**Mastering Embedded System Online Diploma**

[www.learn-in-depth.com](http://www.learn-in-depth.com)

First Term (Final Project 1)

**Pressure Detection System Report**

Eng. Mohamed Hazem Yahya Mahrous Ali

Contents:

1. [Introduction……………………………………………………….(3)](#intro)
2. [System Overview………………………………………………..(3)](#sys_over)
3. [System Analysis………………………………………………....(4)](#sys_analysis)
4. [Hardware/Software Partitioning…………………………(9)](#hw_sw_Part)
5. [Code Implementation……………………………………….(11)](#code)
6. [Output Program………………………………………………..(20)](#output)
7. [Conclusion…………………………………………………………(23)](#conc)

Introduction

This report presents an analysis of the Pressure Detection System, including its code implementation and a breakdown of the hardware/software partitioning. The Pressure Detection System is designed to monitor pressure values and activate an alarm when the pressure exceeds a certain threshold.

System Overview

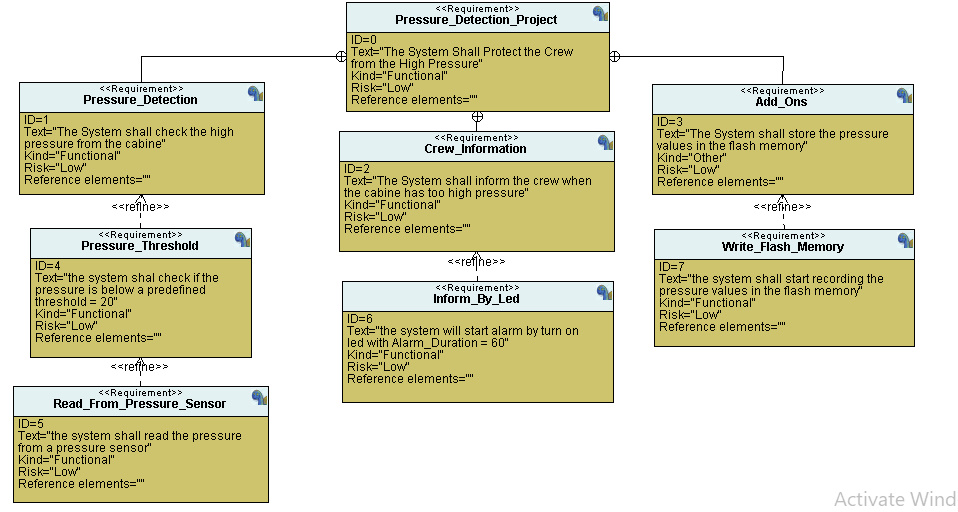
The Pressure Detection System consists of multiple components that interact to achieve its functionality:

1. **Pressure Sensor Driver**: Responsible for reading pressure values from a sensor.
2. **Alarm Monitor**: Monitors pressure values and triggers the alarm if needed.
3. **Alarm Actuator Driver**: Controls the alarm actuator, turning it on or off.

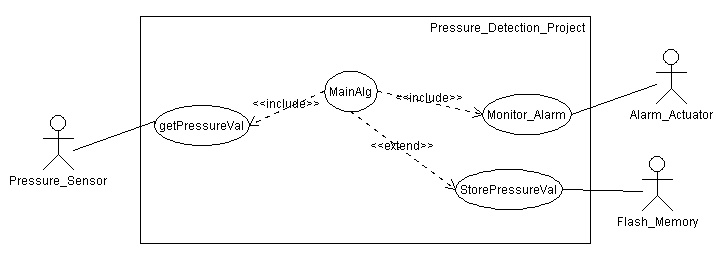
The main goal of the system is to detect pressure values and respond accordingly by triggering the alarm when the pressure exceeds a predefined threshold.

