

LEARN-IN-DEPTH

Pressure Control

Mostafa Ahmed Hesham Hammad Ahmed
9-27-2023

Contents

Introduction	2
System Architecting/Design Sequence	2
Customer Requirements	2
Method	2
Requirements.....	2
System Partitioning.....	3
System Analysis.....	3
Use Case Diagram	3
Activity Diagram	3
Sequence Diagram	4
System Design	4
Pressure Sensor.....	5
Main Algorithm	6
Alarm Monitor.....	7
Alarm Actuator.....	8
SW Parameters	9
Map File.....	9
Sections	9
Symbols	9
Code Appendix	10
State.h	10
PressureSensor_Driver.h	10
PressureSensor_Driver.c.....	10
algorithm.h.....	11
algorithm.c	11
AlarmMonitor.h	12
AlarmMonitor.c.....	12
AlarmActuator.h.....	13
AlarmActuator.c.....	13
main.c.....	14

Introduction

The project aims to design and implement a state machine-based system to monitor and respond to high-pressure events using a Pressure Sensor, Algorithm, Alarm Monitor, and Alarm Actuator. This report outlines the architecture, design, and functionality of the state machine-based system.

System Architecting/Design Sequence

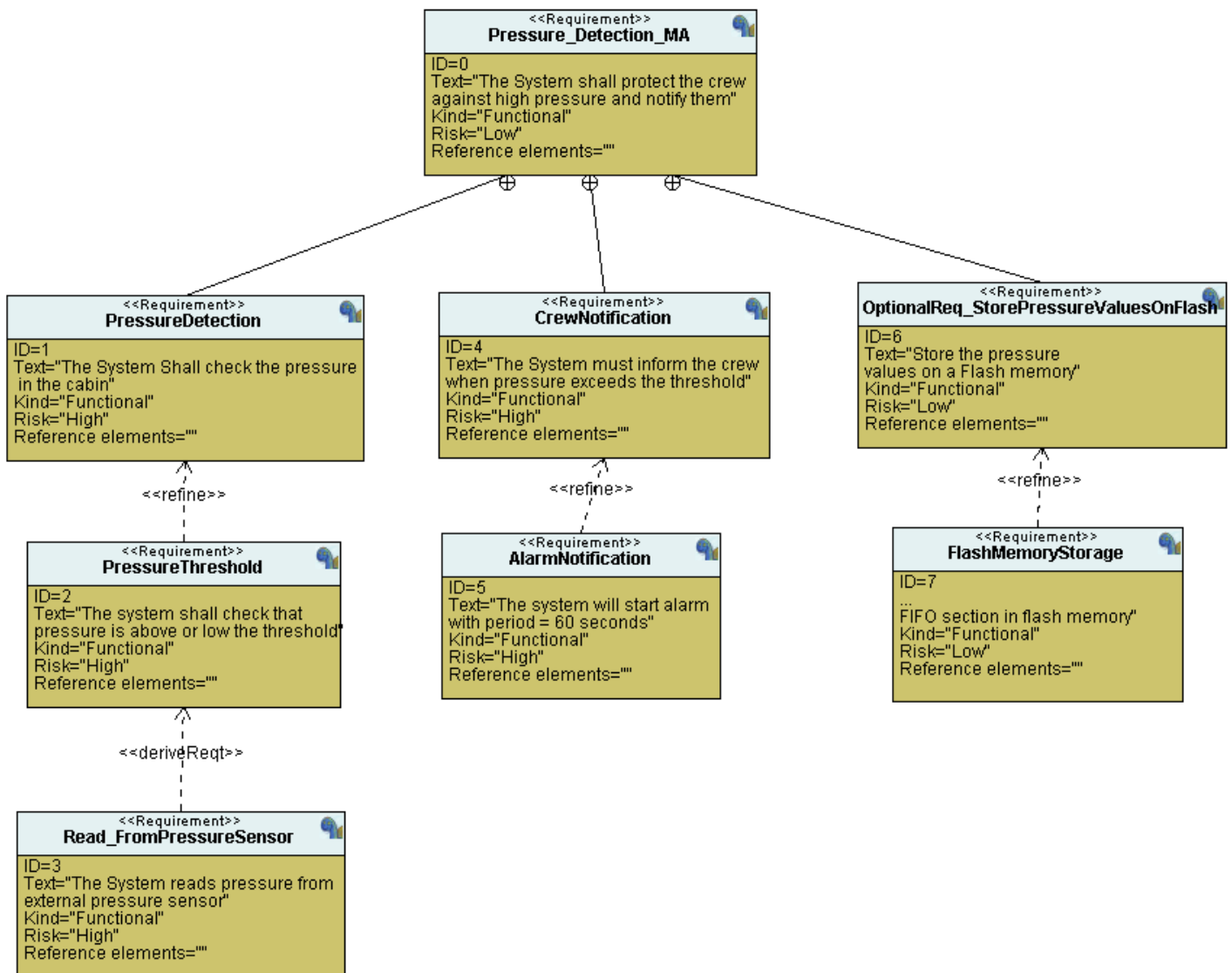
Customer Requirements

- 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.

