Mastering Embedded System Online Diploma

www.learn-in-depth.com

First Term (Final Project 1)

Eng. Wessam Gamal Abdel Mohsen

[Pressure Controller]

My profile:

https://www.learn-in-depth.com/review-datamasteringdiploma/wessamgamal539%40gmail.com

Pressure Controller

An Embedded system is a controller, which controls many other electronic devices. It is a combination of embedded hardware and software. There are two types of embedded systems microprocessors and micro-controller. Micro-processor is based on von Neumann model/architecture (where program + data resides in the same memory location), it is an important part of the computer system, where external processors and peripherals are interfaced to it. The application of the microprocessor is personal computers.

A system designed with the embedding of hardware and software together for a specific function with a larger area is embedded system design. In embedded system design, a microcontroller plays a vital role. Micro-controller is based on Harvard architecture, it is an important component of an embedded system. External processor, internal memory and i/o components are interfaced with the microcontroller. It occupies less area, less power consumption.

The possible need for the embedded product may come from the manufacturer, or even customers, in how they think the larger product should work. Engineers will want to brainstorm how the embedded product could work and the benefits it would bring. They will also want to get a sense of the price customers or manufacturers will pay for the embedded product.

Design Sequence

1. Case Study

1.1 Specification

- A pressure controller informs the crew of a cabin with an alarm when the pressure exceeds 20 bars in the cabin.
- The alarm duration equals 60 seconds.
- Keeps track of the measured values.

1.2 Assumptions

- The controller set up and shutdown procedures are not modeled
- The controller maintenance is not modeled
- The pressure sensor never fails
- The controller never faces power cut

1.3 Versioning

- The ”keep track of measured value” option is not modeled in the first version of the design

2. Method

2.1 Difference between SDLC and STLC

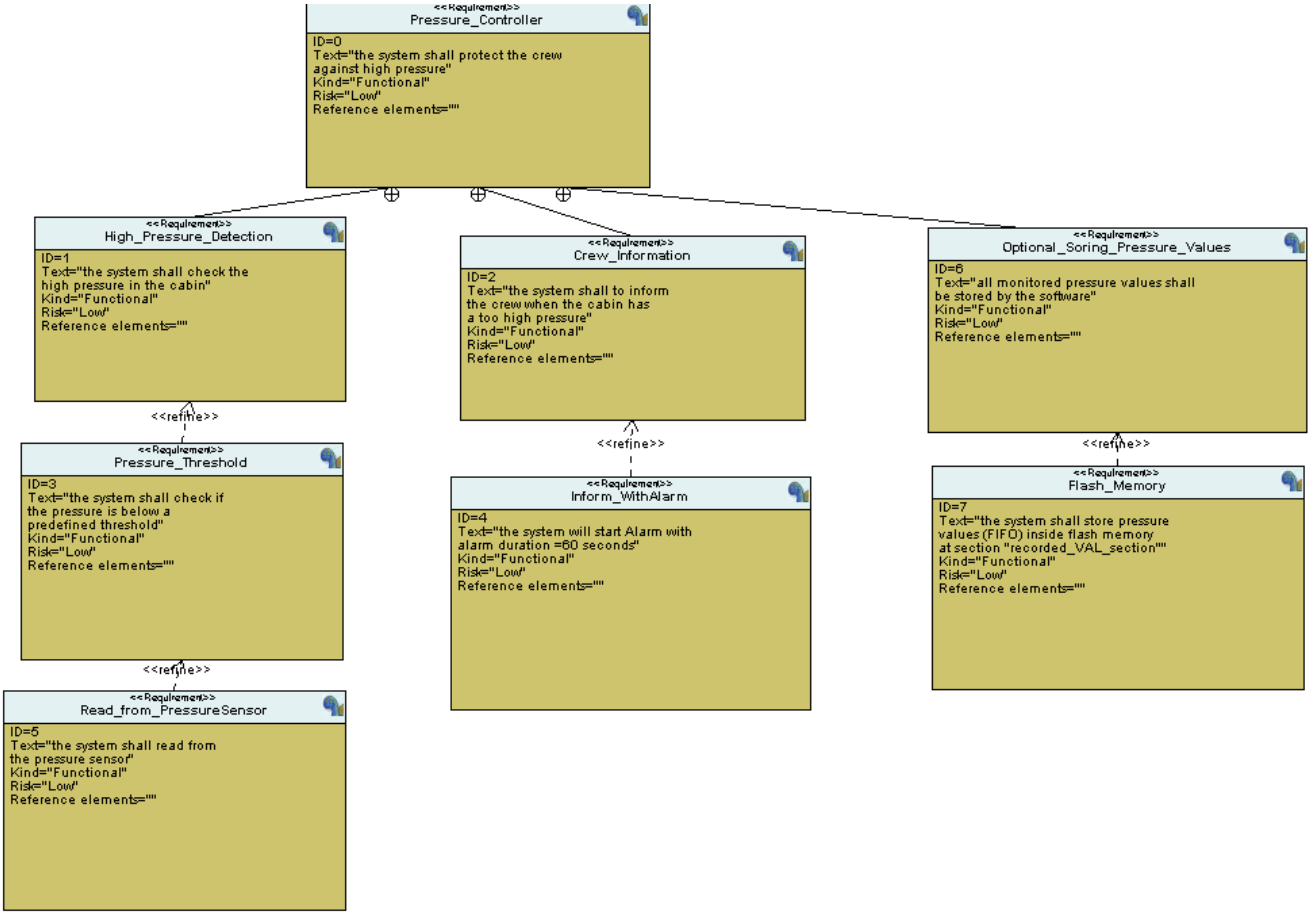
- SDLC is a Development Life Cycle, whereas STLC is a Testing Life Cycle.
- SDLC defines all the standard phases which are involved during the software development process, whereas the STLC process defines various activities to improve the quality of the product.
- In SDLC, the development team creates the high and low-level design plans, while In STLC, the test analyst creates the System, Integration Test Plan
- In SDLC, real code is developed, and actual work takes place as per the design documents, whereas in STLC testing team prepares the test environment and executes test cases.
- The SDLC life cycle helps a team complete the software's successful development, while the STLC phases only cover software testing.