System Analysis

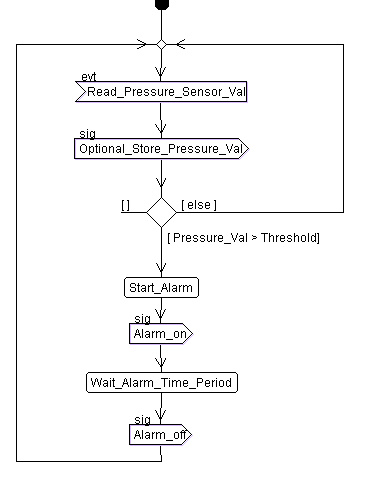
1. Requirements



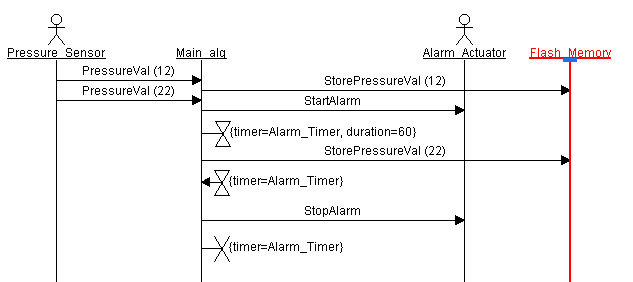
1. Analysis
   1. Use Case Diagram



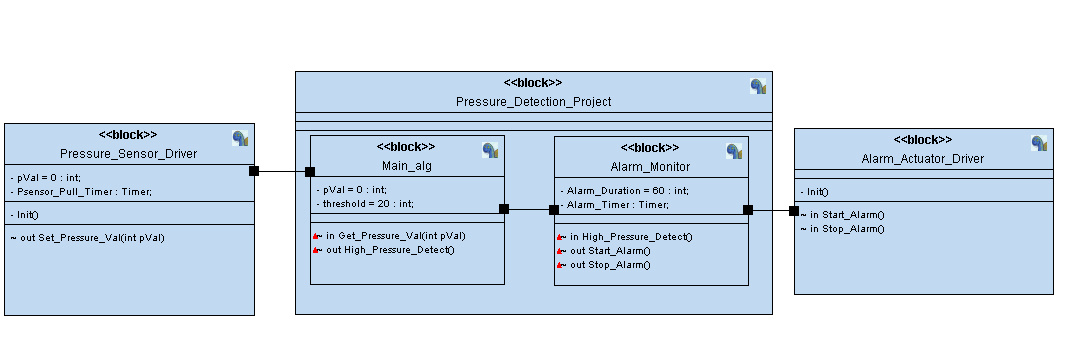
* 1. Activity Diagram



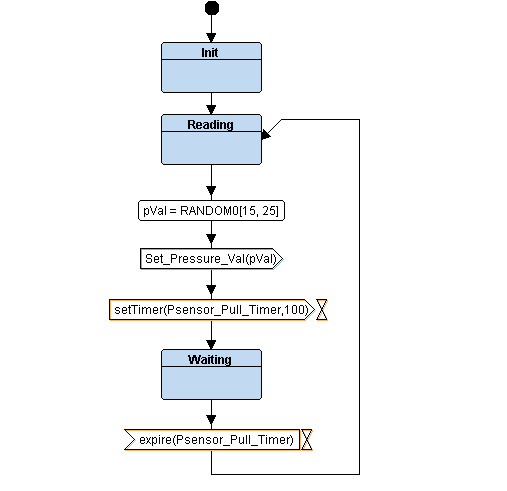
* 1. Sequence Diagram



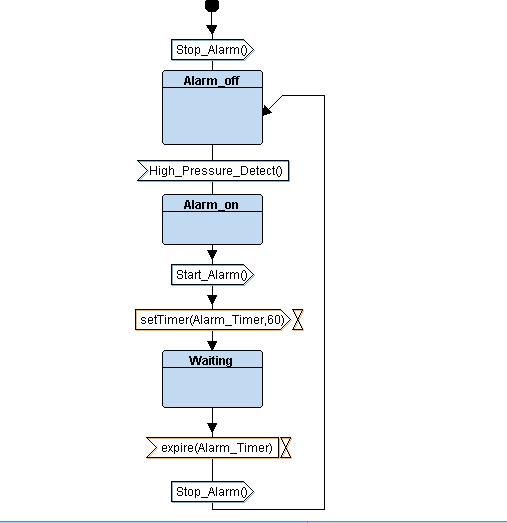
1. Design
   1. Block Diagram



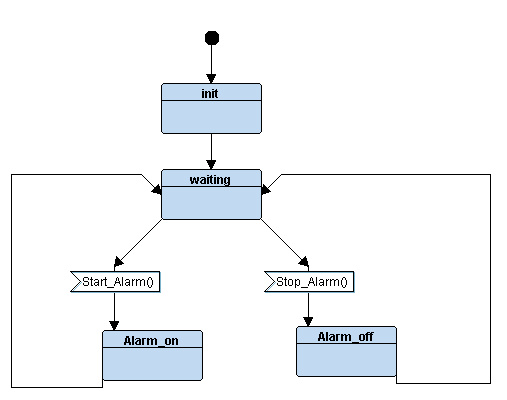
* 1. State Machine
     1. Pressure Sensor Driver