Method

V-Model Method

Requirements

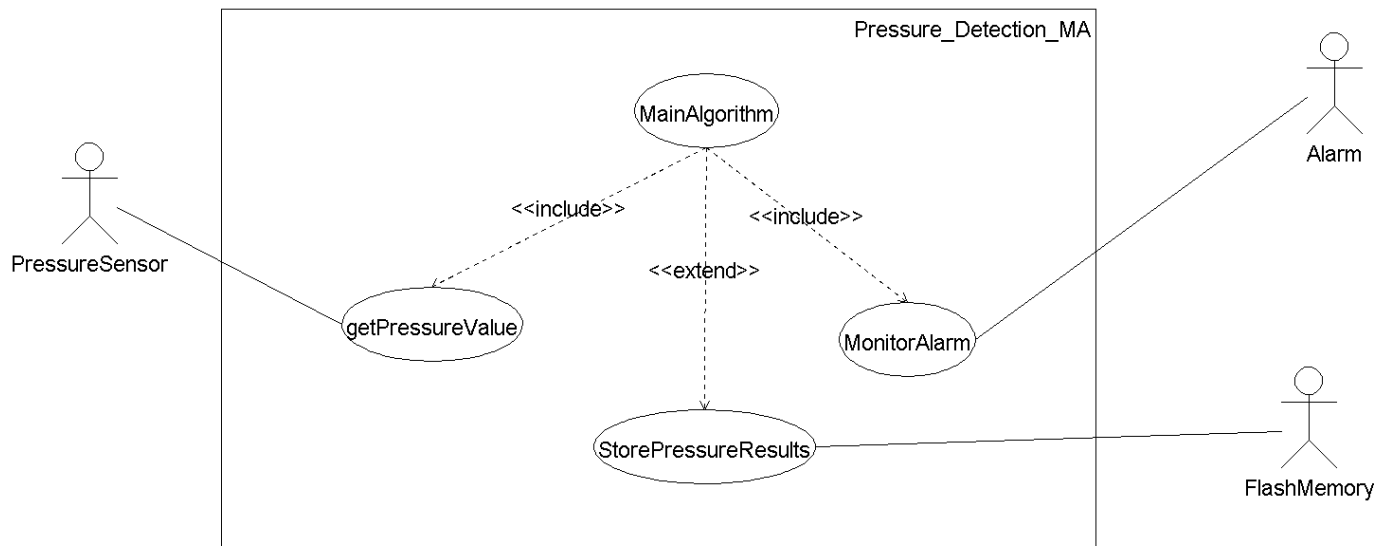


System Partitioning

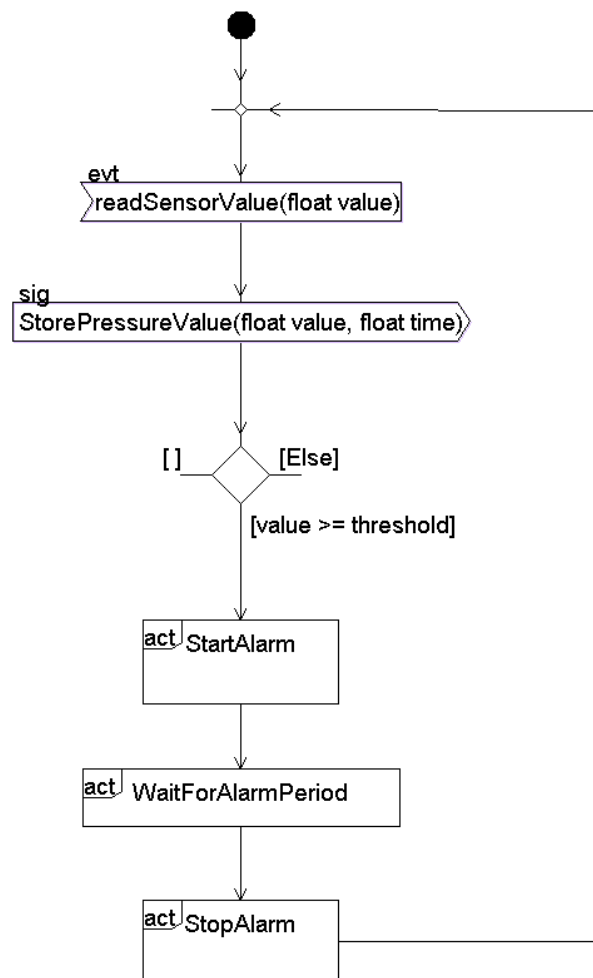
After doing a several analysis the chosen MCU which will run all the system modules will be [stm32f103c8t6](#)

