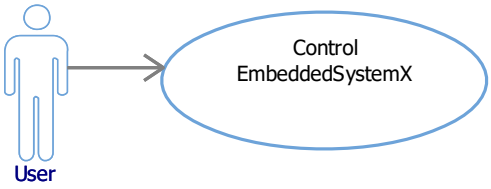
Embedded Real-time Systems
Exercise 1. Implementation of a Generic State Machine for an Embedded System

Description:
In this exercise you will implement a State Machine with the *GoF State Pattern* and the *GoF Singleton Pattern*

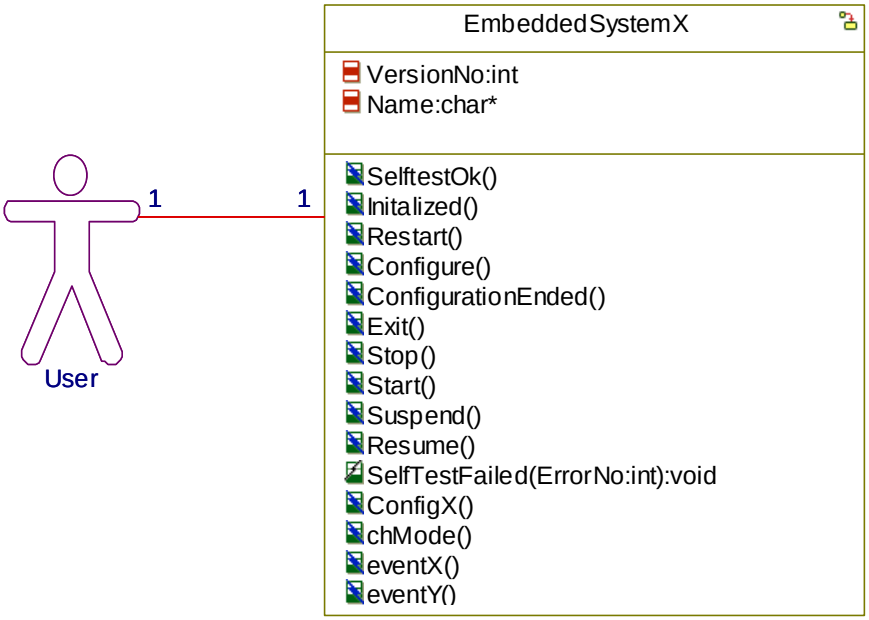
Goals:
When you have completed this exercise, you will
· Experience with implementing the GoF State pattern and active objects for a Hierarchical State Machine
· Experience with implementing command pattern and interconnect it to the state pattern
· have developed an implementation of a Generic State Machine for an Embedded System design
· Use of Visual studio or an alternative UML tool to document your design

Exercise 1:

Use Case Diagram:

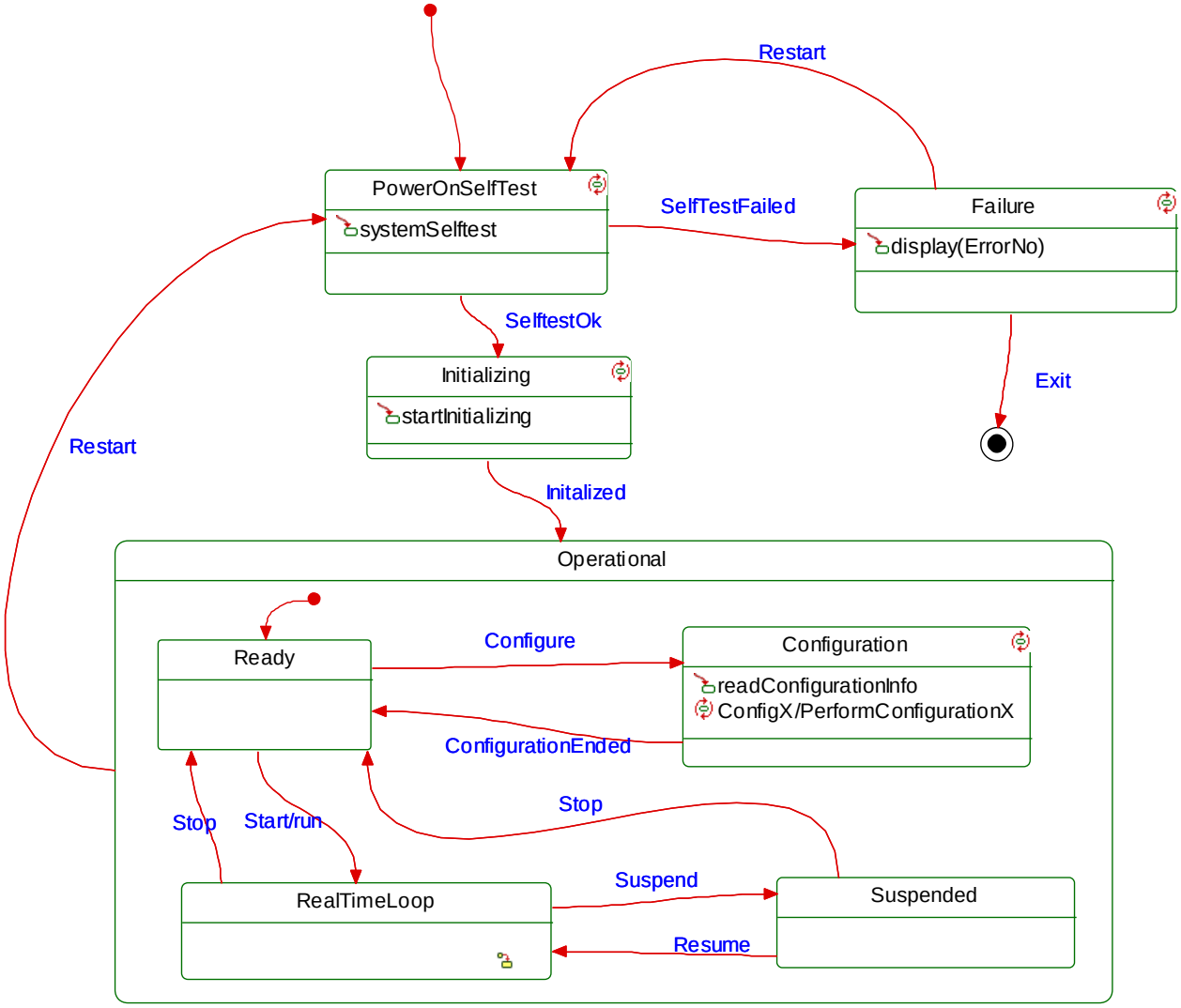


Class Diagram with event operations:



Class EmbeddedSystemX has the state machine shown on the following pages.

State Diagram of EmbeddedSystemX:



1.1. Design a solution for implementing this State Diagram with the GoF State Pattern, where each state is implemented with the Singleton Pattern.
Result: A class diagram.

1.2. Design, implement and test the design with an application implemented in C++. Consider the hardware model provided (**Platform-model**) to run the application on the **Zybo** board. An operating system snapshot is depicted in **Specifikation_af_OS_API** file.

1.2.1 Insert the class diagram for the solution in Visual studio or an alternative UML tool

1.2.2 Implement the Context Class (EmbeddedSystemX) with the shown event operations and add the necessary operations for implementing the State Pattern. Add a test operation for displaying the actual state.

1.2.3 Implement the classes for the State Pattern

1.2.4 Test the solution with a main program, where the user activates the public event operations and the actual state is displayed after each invocation.
1.3 Iterate you design where the events happened within state Operational are processed as commands. Use the command pattern to implement the processing of such events.
1.4 What is the main difference between the design alternatives (1.2.2; 1.3) in terms of performance and complexity.