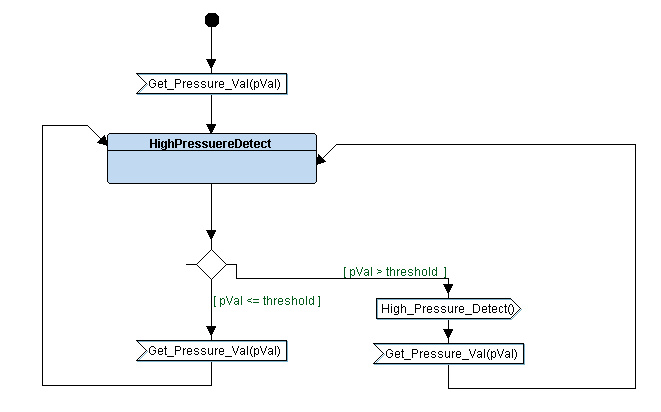
* + 1. Alarm Monitor



* + 1. Alarm Actuator Driver



* + 1. Main Program



**Hardware/Software Partitioning**

**Hardware Components:**

1. **Microcontroller Unit (MCU)**:
   * The central hardware component that houses the microprocessor, memory, GPIO ports, and other peripherals.
   * Responsible for executing the software code and interacting with hardware components.
2. **Pressure Sensor**:
   * A hardware component responsible for measuring pressure values.
   * Connects to the microcontroller via GPIO pins to transmit pressure readings.
   * Purely hardware; pressure values are read from the sensor using hardware interactions.
3. **Alarm Actuator**:
   * A hardware component that generates alarm signals (e.g., sound, light) when activated.
   * Controlled by the microcontroller through GPIO pins to turn the alarm on/off.
   * Purely hardware; alarm activation is triggered by hardware interactions.
4. **GPIO Pins**:
   * General-purpose input/output pins on the microcontroller.
   * Used for digital communication between the microcontroller and external components (sensor and actuator).
   * Hardware component, but their manipulation is facilitated by software.

**Software Components:**

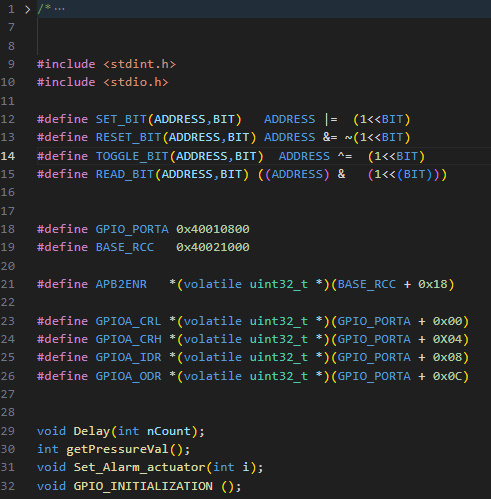
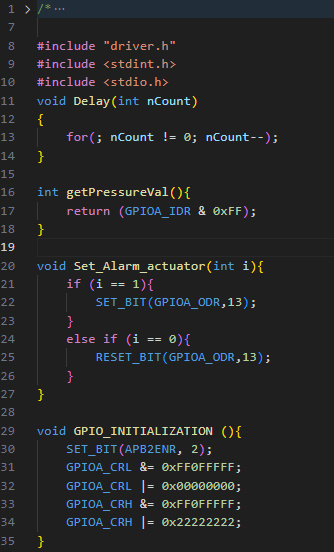
1. **Microcontroller Firmware**:
   * Written in C, the firmware is the main software code running on the microcontroller.
   * Contains the logic for system initialization, state transitions, and interaction with hardware components.
   * Executes the main loop that reads pressure values, monitors alarms, and controls the alarm actuator.
2. **State Management**:
   * A collection of state functions (pSensor\_state, AM\_state, AA\_state) that define the system's behavior.
   * Each state function encapsulates a specific system state and its associated actions.
   * Facilitates state transitions and ensures that the system responds appropriately to pressure changes.
3. **Driver Functions**:
   * Software functions that provide an abstraction layer to interact with hardware registers and GPIO pins.
   * Enable communication with the pressure sensor (reading pressure values) and the alarm actuator (activating/deactivating the alarm).
   * Abstract the low-level hardware interactions, making it easier to control the hardware components.
4. **Main Program**:
   * The main loop of the firmware, which continuously iterates through state functions.
   * Orchestrates the execution of state functions, pressure reading, alarm monitoring, and actuator control.
   * Reads pressure values from the sensor and triggers the alarm actuator based on system conditions.