System Analysis

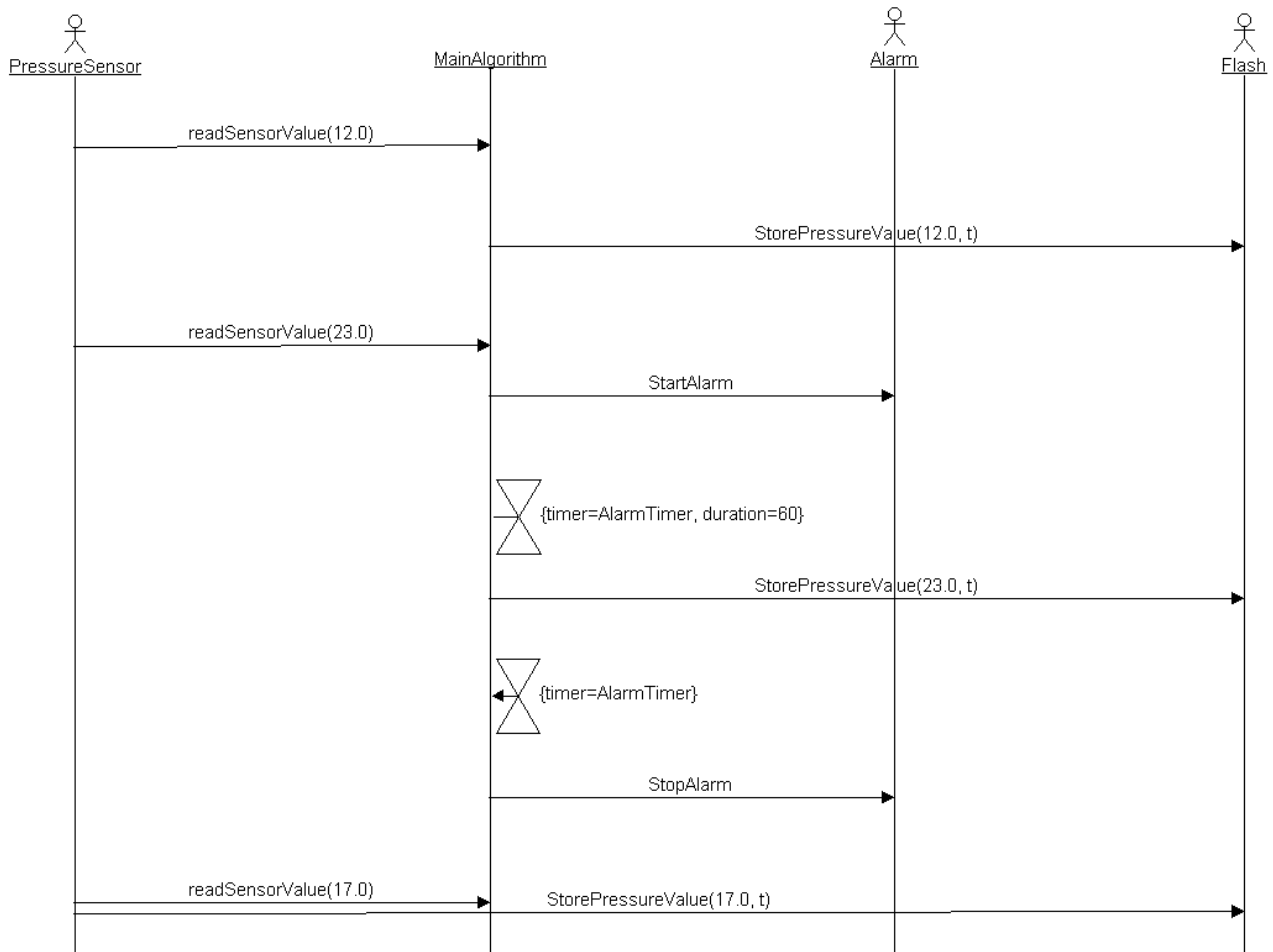
Use Case Diagram



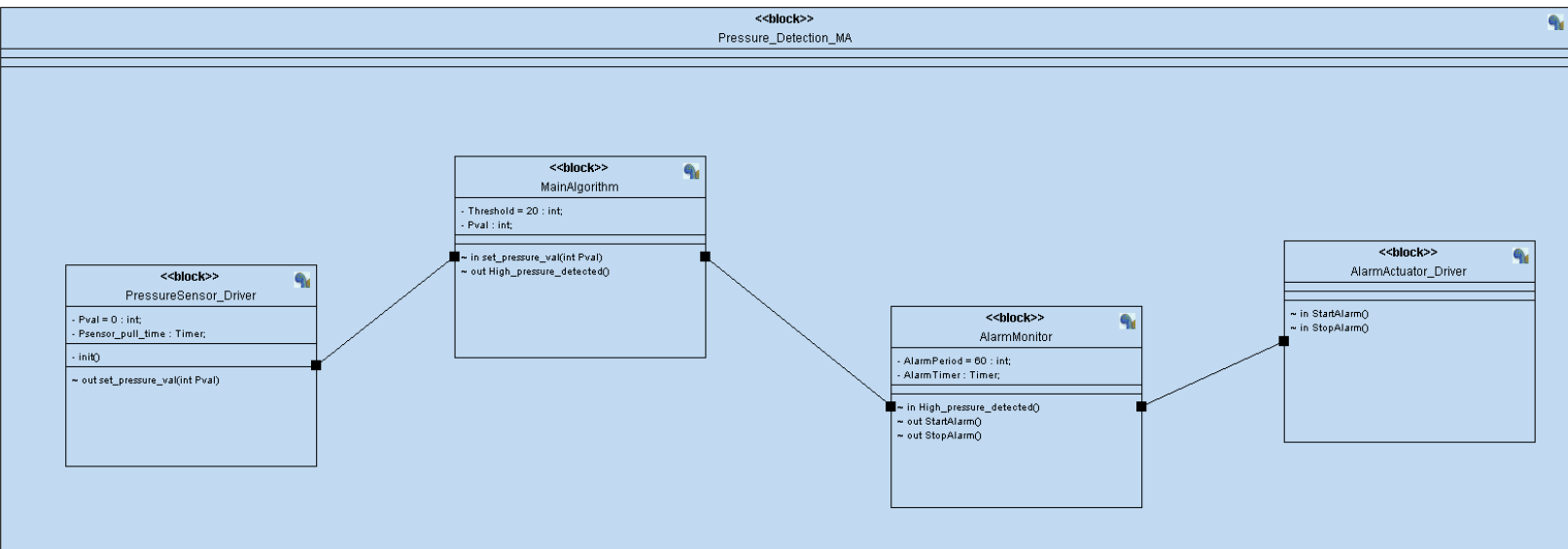
Activity Diagram



