OO Analysis and OO Design and Implementation Assignment

Semester 3 Embedded Systems

OO Analysis and OO Design

Jan 20, 2023.

Student: Andre Sanao

Course: Technology

Table of Contents

Acronyms

Acronym	Meaning
ICT	→ Information and communication technologies

Table 1 – List of acronyms used throughout the report

Introduction

The assignment on which this document presents an overview of various types of ICT diagrams used to represent embedded systems. Embedded systems are computer systems that are integrated into devices or products and are used to perform a specific function or set of functions. They are commonly used in fields such as manufacturing, transportation, and healthcare. In the following sections, we will provide an overview of the different types of ICT diagrams used to represent embedded systems and provide examples of how they can be used.

Dynamic Behavior: Microwave

Part 2.1: Implement and unit test the Microwave state diagram

In the part of the assignment, it is required to implement our knowledge that we learned in the semester. This part focuses mainly on making a state diagram.

A state diagram is a type of diagram that consists of states, transitions, events, and activities. State diagrams are used to illustrate the dynamic view of a system.

The system that will be used in the assignment is a microwave oven. The task is to extend the behavior of the microwave by adding states and describing the added behavior. The following figure will show how I designed the microwave oven's behavior. (Figure 1)

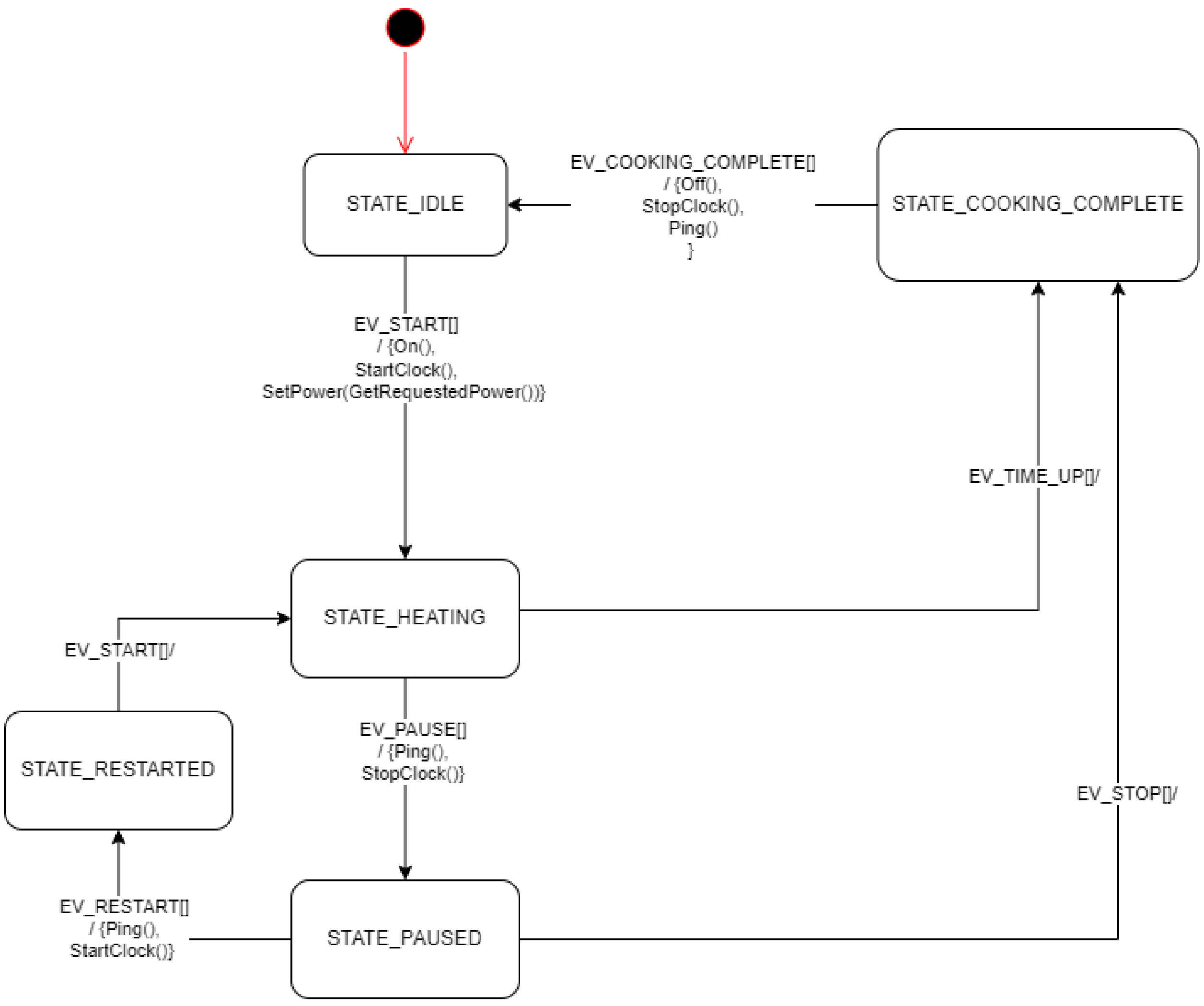

STATE_IDLE
EV_COOKING_COMPLETE[] / {Off(), StopClock(), Ping() }
STATE_COOKING_COMPLETE
EV_START[] / {On(), StartClock(), SetPower(GetRequestedPower())}
EV_TIME_UP[]/
STATE_HEATING
EV_START[]/
STATE_RESTARTED
EV_PAUSE[] / {Ping(), StopClock()}
EV_STOP[]/
EV_RESTART[] / {Ping(), StartClock()}
STATE_PAUSED

Figure 1 Microwave State diagram

The microwave state diagram has a simple design behavior that does what a user should expect. I thought off adding extra features like a defrosting or oven state but that would make the state diagram way too large.

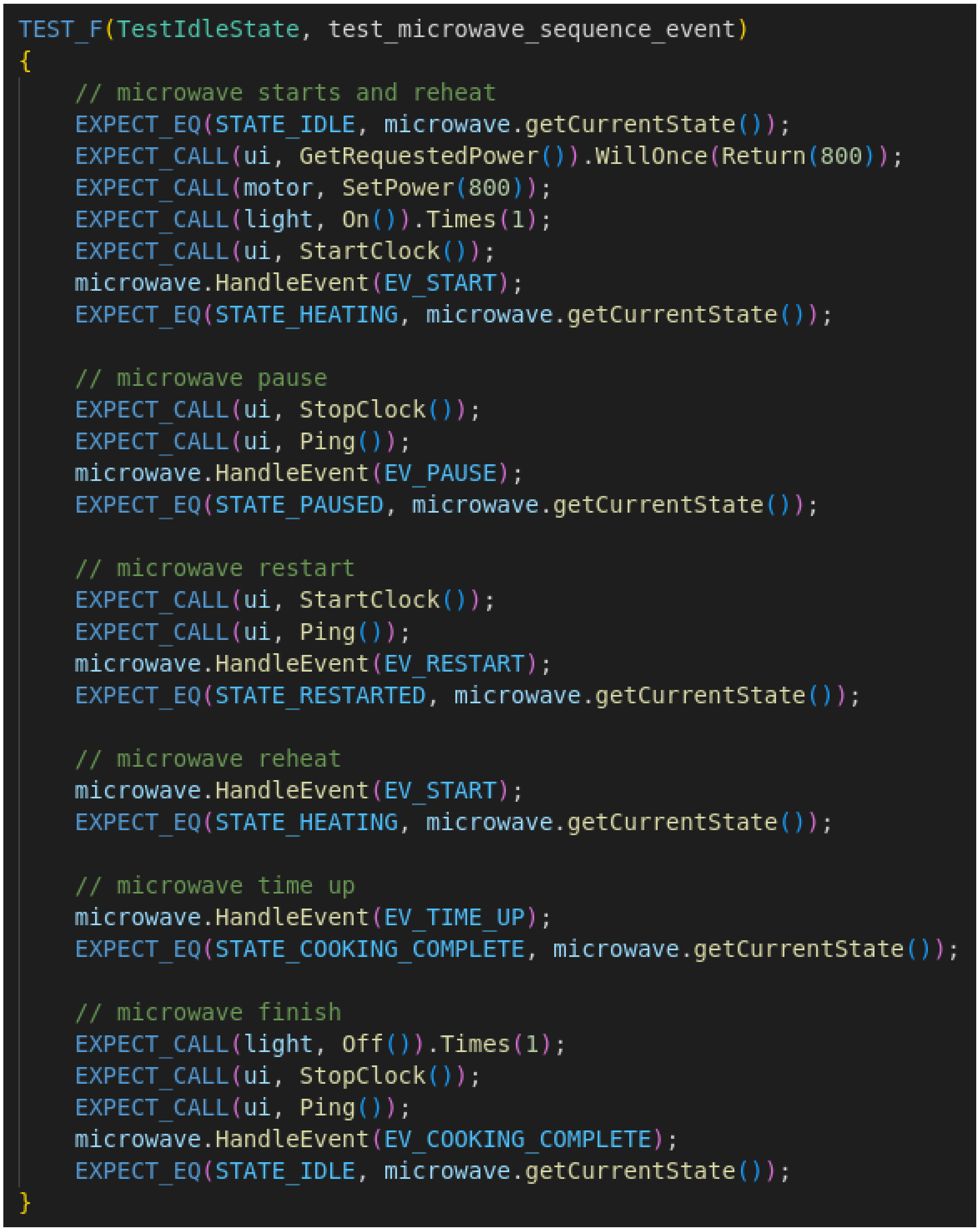
A table below provides function message with its description. (Table 1)