**Code Implementation**

The system's functionality is implemented in the provided code. The following sections describe the main components and their roles.

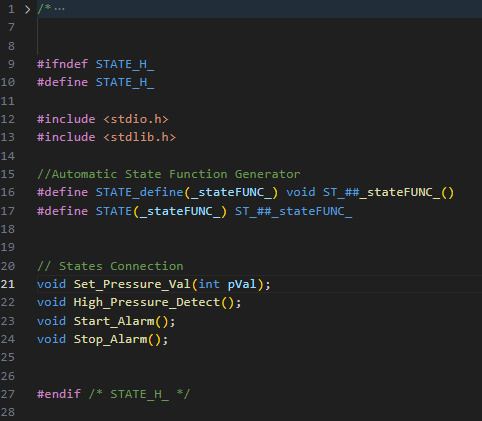
**1. Driver Layer (driver.c, driver.h)**

The driver layer provides low-level functions for hardware manipulation. It includes functions for setting and resetting individual bits in registers, reading pressure values from the GPIO port, and controlling the alarm actuator.



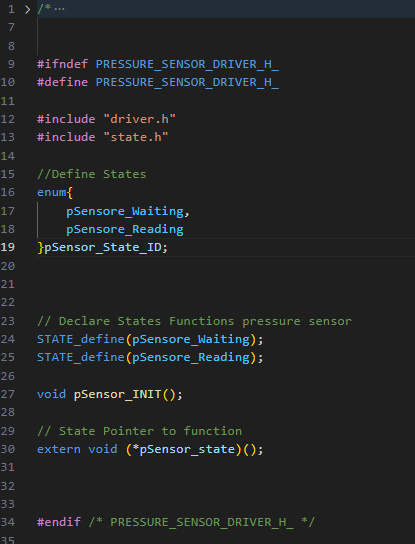
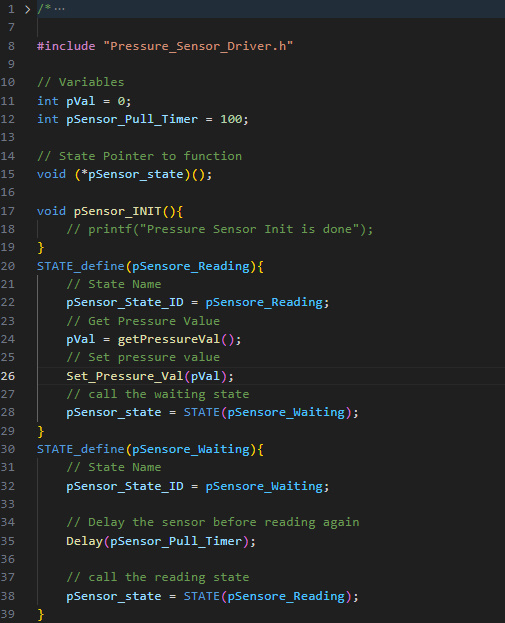
**2. State Layer (state.h)**

The state layer defines the different states of the system. It provides a mechanism to generate state functions automatically and defines state transition functions. In this system, there are states for reading pressure, waiting, and managing the alarm.