Sequence Diagram

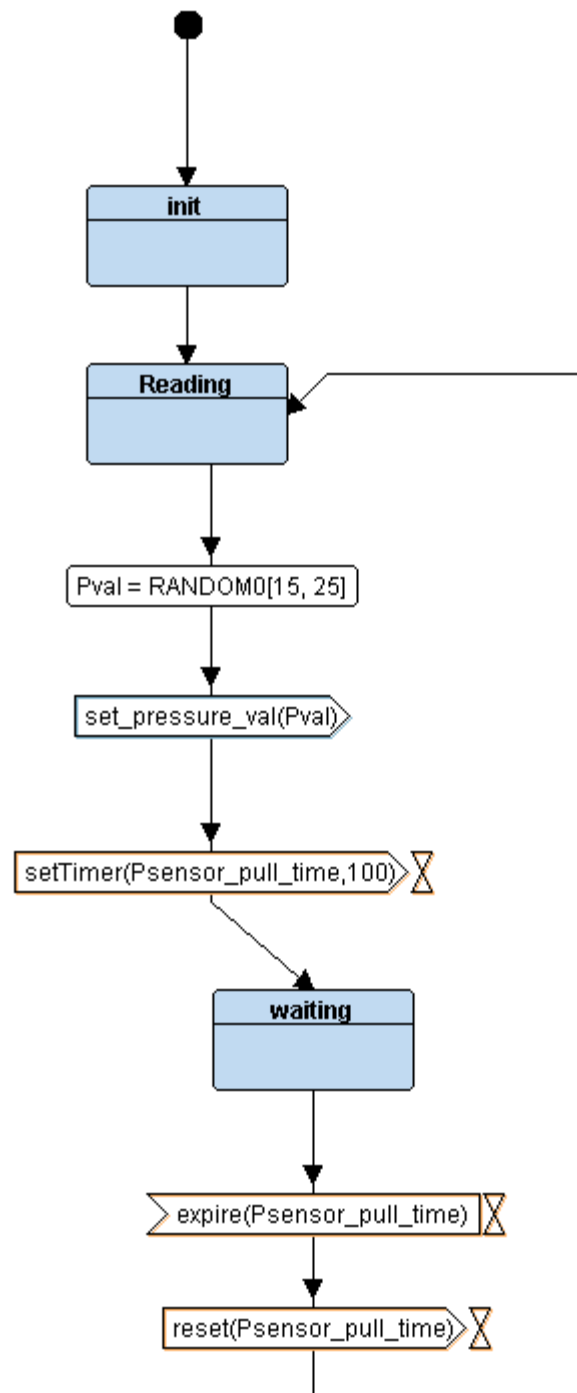


System Design



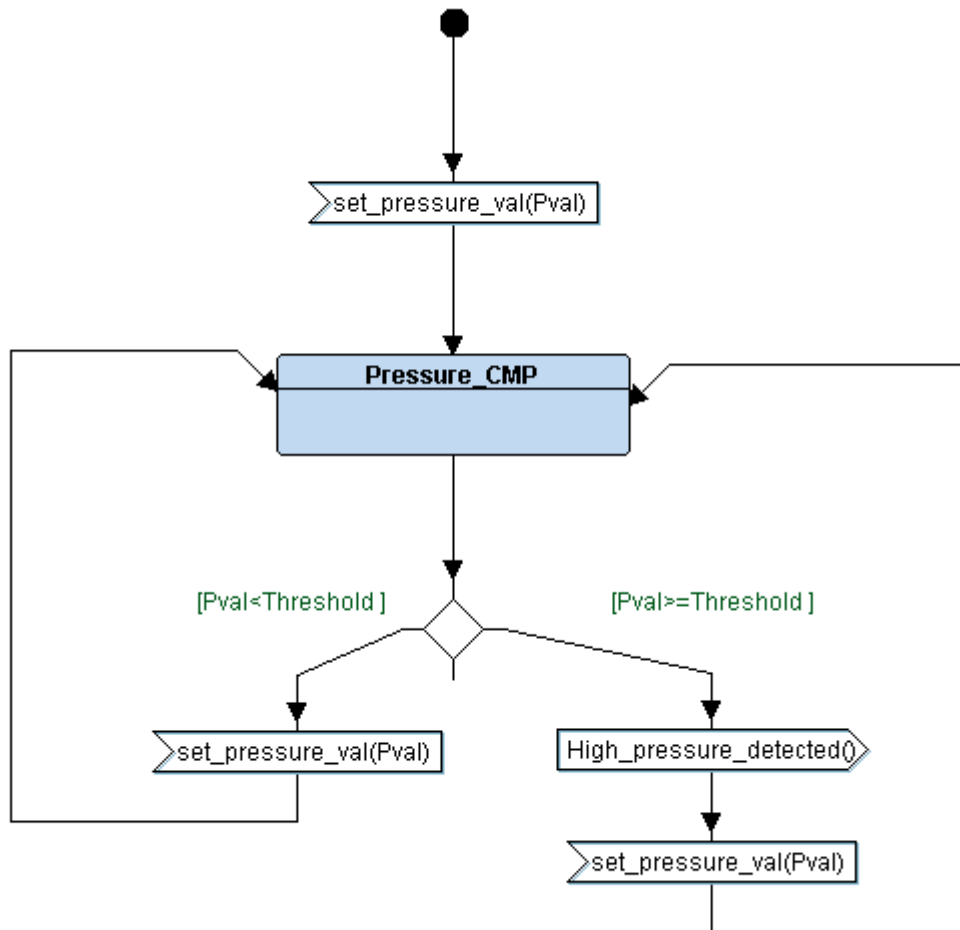
Pressure Sensor

