Mastering Embedded System Online Diploma

www.learn-in-depth.com

First Term (Final Project 1)

Eng. Abdallah Khater

My Profile:

https://www.learn-in-depth.com/online-diploma/abdallahahmed363%40gmail.com

Github:

https://github.com/AbdallahKhat

Contents

1. Case Study: A Pressure Controlling System

A client expects you to deliver the software of the following system:

Specification (from the client): 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.

1.1 Assumptions:

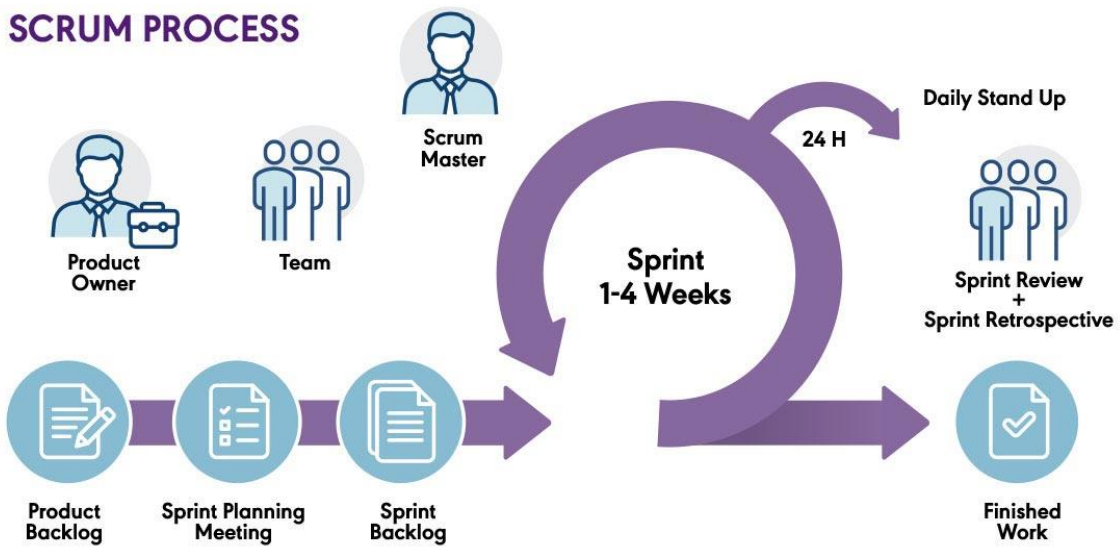
- The controller set up and shutdown procedures are not modeled.
- The controller maintenance is not modeled.
- The pressure sensor never fails.
- The alarm never fails.
- The controller never faces power cut.
- The “keep track of measured value” option is not modeled in the first version of the design

2. Design Sequence

The sequence of activities carried out by Developers to design and develop high-quality software.