2.2 Why Use SDLC?

- It aims to produce a high-quality software system which helps you to meet the customer expectations
- A formal review is created after completion of every stage that provides optimum management control.
- SDLC helps you to create considerable system documentation
- It produces many intermediate products which can be reviewed to verify whether they can meet the user's needs and are according to the stated requirement.
- SDLC helps you to ensures that system requirements can be traced back to stated business requirements
- Every phase has a specific deliverable, entry and exit criteria
- Development stages go one by one which is an ideal option for the small or mid-sized projects where requirements are clear

3. Requirements Analysis

- In systems engineering and software engineering, requirements analysis focuses on the tasks that determine the needs or conditions to meet the new or altered product or project, taking account of the possibly conflicting requirements of the various stakeholders, analyzing, documenting, validating and managing software or system requirements.
- Requirements analysis is critical to the success or failure of a systems or software project. The requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.



4. Space Exploration / Partitioning

One of the most crucial steps in the design of embedded systems is hardware-software partitioning, it is the division of an application's computations into a part that executes as sequential instructions on a microprocessor (the "software") and a part that runs as parallel circuits on some IC fabric like an ASIC or FPGA (the "hardware"), to achieve design goals set for metrics like performance, power, size, and cost. The circuit part commonly acts as a coprocessor for the microprocessor.

HW/SW partitioning is an important development step during HW/SW co-design to ensure application performance in embedded System-on-Chip (SoC). The partitioning is done in the earliest stages of the design; at the stage where there is the greatest possibility for changes. The hardware software partitioning tries to exploit the synergy of hardware and software.

The classic HW/SW partitioning process includes the system that was immediately partitioned into hardware and software components. The hardware and software are developed separately. The "Hardware First" approach is often adopted. Some of the implications of these features are that the HW/SW trade-offs are restricted. The impact of HW and SW on each other cannot be assessed easily. Consequences of adopting these classic techniques are-poor quality designs, costly modifications and time to market increased.