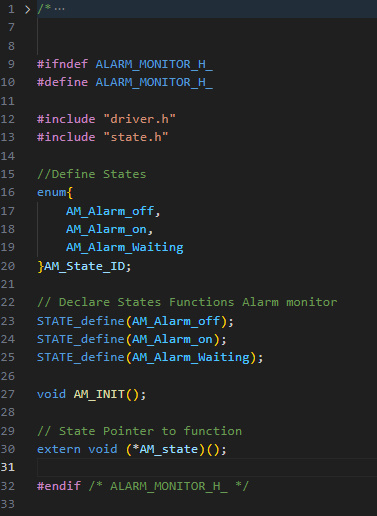
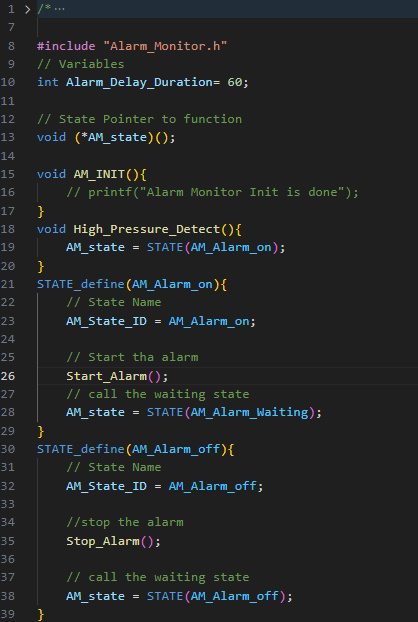
**3. Pressure Sensor Driver (Pressure\_Sensor\_Driver.c, Pressure\_Sensor\_Driver.h)**

This component is responsible for reading pressure values from the sensor. It defines two states: "Reading" and "Waiting." The system alternates between these states to periodically read pressure values. If the pressure exceeds a threshold, it triggers the alarm.



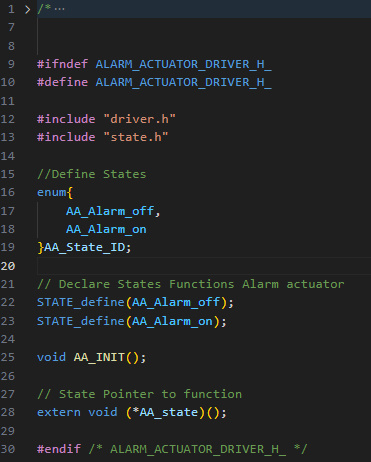
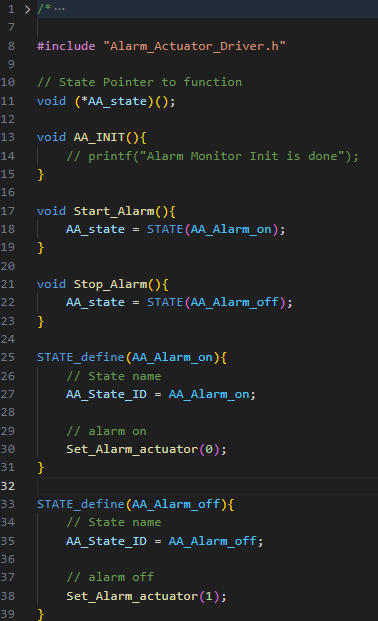
**4. Alarm Monitor (Alarm\_Monitor.c, Alarm\_Monitor.h)**

The Alarm Monitor oversees the pressure values and responds by activating or deactivating the alarm as needed. It defines states for the alarm being on, off, and waiting. The system transitions between these states based on pressure conditions.



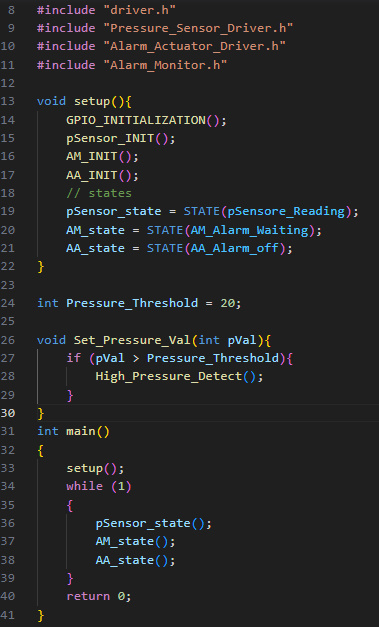
**5. Alarm Actuator Driver (Alarm\_Actuator\_Driver.c, Alarm\_Actuator\_Driver.h)**

This component controls the alarm actuator. It defines states for the alarm being on and off. The alarm actuator is activated or deactivated based on the system's state.



