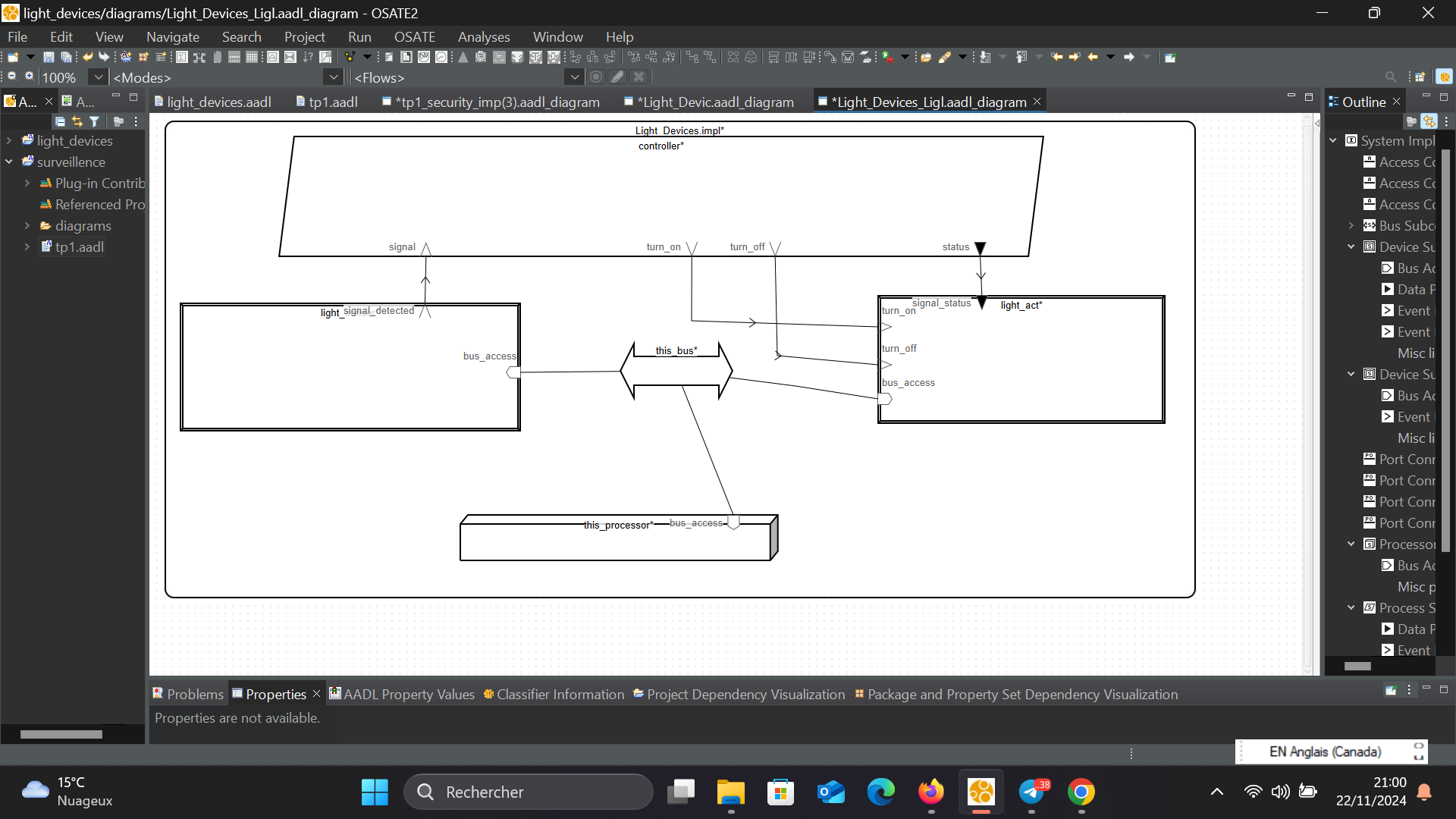
|  |
| --- |
| TP software engineering |
| Belkahla saoussen  Nouioua Zineb  **SEM1** |



**package** Light\_Devices

**public**

**system** Light\_Devices

**end** Light\_Devices;

**system** **implementation** Light\_Devices.impl

**subcomponents**

light\_sensor: **device** light\_sensor.impl;

controller: **process** controller.impl;

light\_act: **device** light\_act.impl;

this\_bus: **bus** communication\_bus.impl;

this\_processor: **processor** processing\_unit.impl;