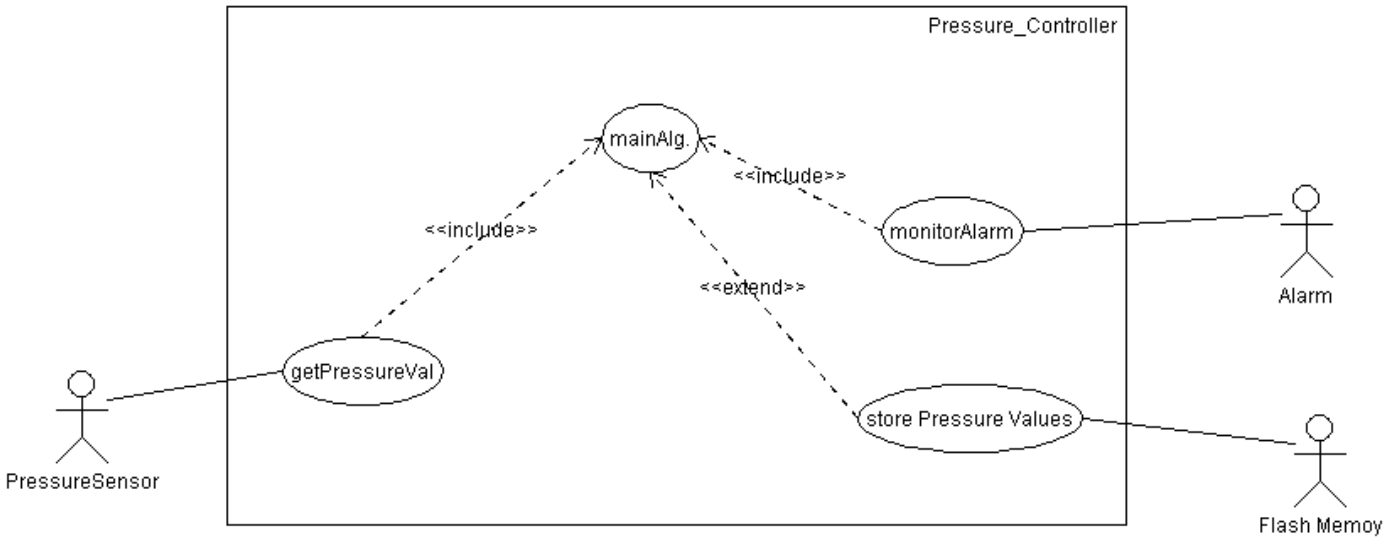
Partitioning has dramatic impact on the cost and performance of the whole system.

The HW/SW partitioning improves overall system performance, reliability, and cost effectiveness because defects found in hardware can be corrected before tape-out. Partitioning benefits the design of embedded systems and SoCs, which need the HW/SW tailored for a particular application.

5. System Analysis

5.1 Use Case Diagram

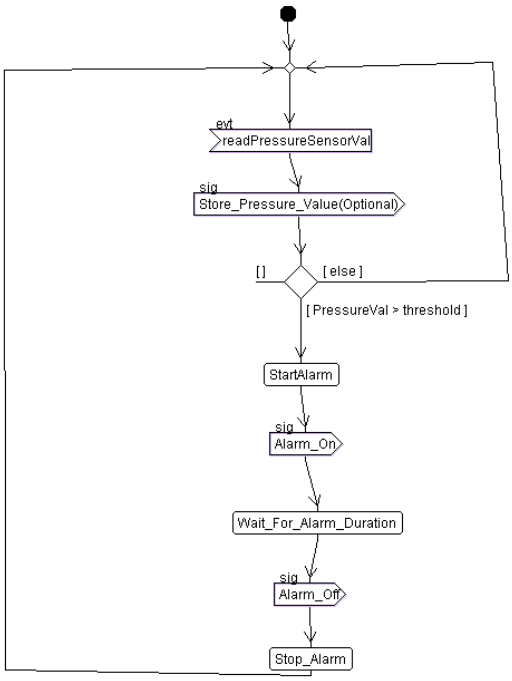
In the use case modeling approach, functional requirements are defined in terms of actors, which are external to the system, and use cases. A use case defines a sequence of interactions between one or more actors and the system. The use case model describes the functional requirements of the system in terms of the actors and use cases. In particular, the use case model considers the system as a black box and describes the interactions between the actor(s) and the system in a narrative textual form consisting of actor inputs and system responses.