- **Reading:** Reads pressure values and initiates pressure comparison.
- **Waiting:** Waits for a specified time before returning to reading.



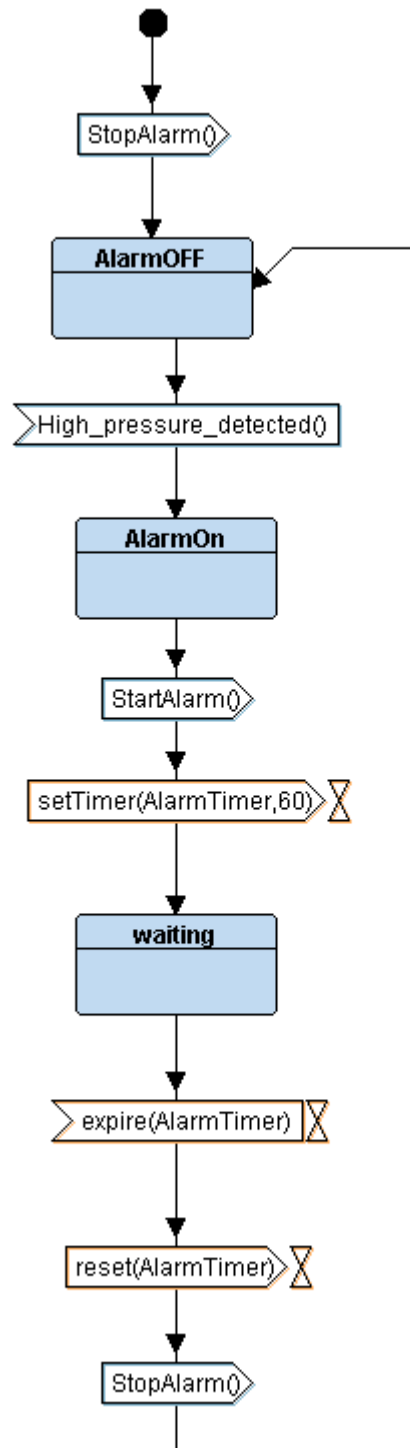
Main Algorithm

- **Pressure_CMP**: Compares pressure values with a predefined threshold.