Direction	Message name:	Description:
In msg	On()	Switches the light in the microwave on.
	Off()	Switches the light in the microwave off.
	StartClock()	Enables the time dial to slowly turn back to 0.
	StopClock()	Stops the time dial.
	SetPower(pow)	Sets the microwave engine to the required power**(pow).**
	GetRequestedPower()	Returns the power dialed by the user.
	Ping()	Gives a subtle beeping sound to notify user.

Table 1 Microwave message description

After designing the state machine, it is required to test the microwave oven in a C++. The code does not provide actual full detailed code that makes the full use of the functions. The code is mainly to design a state machine that can be used in a mock test. More details of the microwave code will be in the folder together with this report.

Mock test is when you write a prototype or test, often not feasible or wise to rely on real objects entirely. A mock object is used to implement the same interface as a real object, but lets you specify at run time how it will be used and what it should do. The following figure will show the mock tests used for the microwave oven.

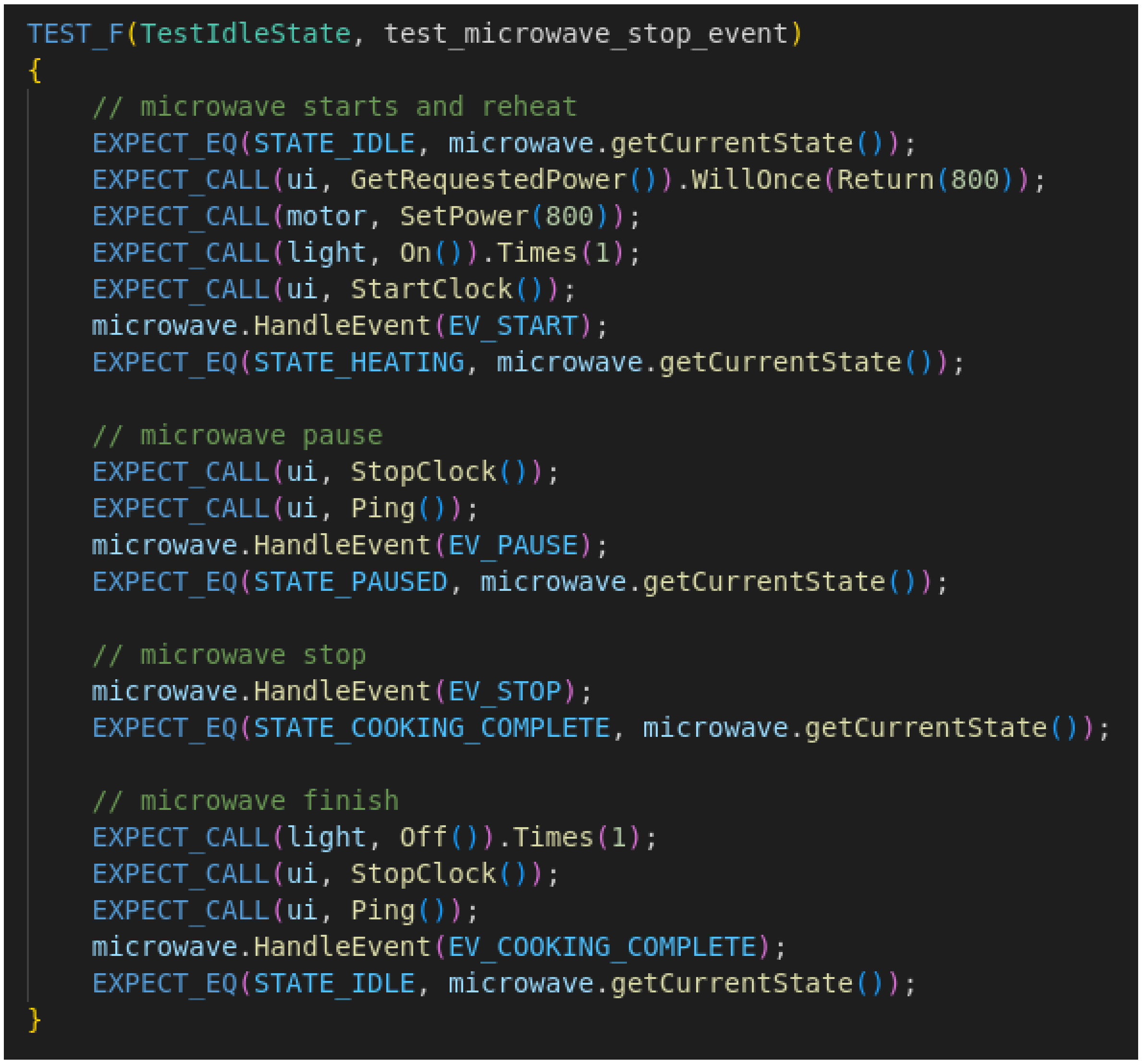


```
TEST_F(TestIdleState, test_microwave_sequence_event)