5.2 Activity Diagram

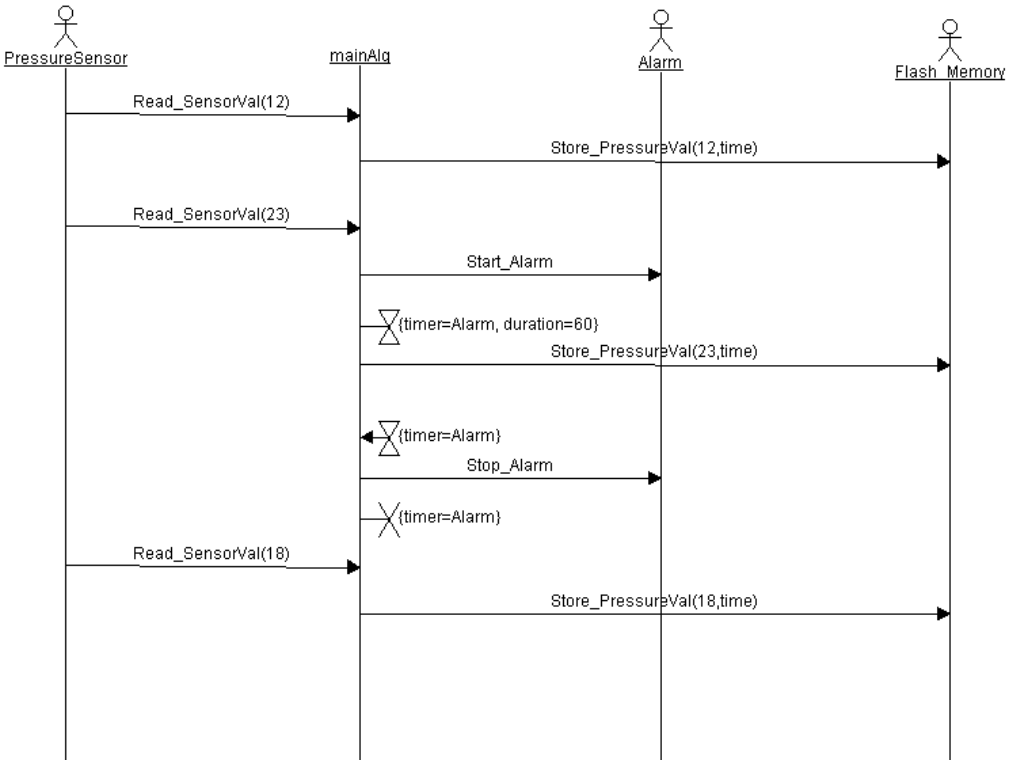
Activity Diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The basic purpose of activity diagrams is to capture the dynamic behavior of the system.. It is also called object-oriented flowchart.

This UML diagram focuses on the execution and flow of the behavior of a system instead of implementation. Activity diagrams consist of activities that are made up of actions that apply to behavioral modeling technology.



5.3 Sequence Diagram

A sequence diagram is a Unified Modeling Language (UML) diagram that illustrates the sequence of messages between objects in an interaction.
A sequence diagram consists of a group of objects that are represented by lifelines, and the messages that they exchange over time during the interaction.
A sequence diagram shows the sequence of messages passed between objects.
Sequence diagrams can also show the control structures between objects.