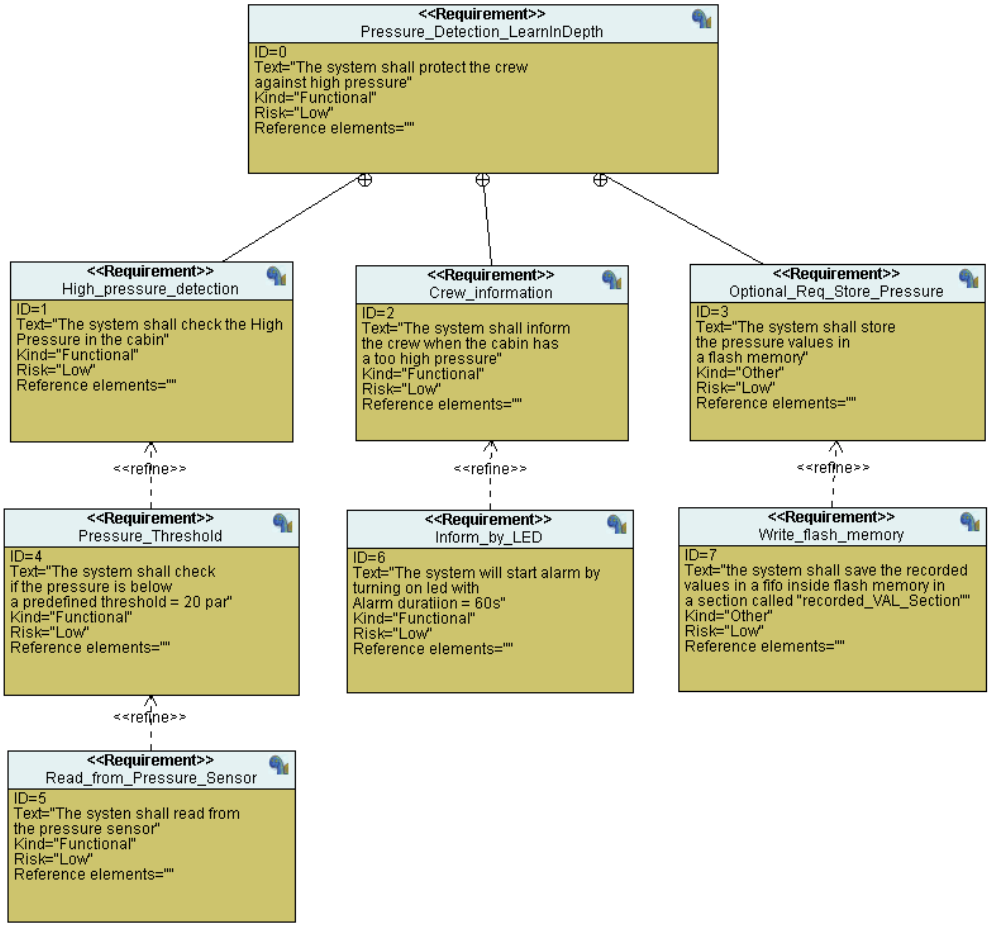
2.1 Method

The used method in this case study is Agile Scrum, which is a framework for software development that emphasizes iterative and incremental development cycles called sprints.



2.2 System Requirements UML

This system requirement UML (Unified Modeling Language) diagram is a type of UML diagram that is used to model and specify the system requirements of the software system.



2.3 Design Space Exploration

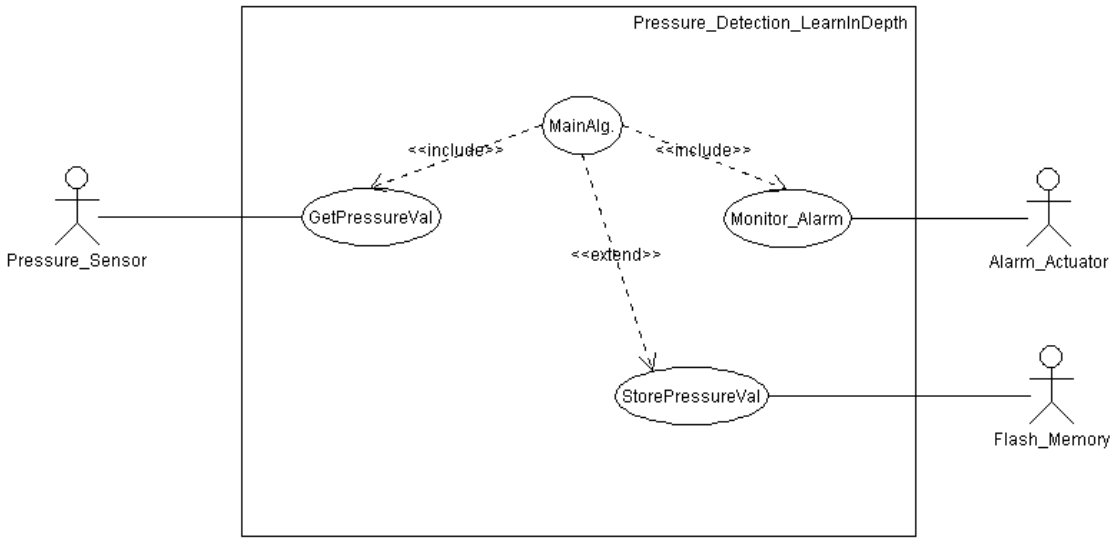
The SW will be implemented on a STM32F103C8 Microcontroller.

2.4 System Analysis

- System boundary and main functions → Use Case Diagram
- Relations between main functions → Activity Diagram
- Communications between main system entities and actors → Sequence Diagram

2.4.1 Use Case Diagram

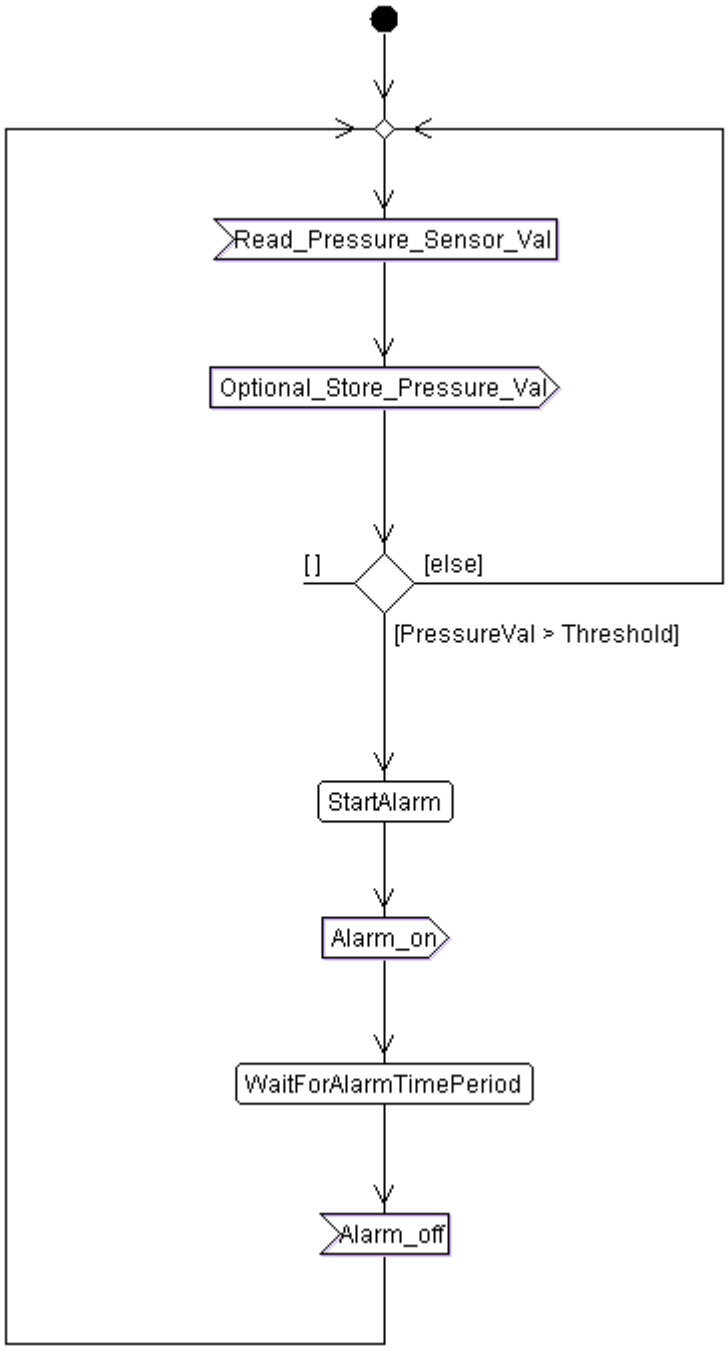
Shows what the system does and who uses it. A use case diagram UML is used to model the interactions between the system and its external systems (actors). It defines the boundary of the system.



