandline arguments:

```
./traffic.x TIME_TO_SIM INPUT_FILE OUTPUT_FILE
```

We usually use a *TIME_TO_SIM* of 20 for all of our small test-cases. The formatting of input files can be seen below:

```
0 0 0 0
0 0 0 0
0 0 0 0
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 1 0 0
0 0 0 0
0 0 0 0
```

Listing 2.1: Formatting of input samples. The columns represent N, S, W, E.

To generate arbitrary and random inputs, the file *random_gen.cc* is used. It takes a command line argument for how many rows of inputs it should generate. The inputs are written to a text file in the correct format. The txt-file name is hardcoded to be *input/random.txt* and can then be used as the INPUT_FILE argument in the testbench. The random generator is using more modern C++ and can be compiled and used like so on the university machines:

```
        w++17 random_gen.cc
        ./a.out NUM_OF_ROWS
```

The number of rows is also the number of seconds that the simulation will take, i.e. *TIME_TO_SIM* should be equal to *NUM_OF_ROWS* if one wants to use the whole input file.

2.2 Input Generator

The input generator reads the from the specified input file and writes all 4 boolean inputs every second. Through an SC_THREAD it accomplishes this to simulate cars approaching and leaving.

```
void Generator::generate_thread()
{
    bool n[4];
    for (;;)
    {
        wait(1, SC_SEC);      // Generate new inputs every second.
        *in >> n[0] >> n[1] >> n[2] >> n[3]; // Read from the input file.

        for(int i{}; i<4; i++)
        {
            cars[i]->write(n[i]);
        }
    }
}
```

Listing 2.2: The generating part of input_gen.cc

2.3 Monitor

A monitor is used to verify operation of the traffic-controller. It writes the state of both inputs (sensors) and outputs (trafficlights) to a txt-file. It uses an SC_METHOD which is sensitive to changes in these signals, but is also sensitive to an SC_THREAD. The THREAD triggers an event every second, which makes the monitor log atleast every second, even if there is no change on the inputs.

```
1  #include <cassert>
2  #include "monitor.h"
3
4  Monitor::Monitor(sc_module_name name, char *outfile)
5    : sc_module(name)
6  {
7    assert(outfile != 0);
8    out = new ofstream(outfile);
9    assert(*out);
10
11   SC_THREAD(event_trigger_thread);
12
13   SC_METHOD(monitor_method);
14   dont_initialize();
15   sensitive << cars[0] << cars[1] << cars[2] << cars[3] << lights[0] <<
      lights[1] << lights[2] << lights[3];
16   sensitive << count_event;
```

Listing 2.3: Structure of the monitor module.

To verify the safety constraint imposed by the problem, the monitor checks for conflicting directions. There is an *assert* to verify that the lights are never both green in conflicting directions. The monitor also logs an error to the output file if this happens. Although this is redundant as the assert will stop execution, it is nice to have when debugging.

```
39   if( (lights[0] || lights[1]) && (lights[2] || lights[3] ) ) //checks if NS
       are on at the same time as WE
40   {
41       *out << "ERROR";
42   }
43   assert( !((lights[0] || lights[1]) && (lights[2] || lights[3] )) );
```

Listing 2.4: Verification of safety constraints within monitor.cc. The if-block and assert check for the same value, it´s just negated in the assert case.

2.4 Traffic Controller

The heart of the project is the controller for the trafficlight. It uses an SC_METHOD that is sensitive to changes in the sensors. It turns on the lights instantly if there are no conflicting sensors activated: i.e if both sensor N and S are off - then it can freely turn on W and E, and vice versa. See the following code listings:

```
11   SC_THREAD(trigger_timer);
12
13   SC_METHOD(turn_on_lights);
14   dont_initialize();
15   sensitive << cars[0] << cars[1] << cars[2] << cars[3];
16   sensitive << trigger_event;
```

Listing 2.5: Structure of the traffic controller module

```
28 void Traffic::turn_on_lights()
29 {
30     bool N,S,W,E;
31     N = cars[0]; S = cars[1]; W = cars[2]; E = cars[3];
32
33     // DEFAULT: turn off lights
34     lights[0] = false;
35     lights[1] = false;
36     lights[2] = false;
37     lights[3] = false;
38
39     if( !(W || E) )  //let cars through for N and S, if no cars at W or E
40     {
41         lights[0] = N;
42         lights[1] = S;
43     }
44     else if( !(N || S) )  //let cars through for W and E, if no cars at N or
       S
45     {
46         lights[2] = W;
47         lights[3] = E;
48     }
49     else if( timer )
50     {
51         lights[0] = N;
52         lights[1] = S;
53
54     }
55     else if( !timer )
56     {
57         lights[2] = W;
58         lights[3] = E;
```

```
59
60      }
```

Listing 2.6: Logic within *traffic.cc* which handles how the lights turn on

An SC_THREAD is used to handle the problem of conflicting directions and the problem of starvation. The controller use a boolean variable called *timer* to determine which of the two direction goes first. The THREAD changes the boolean every 5 seconds. The THREAD also forces the METHOD to activate by triggering an event it is sensitive to. This is because the METHOD will not active under some circumstances, as it relies on changing sensor inputs to get going. The feature causes cars to be let through within 5 seconds of arriving, independant of traffic.

```
19 void Traffic::trigger_timer()
20 {
21     for (;;)
22     {
23     wait(5, SC_SEC);
24     trigger_event.notify();
25     timer = !timer;  // change boolean timer to change priority periodically
26     }
27 }
```

Listing 2.7: The thread which handles starvation and conflicting directions.