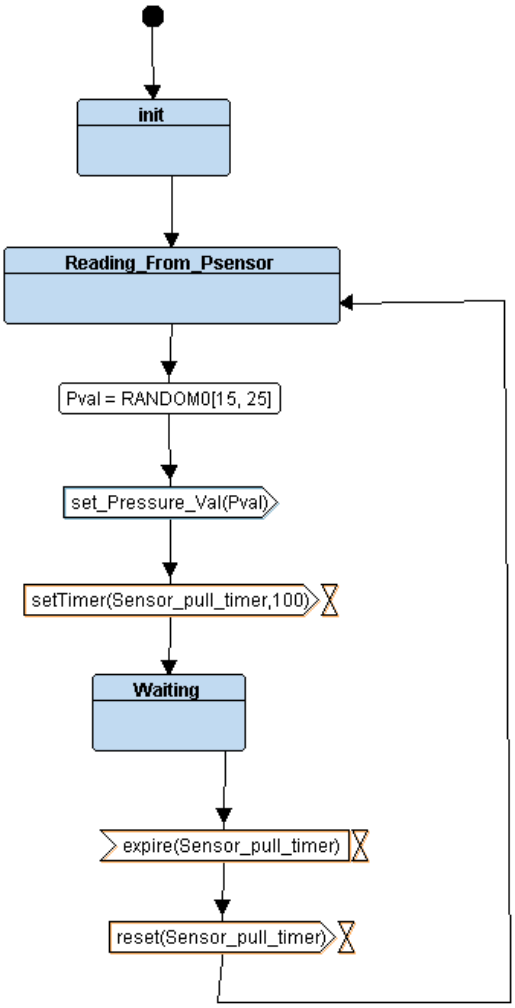
6. System Design

State Machine Diagram

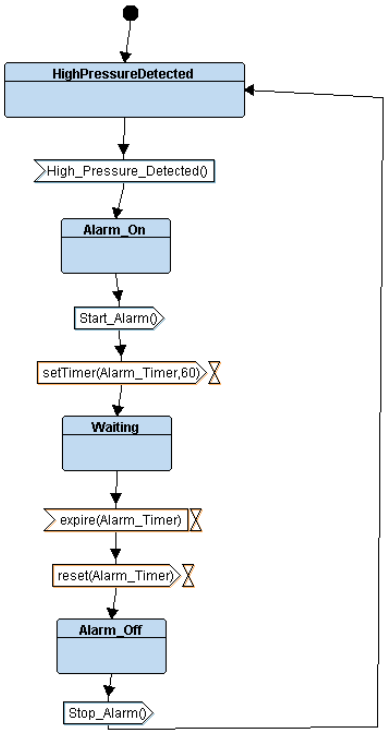
A state machine is a behavior model. It consists of a finite number of states and is therefore also called finite-state machine (FSM). Based on the current state and a given input the machine performs state transitions and produces outputs. There are basic types like Mealy and Moore machines and more complex types like Harel and UML statecharts.

The basic building blocks of a state machine are states and transitions. A state is a situation of a system depending on previous inputs and causes a reaction on following inputs. One state is marked as the initial state; this is where the execution of the machine starts. A state transition defines for which input a state is changed from one to another. Depending on the state machine type, states and/or transitions produce outputs.