**connections**

sensor\_to\_controller: **port** light\_sensor.signal\_detected -> controller.signal;

controller\_to\_actuator: **port** controller.status -> light\_act.signal\_status;

controller\_to\_actuators: **port** controller.turn\_on -> light\_act.turn\_on;

controller\_to\_actuatore: **port** controller.turn\_off -> light\_act.turn\_off;

sensor\_to\_bus: **bus** **access** light\_sensor.bus\_access <-> this\_bus;

actuator\_to\_bus: **bus** **access** light\_act.bus\_access <-> this\_bus;

processor\_to\_bus: **bus** **access** this\_processor.bus\_access <-> this\_bus;

**end** Light\_Devices.impl;

**device** light\_sensor

**features**

signal\_detected: **out** **event** **port**;

bus\_access: **requires** **bus** **access** communication\_bus.impl;

**end** light\_sensor;

**device** **implementation** light\_sensor.impl

**end** light\_sensor.impl;

**process** controller

**features**

signal: **in** **event** **port**; -- Réception du signal du capteur

turn\_on: **out** **event** **port**; -- Commande d'allumage

turn\_off: **out** **event** **port**; -- Commande d'extinction

status: **out** **data** **port**; -- État du contrôleur

**end** controller;

**process** **implementation** controller.impl

**end** controller.impl;

**device** light\_act

**features**

signal\_status: **in** **data** **port**;

turn\_on: **in** **event** **port**;

turn\_off: **in** **event** **port**;

bus\_access: **requires** **bus** **access** communication\_bus.impl;

**end** light\_act;

**device** **implementation** light\_act.impl

**end** light\_act.impl;

**bus** communication\_bus

**end** communication\_bus;

**bus** **implementation** communication\_bus.impl

**end** communication\_bus.impl;

-- Définition du processeur

**processor** processing\_unit

**features**

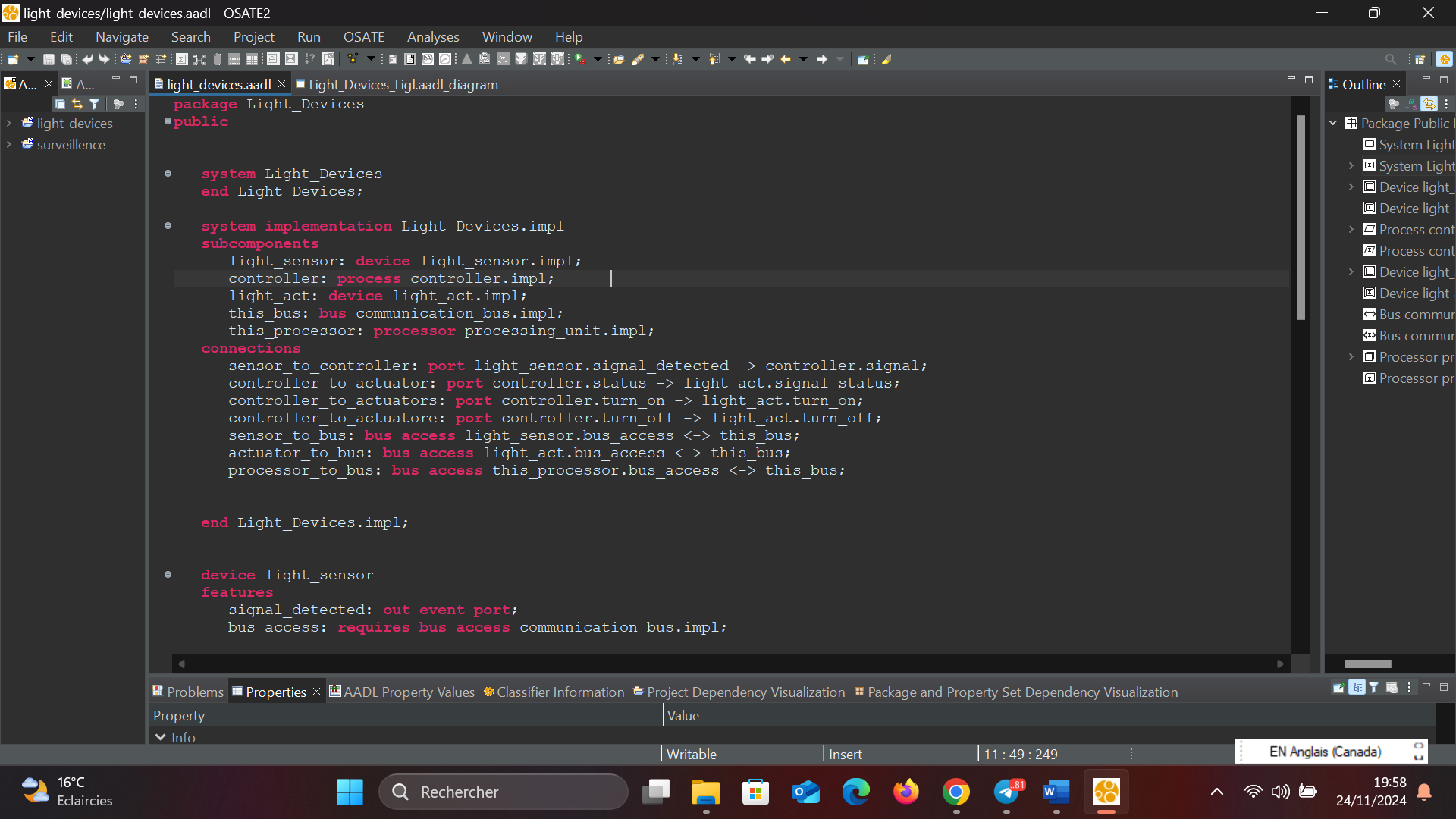
bus\_access: **requires** **bus** **access** communication\_bus.impl;