2.4.2 Activity Diagram

The activity diagram describes the workflow behavior of the system.

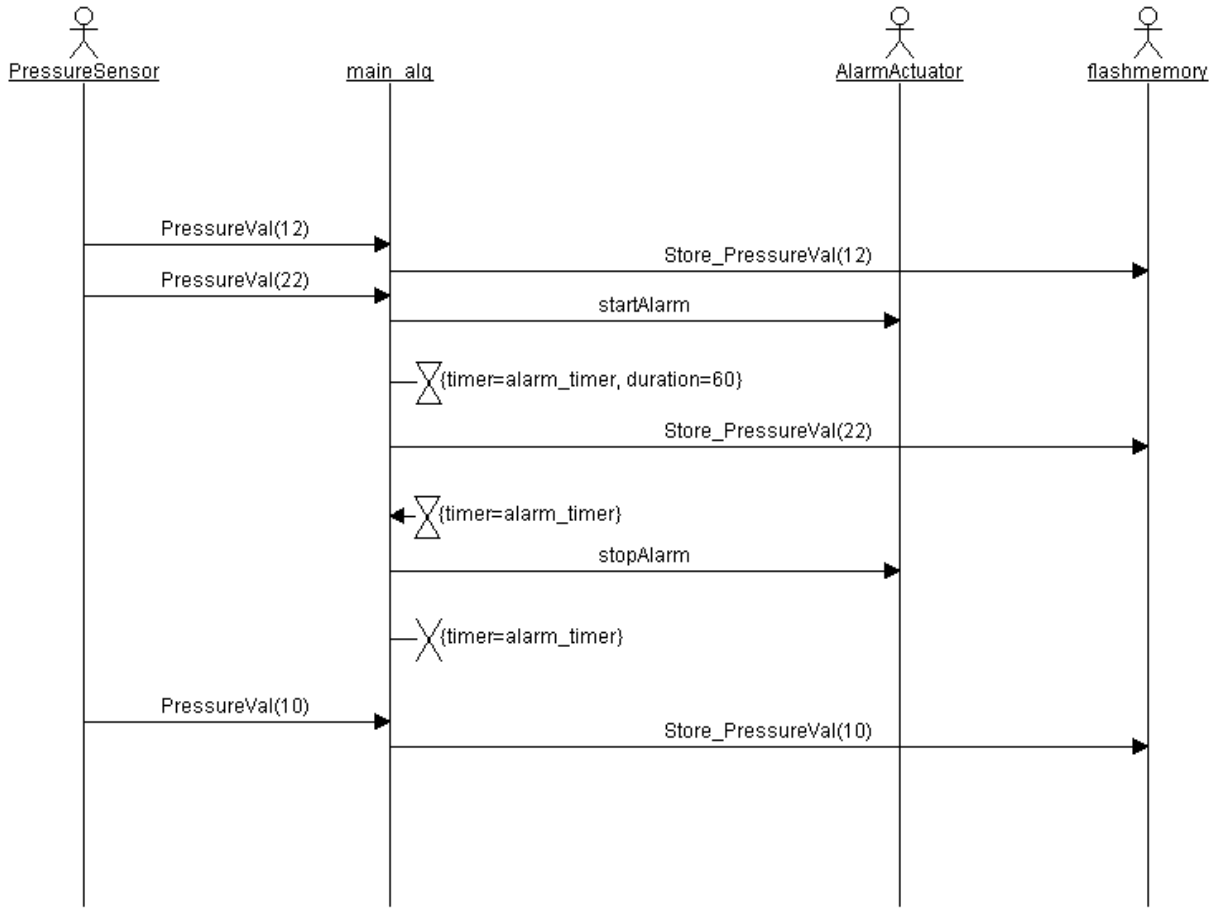
An activity diagram is a special case of a state chart diagram in which states are activities (“functions”).