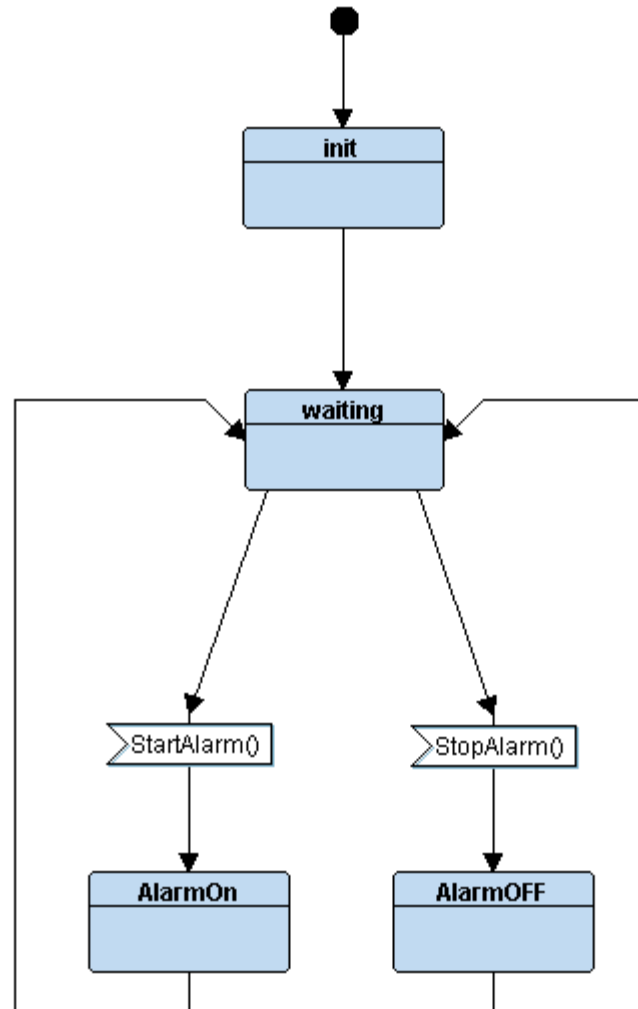
Alarm Monitor

- **AM_AlarmOFF**: Represents the state when the alarm is deactivated.
- **AM_AlarmOn**: Activates the alarm and transitions to AM_waiting.
- **AM_waiting**: Waits for a specified time interval before deactivating the alarm.



Alarm Actuator

- **AA_waiting**: Represents the waiting state of the alarm actuator.
- **AA_AlarmOn**: Activates the alarm actuator to trigger an alarm.
- **AA_AlarmOFF**: Deactivates the alarm actuator to stop the alarm.



SW Parameters

Map File

https://drive.google.com/file/d/1v9XWpOpI0peZbOMPI5qaAKMQAaOf3PwS/view?usp=drive_link

Sections

https://drive.google.com/file/d/1IW42uUNS3AmdDmgfmfVa2VX8wFbzzdje/view?usp=drive_link

Symbols

https://drive.google.com/file/d/1QcMVpLx3l14j5V7Nhi12xLyTna6Imxxg/view?usp=drive_link

Code Appendix

State.h

```
#ifndef STATE_H
#define STATE_H
#include "driver.h"
#include "stdio.h"
#include "platform_types.h"
#define STATE_DECLARATION(state_name) void state_name(void)
#define STATE(state_name) state_name
void High_pressure_detected();
void StartAlarm();
void StopAlarm();
#endif
```

PressureSensor_Driver.h

```
#ifndef PRESSURE_SENSOR_DRIVER_H
#define PRESSURE_SENSOR_DRIVER_H
enum
{
    PS_reading_id,
    PS_waiting_id
}PressureSensor_state_id;
void PS_init();
STATE_DECLARATION(PS_reading);
STATE_DECLARATION(PS_waiting);
void Pressure_CMP(int Pval);
void (*PS_state_ptr)();
#endif
```

PressureSensor_Driver.c

```
#include "state.h"
#include "PressureSensor_Driver.h"
int Pval;
void (*PS_state_ptr)();
void PS_init()
{
    //init Pressure sensor driver
}
STATE_DECLARATION(PS_reading)
{
    PressureSensor_state_id = PS_reading_id;
    Pval = getPressureVal(); //Read Pressure value
    Pressure_CMP(Pval); //Send Pressure value to algorithm
    PS_state_ptr = STATE(PS_waiting); //Switch to waiting state
}
STATE_DECLARATION(PS_waiting)
{
    PressureSensor_state_id = PS_waiting_id;
    Delay(1000);
    PS_state_ptr = STATE(PS_reading);
}
```

algorithm.h

```
#ifndef ALGORITHM_H
#define ALGORITHM_H

enum{
    ALGO_CMP_id
}algo_state_id;

extern void (*algo_state_ptr)();
void wait_test();
void Pressure_CMP(int Pval);

#endif
```

algorithm.c

```
#include "state.h"
#include "algorithm.h"
void (*algo_state_ptr)();
int Threshold = 20;
void wait_test()
{
    algo_state_ptr = STATE(Pressure_CMP);
}
void Pressure_CMP(int Pval)
{
    algo_state_id = ALGO_CMP_id;
    if (Pval >= Threshold)
        High_pressure_detected();
    algo_state_ptr = STATE(wait_test);
}
```