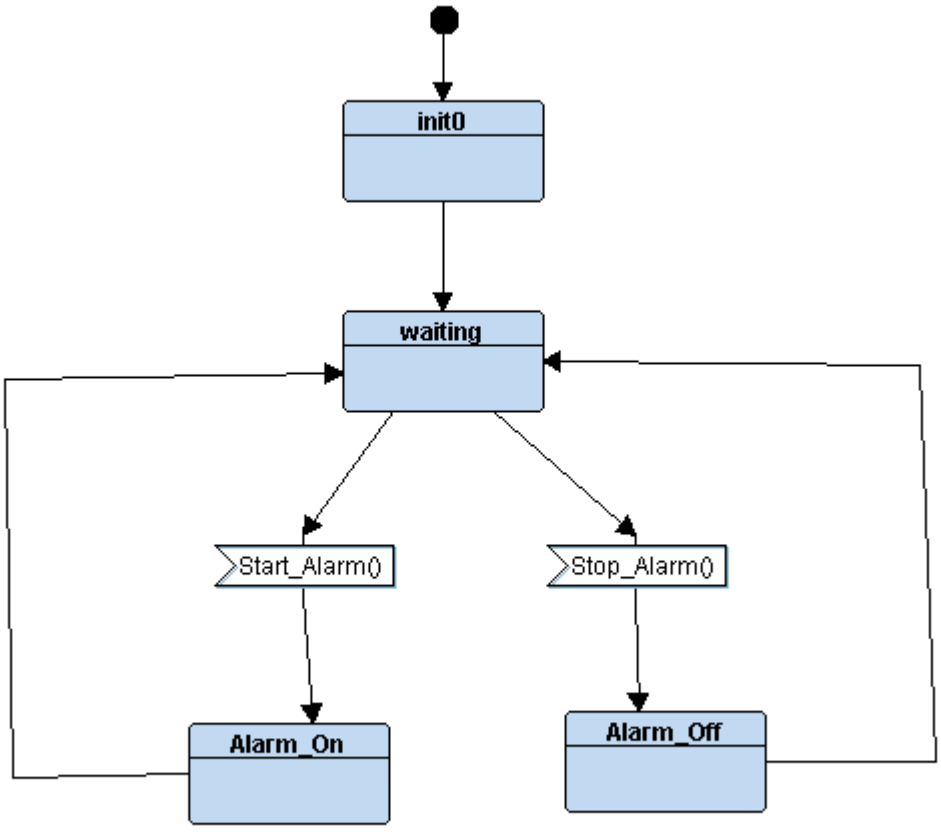
Pressure Sensor Driver



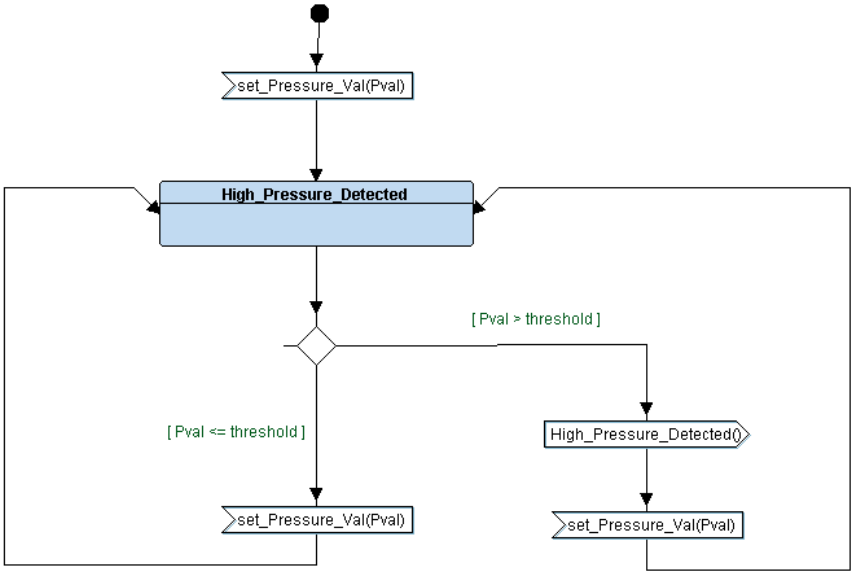
Alarm Manager