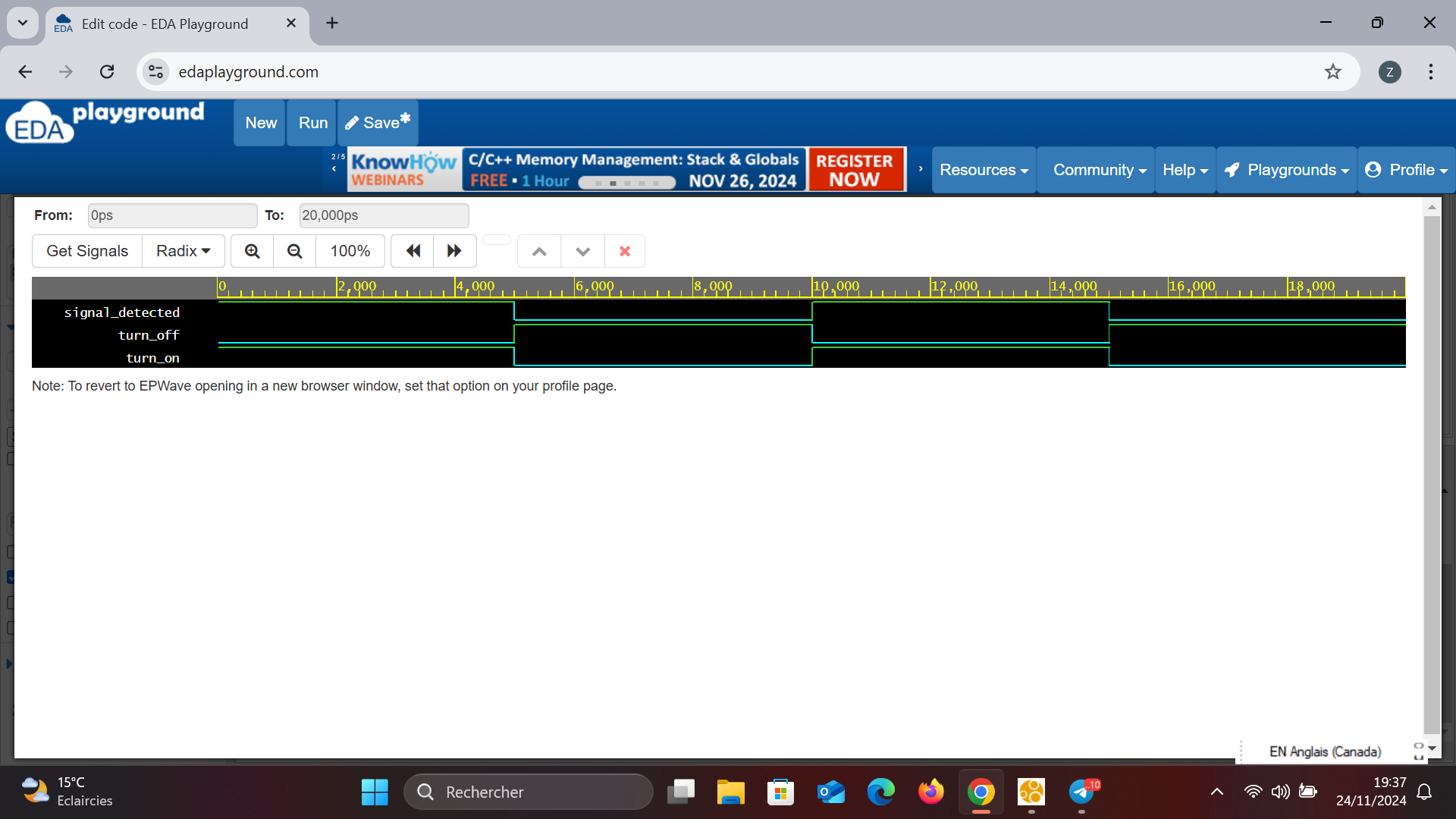
**end** processing\_unit;