2.4.3 Sequence Diagram

The sequence diagram is:

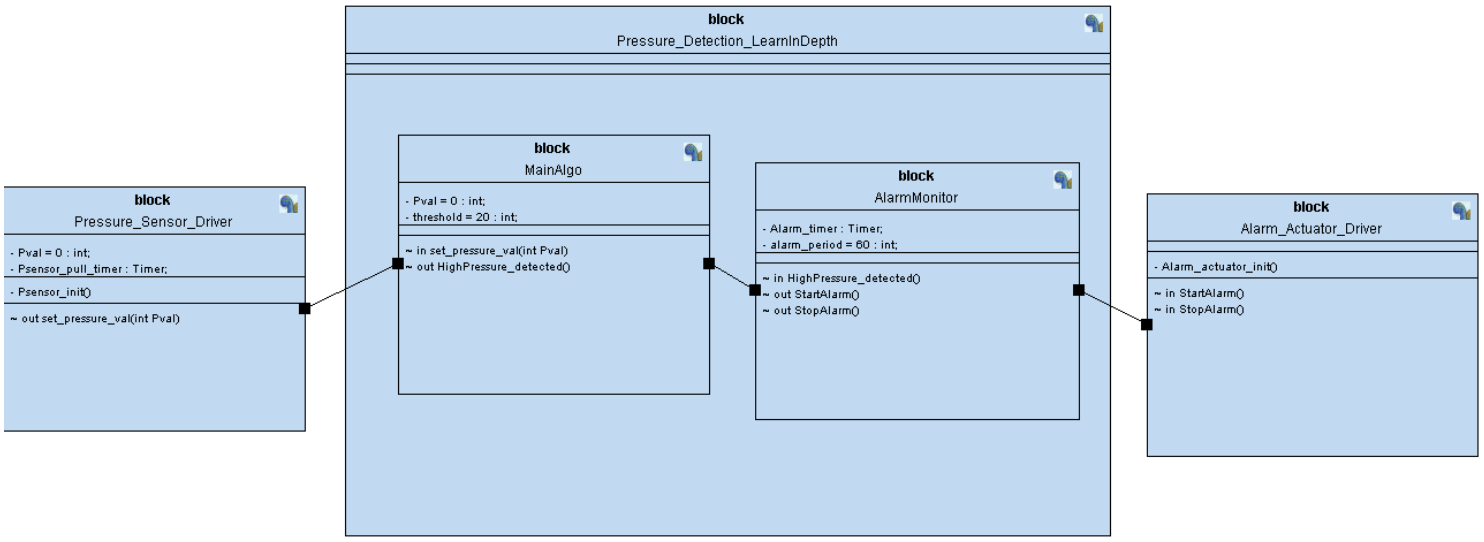
- An interaction diagram that details how operations are carried out.
- What messages are sent and when.
- Sequence diagrams are organized according to time.
- NO message between actors.
- One global clock (applies to the entire system).