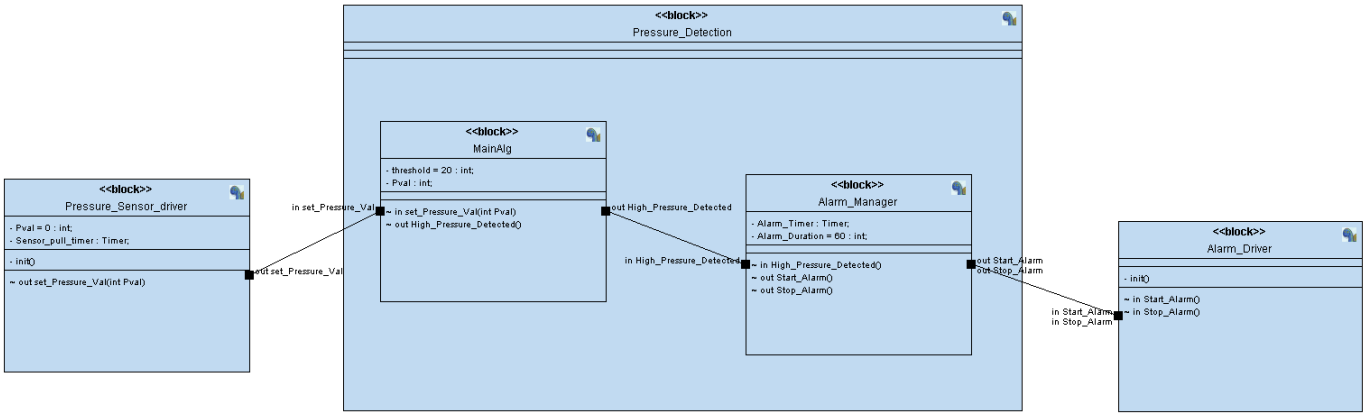
Alarm Driver



Main Algorithm

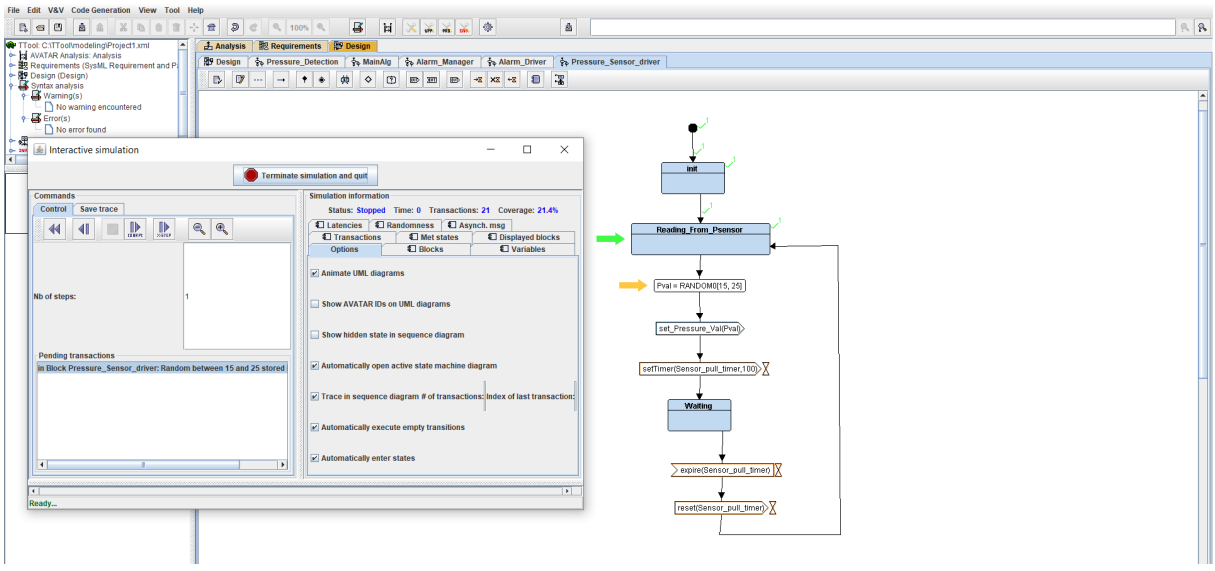


Block Diagram

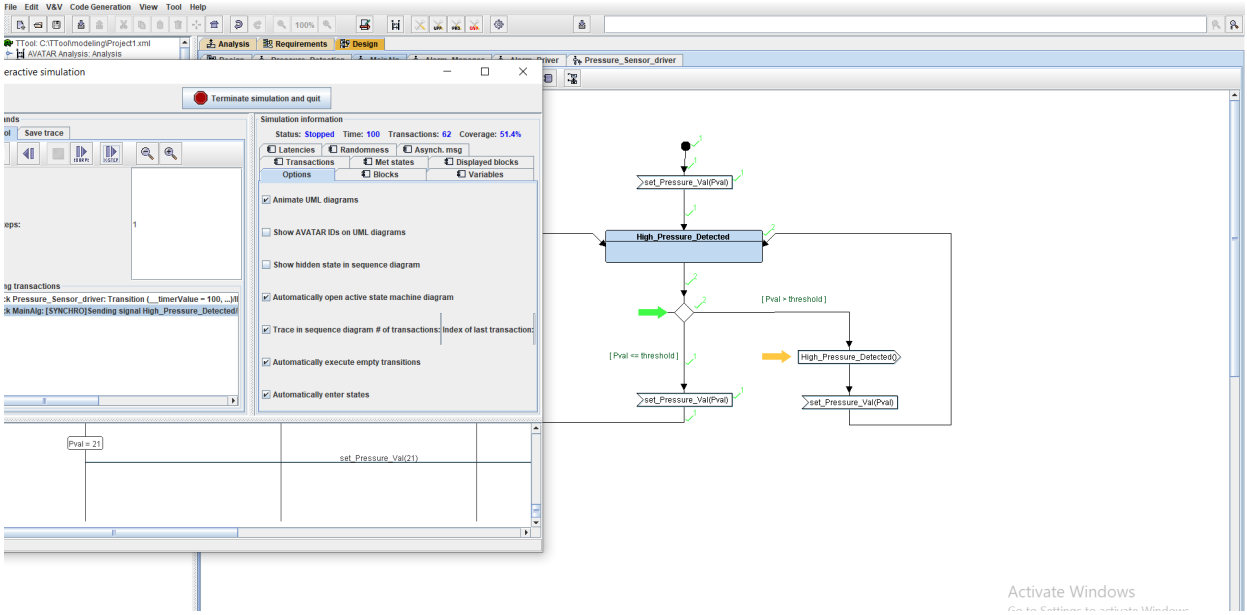