3. Simulation Verification

Inputs which simulate different situations were made to test and help verify the model.

3.1 Independant

To test the property of the lights being independant, *Independant_X.txt* was made for the 2 conflicting directions. The input and output of one of those tests be seen in listing 3.1 and 3.2. The inputs for the other directions, and the complete output logs are located in the given files.

3.2 Safety

To verify the safety of the traffic controller, we can use 100 000 rows of random inputs. When no assertions fail, we can be very certain that our model fulfills this requirement. This can be done by calling the testbench with the given input file *random.txt*. There will be no demonstration of logs for this test, for obvious reasons.

3.3 Starvation

To test the property of suppressing starvation, *Starvation_X.txt* was made for all 4 directions. The input and output of one of those tests can be seen in listing 3.3 and 3.4. The inputs for the other directions, and the complete output logs are located in the given files.

Logs from testing

```
0 0 0 0
0 0 0 0
1 0 0 0
1 0 0 0
1 0 0 0
1 0 0 0
1 0 0 0
1 1 0 0
1 0 0 0
1 1 0 0
0 0 0 0
0 0 0 0
0 1 0 0
0 1 0 0
0 1 0 0
0 1 0 0
0 1 0 0
1 1 0 0
0 1 0 0
0 0 0 0
```

Listing (3.1) Input for Independant_NS.txt

```
Cars  (1 s) = 0, 0, 0, 0,
Lights(1 s) = 0, 0, 0, 0,

Cars  (2 s) = 0, 0, 0, 0,
Lights(2 s) = 0, 0, 0, 0,

Cars  (3 s) = 0, 0, 0, 0,
Lights(3 s) = 0, 0, 0, 0,

Cars  (3 s) = 1, 0, 0, 0,
Lights(3 s) = 0, 0, 0, 0,

Cars  (3 s) = 1, 0, 0, 0,
Lights(3 s) = 1, 0, 0, 0,

Cars  (4 s) = 1, 0, 0, 0,
Lights(4 s) = 1, 0, 0, 0,

Cars  (5 s) = 1, 0, 0, 0,
Lights(5 s) = 1, 0, 0, 0,

Cars  (6 s) = 1, 0, 0, 0,
Lights(6 s) = 1, 0, 0, 0,

Cars  (7 s) = 1, 0, 0, 0,
Lights(7 s) = 1, 0, 0, 0,

Cars  (8 s) = 1, 0, 0, 0,
Lights(8 s) = 1, 0, 0, 0,

Cars  (8 s) = 1, 1, 0, 0,
Lights(8 s) = 1, 0, 0, 0,

Cars  (8 s) = 1, 1, 0, 0,
Lights(8 s) = 1, 1, 0, 0,

Cars  (9 s) = 1, 1, 0, 0,
Lights(9 s) = 1, 1, 0, 0,

Cars  (9 s) = 1, 0, 0, 0,
Lights(9 s) = 1, 1, 0, 0,

Cars  (9 s) = 1, 0, 0, 0,
Lights(9 s) = 1, 0, 0, 0,
```

Listing (3.2) Some of the output of Independant_NS.txt

```
1 1 0 0
1 1 0 0
1 1 0 0
1 1 0 0
1 1 0 0
1 1 0 1
1 1 0 1
1 1 0 1
1 1 0 1
1 1 0 1
1 1 0 1
1 1 0 1
1 1 1 1
1 1 0 1
1 1 0 1
1 1 0 0
1 1 0 0
1 1 0 0
1 1 0 0
0 0 0 0
```

Listing (3.3) Input for Starvation_E.txt

```
Cars  (1 s) = 0, 0, 0, 0,
Lights(1 s) = 0, 0, 0, 0,

Cars  (1 s) = 1, 1, 0, 0,
Lights(1 s) = 0, 0, 0, 0,

Cars  (1 s) = 1, 1, 0, 0,
Lights(1 s) = 1, 1, 0, 0,

Cars  (2 s) = 1, 1, 0, 0,
Lights(2 s) = 1, 1, 0, 0,

Cars  (3 s) = 1, 1, 0, 0,
Lights(3 s) = 1, 1, 0, 0,

Cars  (4 s) = 1, 1, 0, 0,
Lights(4 s) = 1, 1, 0, 0,

Cars  (5 s) = 1, 1, 0, 0,
Lights(5 s) = 1, 1, 0, 0,

Cars  (6 s) = 1, 1, 0, 0,
Lights(6 s) = 1, 1, 0, 0,

Cars  (6 s) = 1, 1, 0, 1,
Lights(6 s) = 1, 1, 0, 0,

Cars  (7 s) = 1, 1, 0, 1,
Lights(7 s) = 1, 1, 0, 0,

Cars  (8 s) = 1, 1, 0, 1,
Lights(8 s) = 1, 1, 0, 0,

Cars  (9 s) = 1, 1, 0, 1,
Lights(9 s) = 1, 1, 0, 0,

Cars  (10 s) = 1, 1, 0, 1,
Lights(10 s) = 1, 1, 0, 0,

Cars  (10 s) = 1, 1, 0, 1,
Lights(10 s) = 0, 0, 0, 1,

Cars  (11 s) = 1, 1, 0, 1,
Lights(11 s) = 0, 0, 0, 1,
```

Listing (3.4) Some of the output of Starvation_E.txt