{
    // microwave starts and reheat
    EXPECT_EQ(STATE_IDLE, microwave.getCurrentState());
    EXPECT_CALL(ui, GetRequestedPower()).WillOnce(Return(800));
    EXPECT_CALL(motor, SetPower(800));
    EXPECT_CALL(light, On()).Times(1);
    EXPECT_CALL(ui, StartClock());
    microwave.HandleEvent(EV_START);
    EXPECT_EQ(STATE_HEATING, microwave.getCurrentState());

    // microwave pause
    EXPECT_CALL(ui, StopClock());
    EXPECT_CALL(ui, Ping());
    microwave.HandleEvent(EV_PAUSE);
    EXPECT_EQ(STATE_PAUSED, microwave.getCurrentState());

    // microwave restart
    EXPECT_CALL(ui, StartClock());
    EXPECT_CALL(ui, Ping());
    microwave.HandleEvent(EV_RESTART);
    EXPECT_EQ(STATE_RESTARTED, microwave.getCurrentState());

    // microwave reheat
    microwave.HandleEvent(EV_START);
    EXPECT_EQ(STATE_HEATING, microwave.getCurrentState());

    // microwave time up
    microwave.HandleEvent(EV_TIME_UP);
    EXPECT_EQ(STATE_COOKING_COMPLETE, microwave.getCurrentState());

    // microwave finish
    EXPECT_CALL(light, Off()).Times(1);
    EXPECT_CALL(ui, StopClock());
    EXPECT_CALL(ui, Ping());
    microwave.HandleEvent(EV_COOKING_COMPLETE);
    EXPECT_EQ(STATE_IDLE, microwave.getCurrentState());
}
```