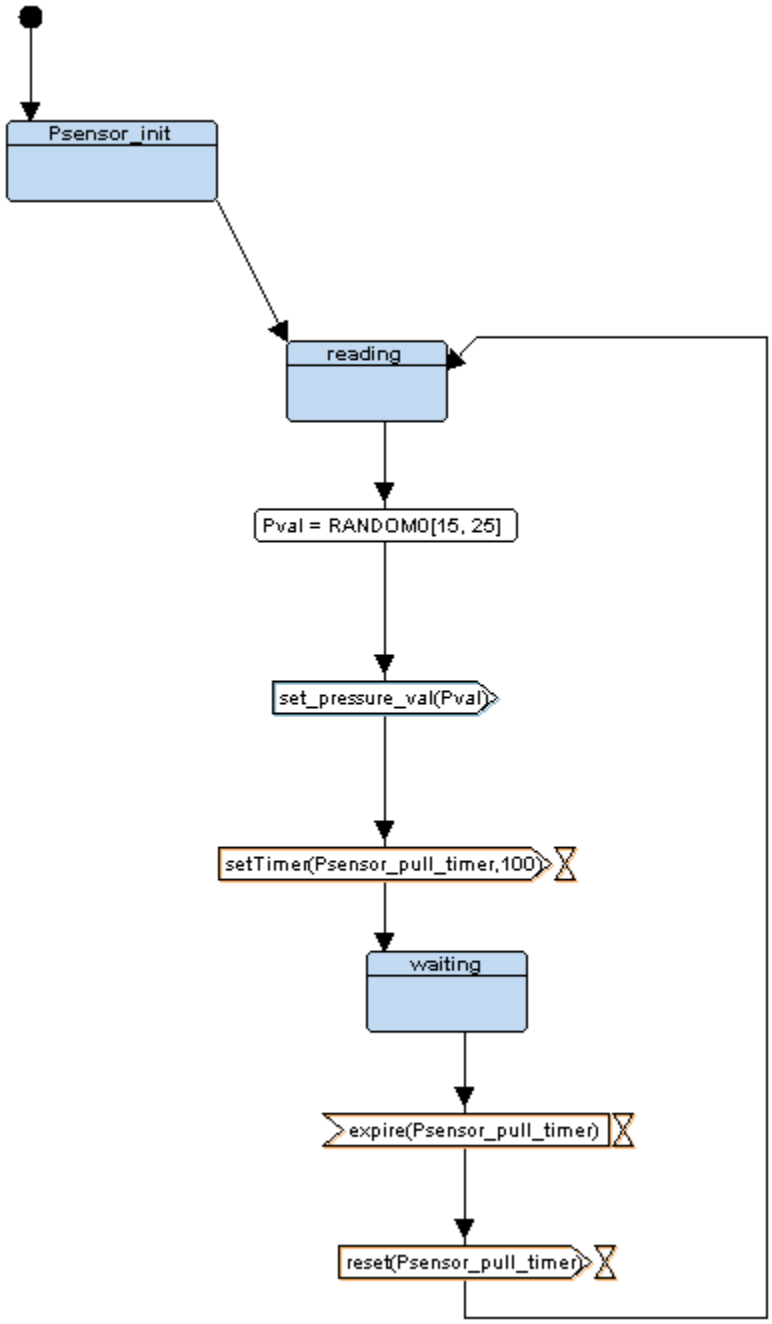
2.5 System Design

2.5.1 Block Diagram UML

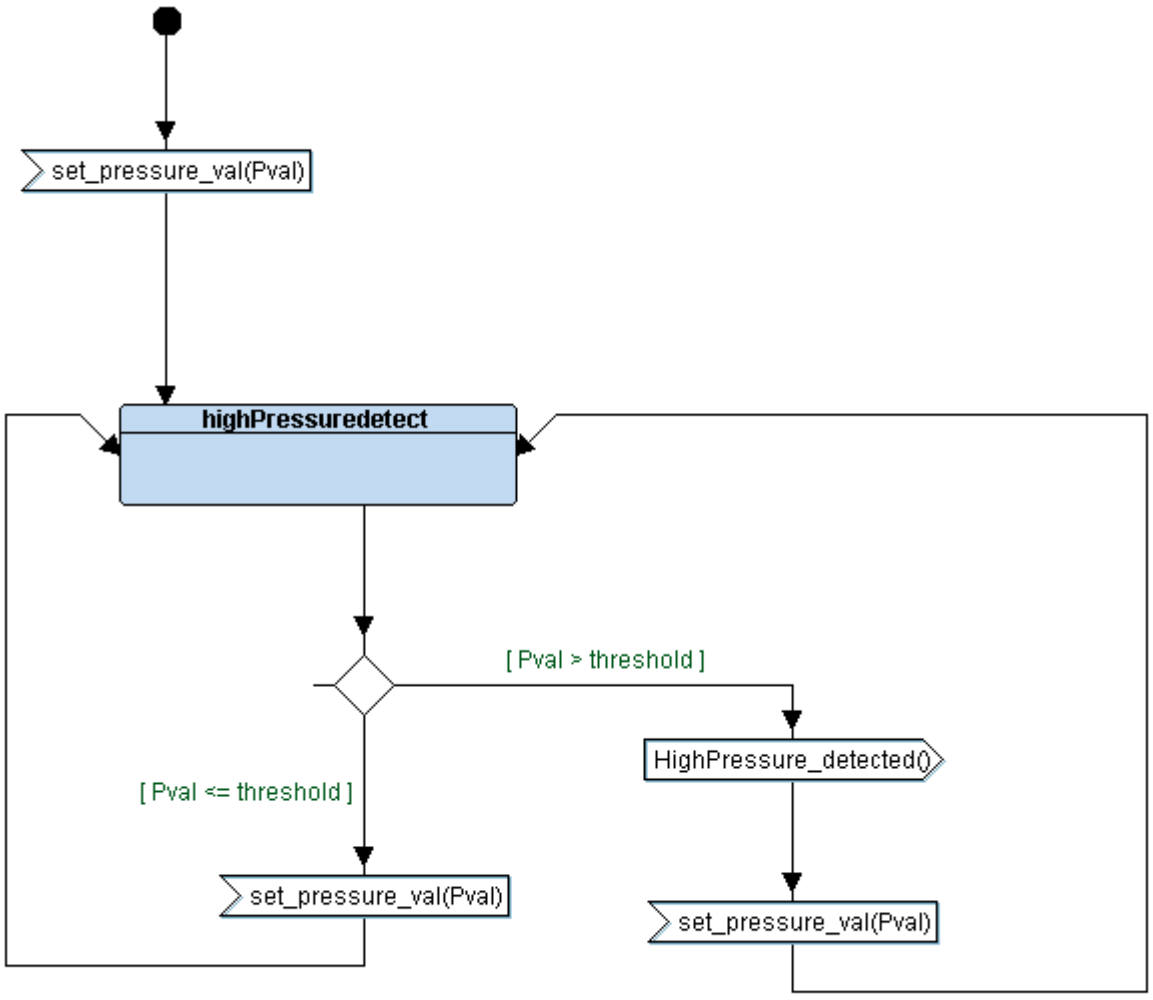
Block Diagram UML of the system:



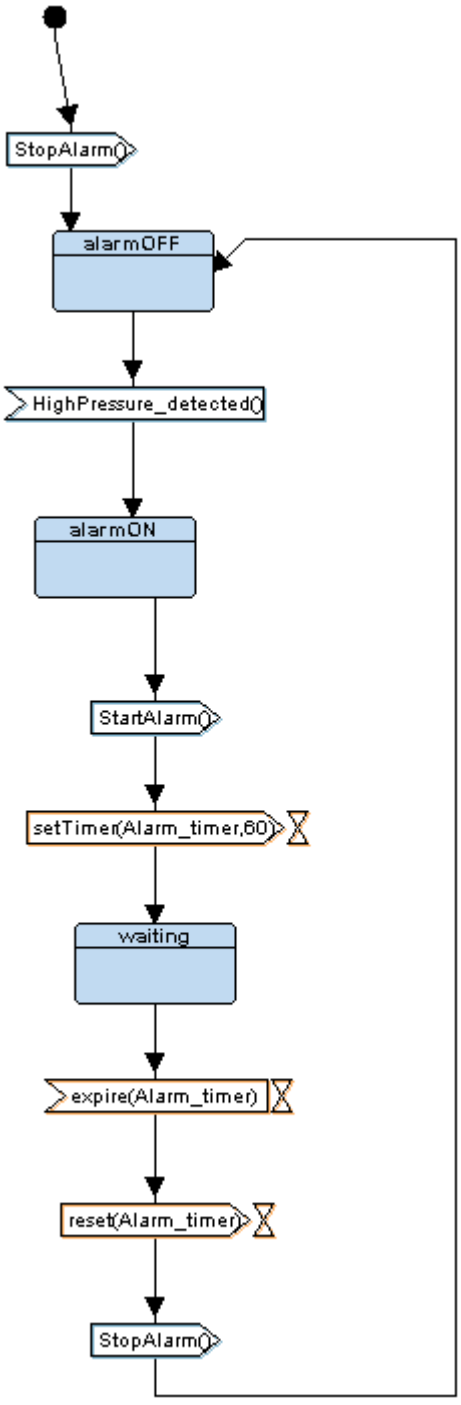
2.5.2 State Machine: Pressure_Sensor_Driver