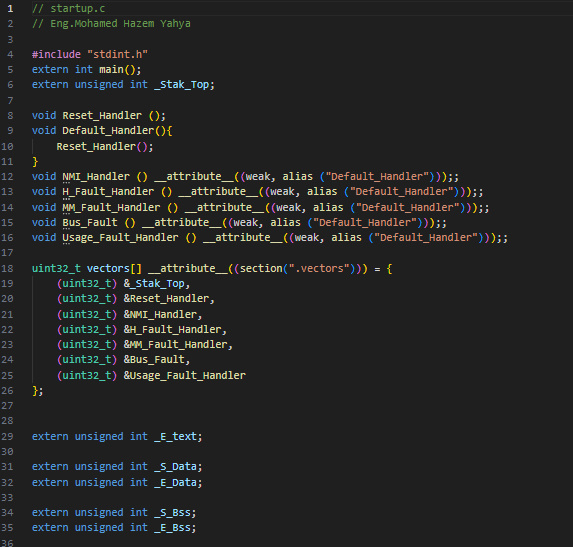
**6. Main Program (main.c)**

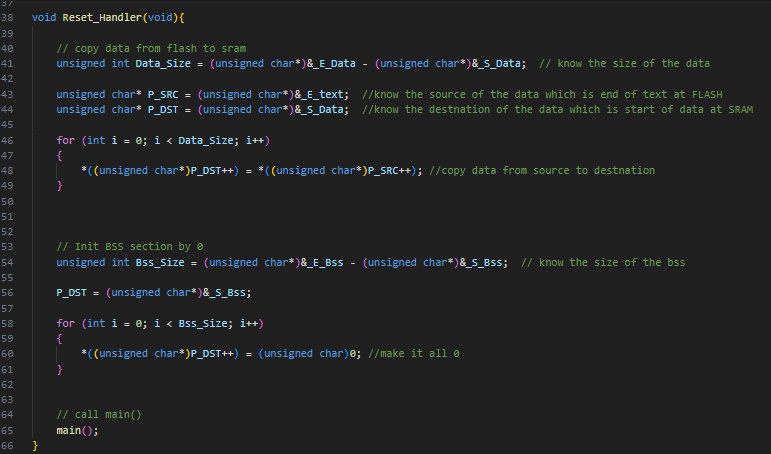
The main program initializes the system components, sets up the initial states, and enters a loop where it continuously reads pressure values, monitors alarms, and controls the alarm actuator.



**7. Startup Code (startup.c)**

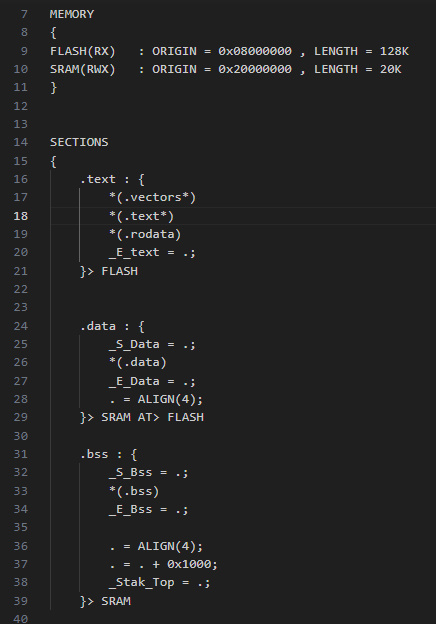
Startup code initializes the microcontroller's hardware, sets up the initial stack and heap, and prepares the environment for running the firmware.