Figure 2 Mock Test 2

Figure 2 shows the full sequence of how the user will use a microwave oven. Firstly, the user dials the sets the power and starts the microwave from idle state. The clock starts counting down at the same time heat the food but the user decided to pause the microwave. Afterwards, the user restarts the microwave which continues to countdown and reheat the food. When the time is up, the cooking of the food will be completed and the microwave will go back to idle state.

The next figure will show another situation of how the user interacts with the microwave. (Figure 3)

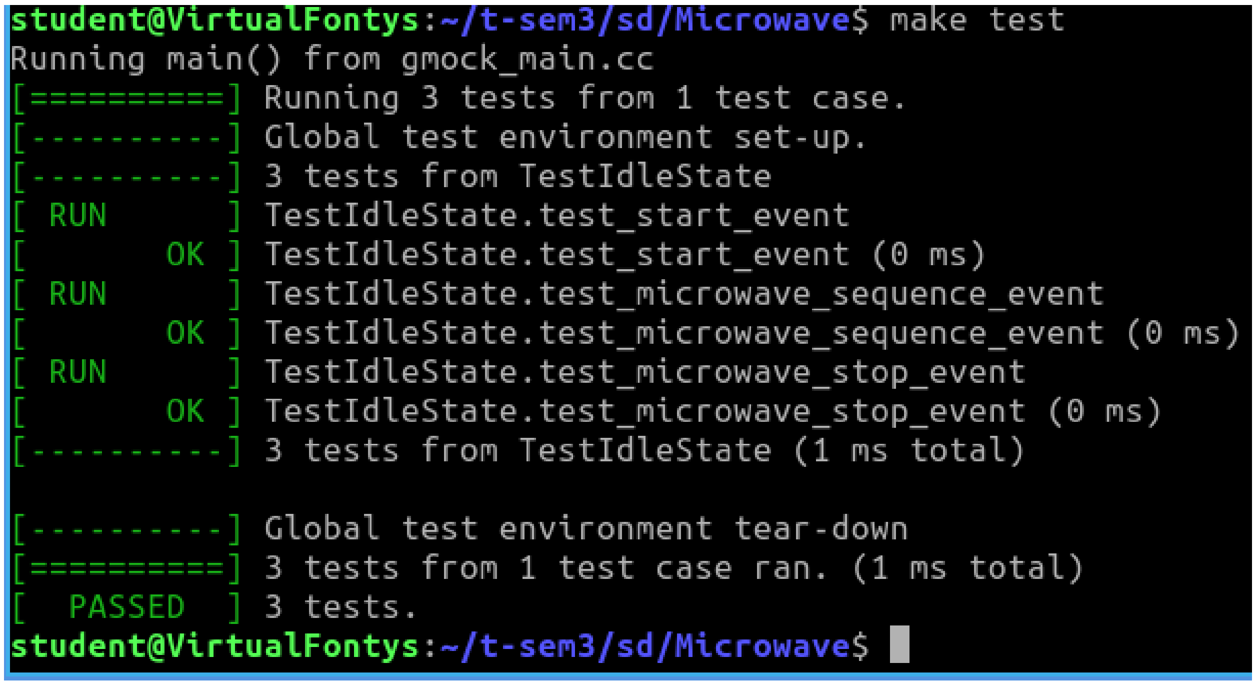


```
TEST_F(TestIdleState, test_microwave_stop_event)
{
    // microwave starts and reheat
    EXPECT_EQ(STATE_IDLE, microwave.getCurrentState());
    EXPECT_CALL(ui, GetRequestedPower()).WillOnce(Return(800));
    EXPECT_CALL(motor, SetPower(800));
    EXPECT_CALL(light, On()).Times(1);
    EXPECT_CALL(ui, StartClock());
    microwave.HandleEvent(EV_START);
    EXPECT_EQ(STATE_HEATING, microwave.getCurrentState());

    // microwave pause
    EXPECT_CALL(ui, StopClock());
    EXPECT_CALL(ui, Ping());
    microwave.HandleEvent(EV_PAUSE);
    EXPECT_EQ(STATE_PAUSED, microwave.getCurrentState());

    // microwave stop
    microwave.HandleEvent(EV_STOP);
    EXPECT_EQ(STATE_COOKING_COMPLETE, microwave.getCurrentState());

    // microwave finish
    EXPECT_CALL(light, Off()).Times(1);
    EXPECT_CALL(ui, StopClock());
    EXPECT_CALL(ui, Ping());
    microwave.HandleEvent(EV_COOKING_COMPLETE);
    EXPECT_EQ(STATE_IDLE, microwave.getCurrentState());
}
```

Figure 3 Mock Test 3

Figure 3 shows the same start sequence as the first test. This time the user pauses the microwave which goes to paused state. The user does not want to wait until the countdown reaches 0 so it stops the microwave which straight away completes the cooking process and goes back to idle state.

These tests were check if they are successful using the “make test” command in Linux. The following diagram shows successful tests. (Figure 4)



```
student@VirtualFontys:~/t-sem3/sd/Microwave$ make test
Running main() from gmock_main.cc
[==========] Running 3 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 3 tests from TestIdleState
[ RUN      ] TestIdleState.test_start_event
[       OK ] TestIdleState.test_start_event (0 ms)
[ RUN      ] TestIdleState.test_microwave_sequence_event
[       OK ] TestIdleState.test_microwave_sequence_event (0 ms)
[ RUN      ] TestIdleState.test_microwave_stop_event
[       OK ] TestIdleState.test_microwave_stop_event (0 ms)
[----------] 3 tests from TestIdleState (1 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test case ran. (1 ms total)
[  PASSED  ] 3 tests.
student@VirtualFontys:~/t-sem3/sd/Microwave$
```

Figure 4 Terminal Test Success

There are 3 tests showing up because the first test is an example test provided by the assignment.

Conclusion

To conclude this assignment, designing the state diagram during the lecture gave it a simple state machine with the purpose of heating and then going back to idle state. I saw that as a very short state diagram and restructured it to become more functional for certain situation like stopping the microwave when the user is already satisfied. Most conventional microwave has this so it is nothing new. I was also thinking of giving it another feature like defrosting state but the newly created state diagram seems to be large enough that adding more would be unnecessary excess. During coding the state machine, I learned a lot on how system could have more cleaner design using this method of coding. Additionally learning how to do mock test was hard because I had to research and understand the given code. But in the end, it became simple that I was able to implement successful tests. This assignment thought me so much about dynamic designing and I look forward in designing using similar method.