Simulation

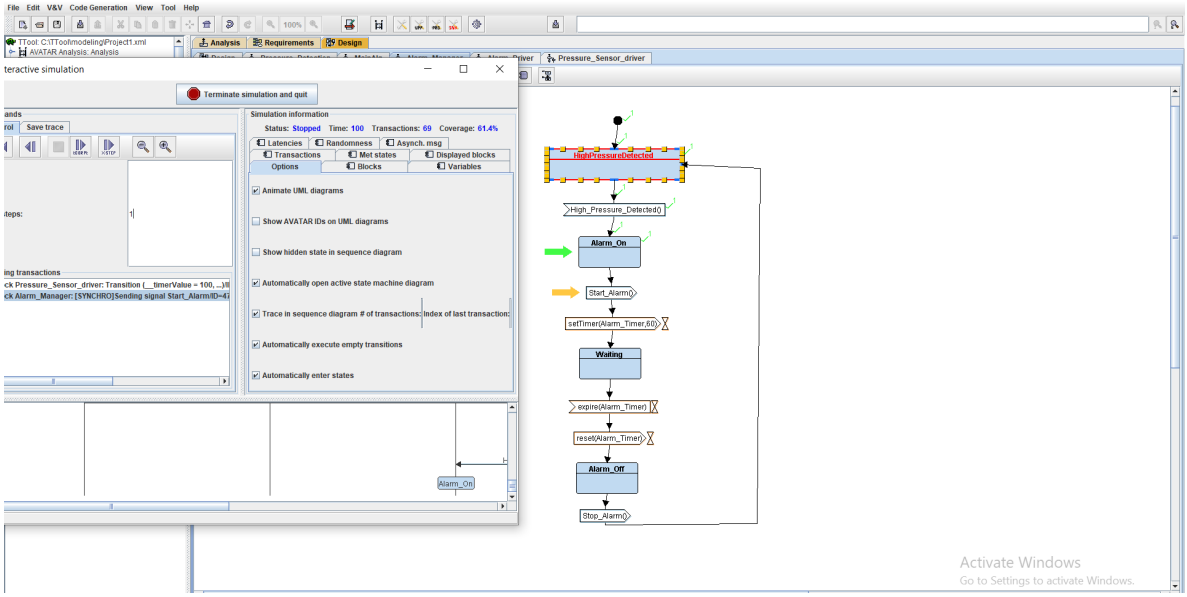
1-Reading From Sensor



2- High Pressure Detected



3-Alarm On



4-Alarm Off