**processor** **implementation** processing\_unit.impl

**end** processing\_unit.impl;

**end** Light\_Devices;





// Code your testbench here.

// Uncomment the next line for SystemC modules.

// #include <systemc>

#include <systemc.h>

#include <fstream>

// Module pour le capteur

SC\_MODULE(LightSensor) {

sc\_out<bool> signal\_detected;

SC\_CTOR(LightSensor) {

SC\_THREAD(generate\_signal);

}

void generate\_signal() {

while (true) {

signal\_detected.write(true); // Signal détecté

wait(5, SC\_NS); // Attendre 5 ns

signal\_detected.write(false); // Aucun signal

wait(5, SC\_NS); // Attendre 5 ns

}

}

};

// Module pour le contrôleur

SC\_MODULE(Controller) {

sc\_in<bool> signal;

sc\_out<bool> turn\_on, turn\_off;

SC\_CTOR(Controller) {

SC\_METHOD(process\_signal);

sensitive << signal;

}

void process\_signal() {

if (signal.read()) {

turn\_on.write(true); // Allumer

turn\_off.write(false);

} else {

turn\_on.write(false);

turn\_off.write(true); // Éteindre

}

}

};

// Module pour l'actionneur

SC\_MODULE(LightActuator) {

sc\_in<bool> turn\_on, turn\_off;

SC\_CTOR(LightActuator) {

SC\_METHOD(control\_light);

sensitive << turn\_on << turn\_off;

}

void control\_light() {

if (turn\_on.read()) {

std::cout << sc\_time\_stamp() << " Light Actuator: ON\n";

}

if (turn\_off.read()) {

std::cout << sc\_time\_stamp() << " Light Actuator: OFF\n";

}

}

};

// Fonction pour générer le fichier Graphviz

void generate\_graph() {

std::ofstream file("system\_graph.dot");

file << "digraph SystemCGraph {\n";

file << " rankdir=LR;\n"; // Orientation de gauche à droite

// Déclaration des noeuds

file << " light\_sensor [label=\"Light Sensor\" shape=box];\n";

file << " controller [label=\"Controller\" shape=ellipse];\n";

file << " light\_actuator [label=\"Light Actuator\" shape=box];\n";

// Connexions entre les noeuds

file << " light\_sensor -> controller [label=\"signal\_detected\"];\n";

file << " controller -> light\_actuator [label=\"turn\_on\"];\n";

file << " controller -> light\_actuator [label=\"turn\_off\"];\n";

file << "}\n";

file.close();

}

// Fonction principale

int sc\_main(int argc, char\* argv[]) {

// Générer le fichier Graphviz

generate\_graph();

// Signaux

sc\_signal<bool> signal\_detected\_signal, turn\_on\_signal, turn\_off\_signal;

// Instanciation des modules

LightSensor sensor("LightSensor");

Controller controller("Controller");

LightActuator actuator("LightActuator");

// Connexions

sensor.signal\_detected(signal\_detected\_signal);

controller.signal(signal\_detected\_signal);

controller.turn\_on(turn\_on\_signal);

controller.turn\_off(turn\_off\_signal);

actuator.turn\_on(turn\_on\_signal);

actuator.turn\_off(turn\_off\_signal);

// Fichier VCD pour visualiser les signaux

sc\_trace\_file \*trace\_file = sc\_create\_vcd\_trace\_file("dump");

sc\_trace(trace\_file, signal\_detected\_signal, "signal\_detected");

sc\_trace(trace\_file, turn\_on\_signal, "turn\_on");

sc\_trace(trace\_file, turn\_off\_signal, "turn\_off");

// Lancement de la simulation

sc\_start(20, SC\_NS); // Simulation pour 20 nanosecondes

sc\_close\_vcd\_trace\_file(trace\_file);

return 0;

}