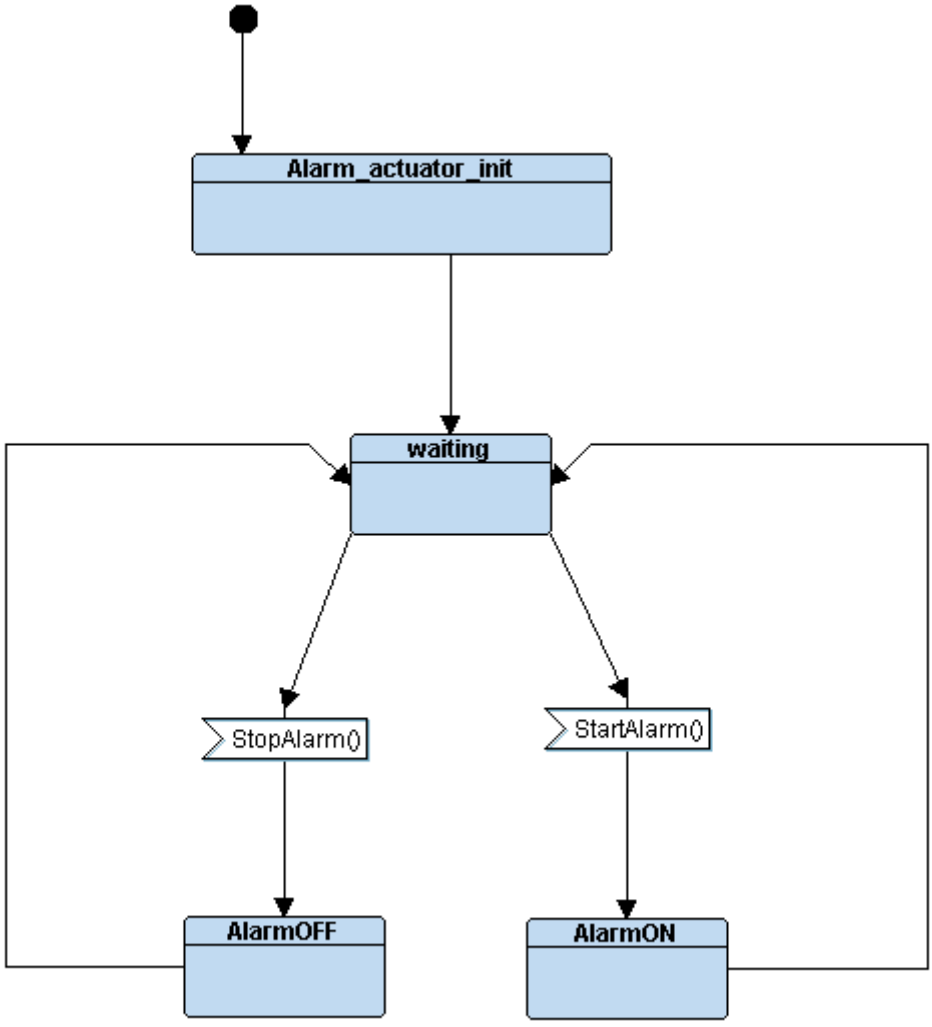
2.5.3 State Machine: MainAlgo



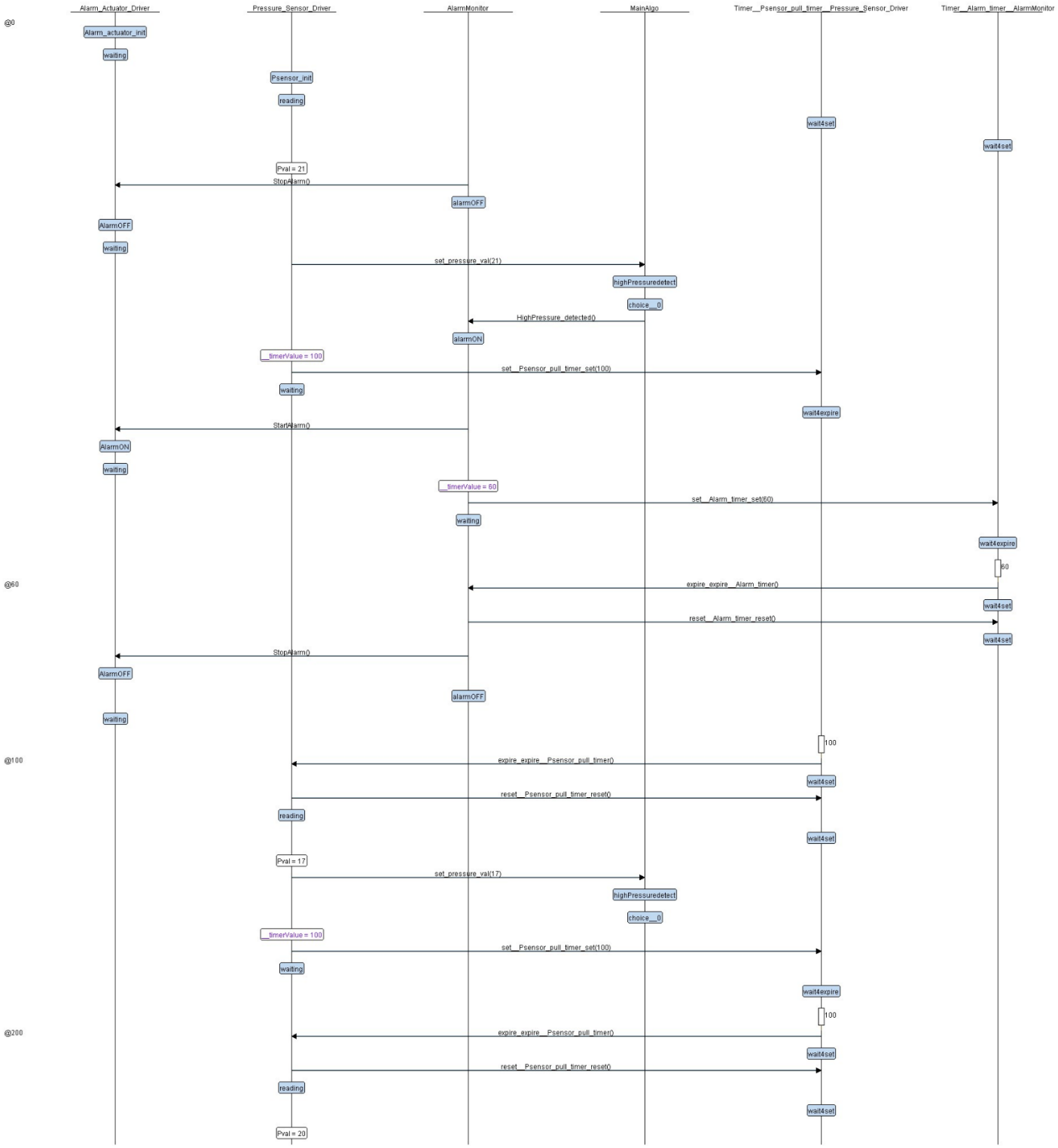
2.5.4 State Machine: AlarmMonitor



2.5.5 State Machine: Alarm_Actuator_Driver



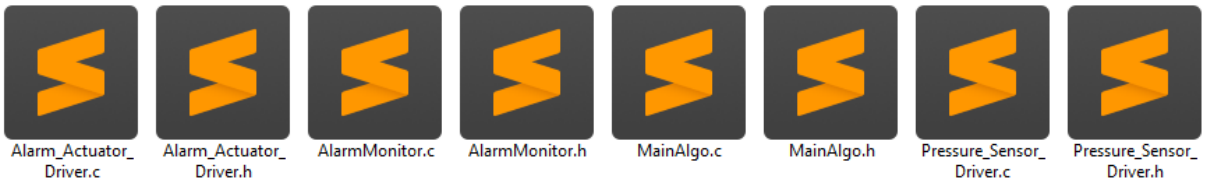
2.5.6 Simulation Trace



3. Design Implementation

The design will be implemented in 4 Modules which are:

1. Pressure_Sensor_Driver (.c/.h)
2. MainAlgo (.c/.h)
3. AlarmMonitor (.c/.h)
4. Alarm_Actuator_Driver (.c/.h)



Other files:

- main.c to run the SW.
- states.h header to define module states and declare function connections.
- driver (.c/.h) to drive the GPIO and read from sensor.