****

****

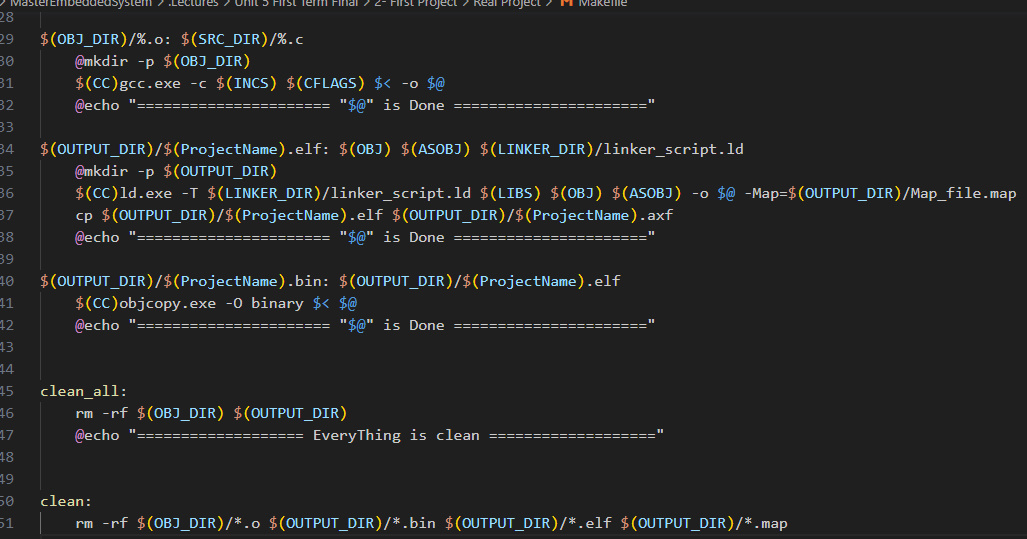
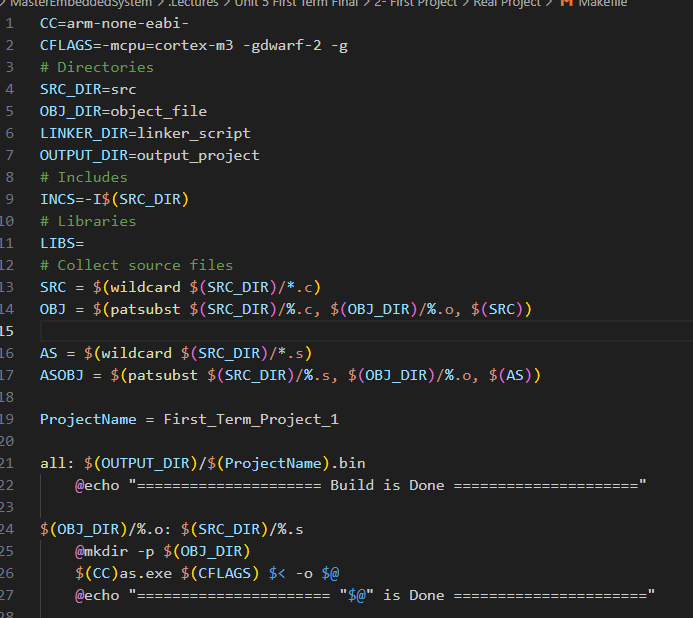
**8. Linker Script (linkerscript.ld)**

The linker script defines the memory layout of the microcontroller, specifying where the code, data, stack, and other sections are located in memory.

****

**9. Makefile**

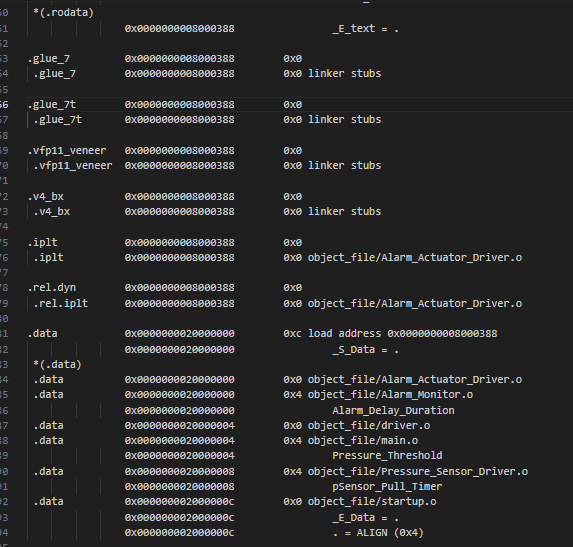
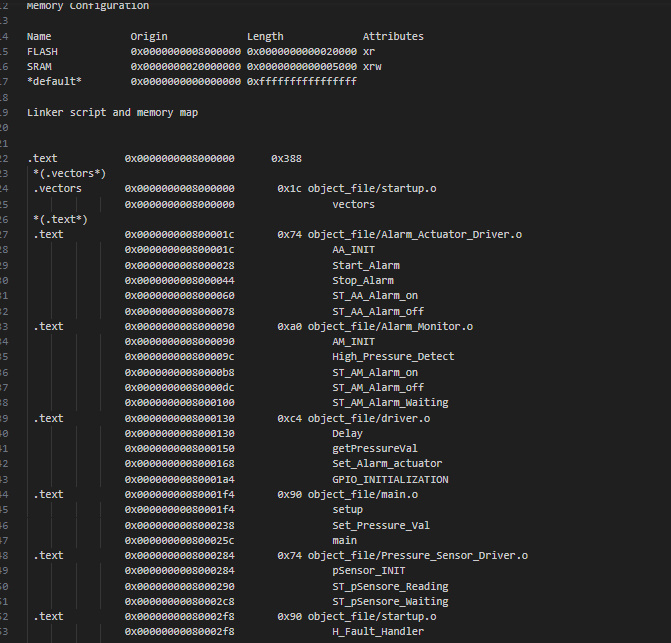
The Makefile automates the build process, compiling source files, linking them together, and generating the final binary file that can be flashed onto the microcontroller.