AlarmMonitor.h

```
#ifndef ALARM_MONITOR_H
#define ALARM_MONITOR_H

enum
{
    AM_AlarmOFF_id,
    AM_AlarmOn_id,
    AM_waiting_id
}AM_state_id;

extern void (*AM_state_ptr)();
void AM_init();
STATE_DECLARATION(AM_AlarmOFF);
STATE_DECLARATION(AM_AlarmOn);
STATE_DECLARATION(AM_waiting);
#endif
```

AlarmMonitor.c

```
#include "state.h"
#include "AlarmMonitor.h"
void (*AM_state_ptr)();
void AM_init()
{
    //init Alarm Monitor
}
STATE_DECLARATION(AM_AlarmOFF)
{
    AM_state_id = AM_AlarmOFF_id;
}
STATE_DECLARATION(AM_AlarmOn)
{
    AM_state_id = AM_AlarmOn_id;
    StartAlarm();
    AM_state_ptr = STATE(AM_waiting);
}
STATE_DECLARATION(AM_waiting)
{
    AM_state_id = AM_waiting_id;
    Delay(6000);
    StopAlarm();
    AM_state_ptr = STATE(AM_AlarmOFF);
}
void High_pressure_detected()
{
    AM_state_ptr = STATE(AM_AlarmOn);
}
```

AlarmActuator.h

```
#ifndef ALARM_ACTUATOR_H
#define ALARM_ACTUATOR_H

enum
{
    AA_waiting_id,
    AA_AlarmOn_id,
    AA_AlarmOFF_id
}AA_state_id;

extern void (*AA_state_ptr)();
void AA_init(void);
STATE_DECLARATION(AA_waiting);
STATE_DECLARATION(AA_AlarmOn);
STATE_DECLARATION(AA_AlarmOFF);
#endif
```

AlarmActuator.c

```
#include "state.h"
#include "AlarmActuator.h"
void (*AA_state_ptr)();
void AA_init(void)
{
    //init alarm actuator
    AA_state_ptr = STATE(AA_waiting);
}
STATE_DECLARATION(AA_waiting)
{
    AA_state_id = AA_waiting_id;
}
STATE_DECLARATION(AA_AlarmOn)
{
    AA_state_id = AA_AlarmOn_id;
    Set_Alarm_actuator(0);
}
STATE_DECLARATION(AA_AlarmOFF)
{
    AA_state_id = AA_AlarmOFF_id;
    Set_Alarm_actuator(1);
}
void StartAlarm()
{
    AA_state_ptr = STATE(AA_AlarmOn);
}
void StopAlarm()
{
    AA_state_ptr = STATE(AA_AlarmOFF);
}
```

main.c

```
#include "state.h"

#include "PressureSensor_Driver.h"

#include "algorithm.h"

#include "AlarmMonitor.h"

#include "AlarmActuator.h"

void setup()

#include "state.h"
#include "PressureSensor_Driver.h"
#include "algorithm.h"
#include "AlarmMonitor.h"
#include "AlarmActuator.h"

void setup()
{
    //Init block
    PS_init();
    AA_init();
    AM_init();
    //Set state pointer for each block
    PS_state_ptr = STATE(PS_reading);
    algo_state_ptr = STATE(Pressure_CMP);
    AM_state_ptr = STATE(AM_AlarmOFF);
    AA_state_ptr = STATE(AA_waiting);
}

int main () {
    //GPIO_INITIALIZATION();
    setup();
    while (1)
    {
        //Implement your Design
        PS_state_ptr();
        AM_state_ptr();
        AA_state_ptr();
    }
}
```