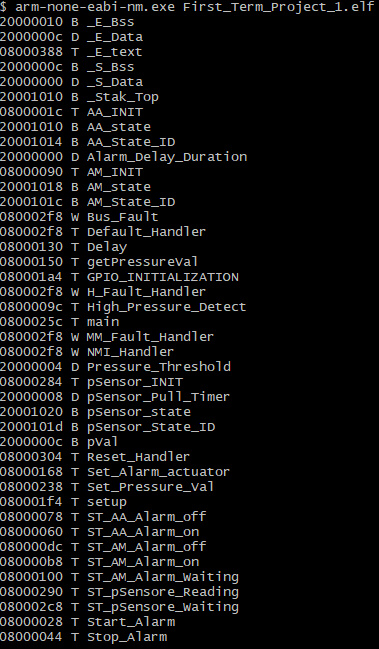
****

**Output Program**

**1. Memory Map and Symbol Table**

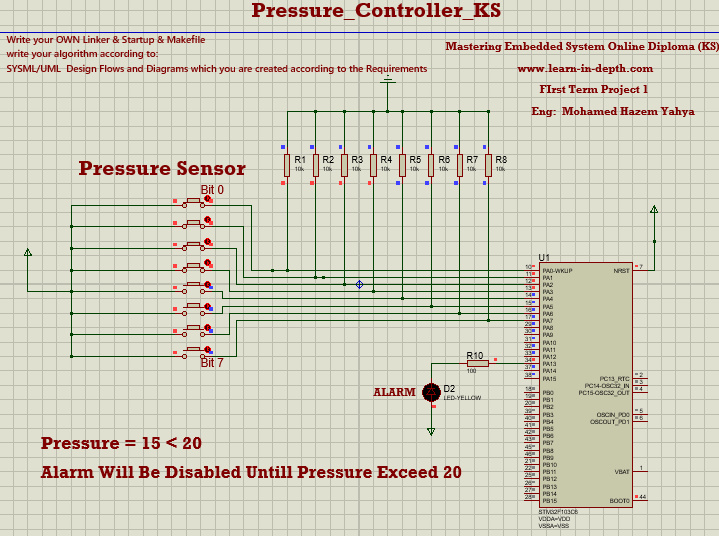
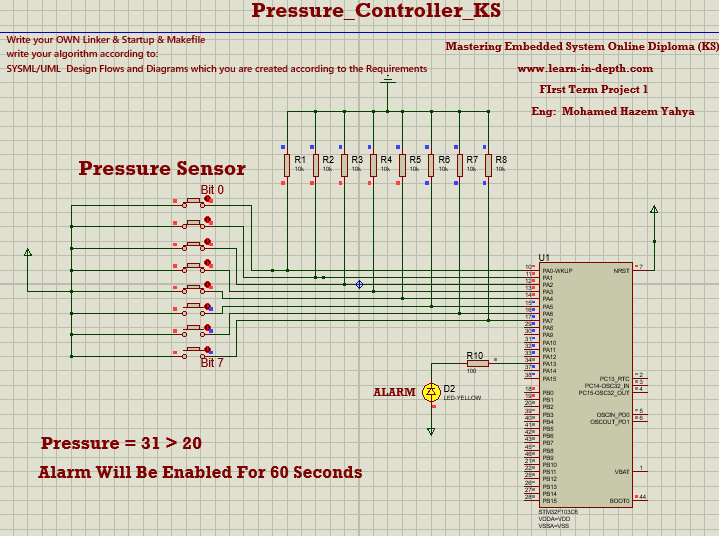
The memory map (map file) and symbol table provide insights into how the program is organized in memory and the addresses of various functions and variables.

****

****

**2. Simulation**

Simulations allow you to observe how the program executes step by step, aiding in debugging and understanding the program's behavior.

****

**Conclusion**

The hardware/software partitioning clearly defines the roles of hardware components and software code in the Pressure Detection System. The microcontroller firmware, state management, and driver functions collaborate to create a functional and responsive system that reads pressure values, monitors alarms, and controls the alarm actuator. This partitioning ensures a clear separation of concerns, enabling efficient development, debugging, and maintenance of the system.