3.1 Pressure_Sensor_Driver (header & source)

Symbol Table:

```
$ arm-none-eabi-nm.exe Pressure_Sensor_Driver.o
         U Delay
         U getPressureVal
00000008 B pfPsensor_state
00000004 b Pressure_value
00000000 T Psensor_init
00000000 B Psensor_state_id
         U set_pressure_val
0000000c T ST_Psensor_reading
00000044 T ST_Psensor_waiting
```

Header:

```
 8  #ifndef PRESSURE_SENSOR_DRIVER_H_
 9  #define PRESSURE_SENSOR_DRIVER_H_
10
11  #include "state.h"
12
13  //Define states
14  enum NUMPsensor{
15      Psensor_reading,
16      Psensor_waiting
17  };
18
19  //Declare states functions for Pressure Sensor
20
21  STATE_define(Psensor_reading);
22  STATE_define(Psensor_waiting);
23
24  void Psensor_init();
25
26  //State pointer to function
27  extern void (*pfPsensor_state)();
28
29  #endif /* PRESSURE_SENSOR_DRIVER_H_ */
30
```

Source:

```
7
8   #include "Pressure_Sensor_Driver.h"
9
10  //global ENUM Variable
11
12  enum NUMPsensor Psensor_state_id;
13
14  //Variables (Scope: only Pressure_Sensor_Driver.c)
15
16  static int Pressure_value = 0;
17
18  //STATE pointer to function
19
20  void (*pfPsensor_state)();
21
22  //function definition
23
24  void Psensor_init()
25  {
26      //init Pressure sensor
27  }
28
29  STATE_define(Psensor_reading)
30  {
31      //State Name
32      Psensor_state_id = Psensor_reading;
33
34      //State Action
35      //get pressure value from sensor
36      Pressure_value = getPressureVal();
37
38      //send pressure value received to MainAlgo
39      set_pressure_val(Pressure_value);
40
41      //Set State pointer to Psensor_waiting
42      pfPsensor_state = STATE(Psensor_waiting);
43  }
44
45  STATE_define(Psensor_waiting)
46  {
47      //State Name
48      Psensor_state_id = Psensor_waiting;
49
50      //state action
51      //delay
52      Delay(100000);
53
54      //Set State pointer to Psensor_reading
55      pfPsensor_state = STATE(Psensor_reading);
56      pfPsensor_state();
57  }
```

3.2 MainAlgo (header & source)

Symbol Table:

```
$ arm-none-eabi-nm.exe MainAlgo.o
         U Highpressure_detected
00000000 B MainAlgo_state_id
00000004 b P_value
00000008 B pfMainAlgo_state
00000000 T set_pressure_val
0000002c T ST_high_Pressure_detect
00000000 d threshold
```

Header:

```
 7
 8  #ifndef MAINALGO_H_
 9  #define MAINALGO_H_
10
11  #include "state.h"
12
13  //Define states
14  enum NUMMainAlgo{
15      MainAlgo_high_Pressure_detect
16  };
17
18
19  //Declare states function for MainAlgo
20
21  STATE_define(high_Pressure_detect);
22
23  //State pointer to function
24  extern void (*pfMainAlgo_state)();
25
26  #endif /* MAINALGO_H_ */
27
```

Source:

```c
#include "MainAlgo.h"

//global ENUM Variable

enum NUMMainAlgo MainAlgo_state_id;

//Variables (Scope: only MainAlgo.c)

static int P_value = 0;
static int threshold = 20;

//STATE pointer to function

void (*pfMainAlgo_state)();

//function definition

void set_pressure_val(int Pval)
{
    //Set pressure value received from Pressure_Sensor_Driver
    P_value = Pval;

    //Set State pointer to high_Pressure_detect
    pfMainAlgo_state = STATE(high_Pressure_detect);

}

STATE_define(high_Pressure_detect)
{
    //State Name
    MainAlgo_state_id = MainAlgo_high_Pressure_detect;

    //State Action
    if(P_value > threshold)
    {
        Highpressure_detected();
    }
}
```

3.3 AlarmMonitor (header & source)

Symbol Table:

```
$ arm-none-eabi-nm.exe AlarmMonitor.o
00000000 d alarm_period
00000000 B AlarmMonitor_state_id
         U Delay
00000000 T Highpressure_detected
00000004 B pfAlarmMonitor_state
0000001c T ST_AlarmMonitor_alarmOFF
00000034 T ST_AlarmMonitor_alarmON
00000060 T ST_AlarmMonitor_waiting
         U StartAlarm
         U StopAlarm
```

Header:

```
 7
 8  #ifndef ALARMMONITOR_H_
 9  #define ALARMMONITOR_H_
10
11  #include "state.h"
12
13  //Define states
14  enum NUMMonitor{
15      Monitor_alarmOFF,
16      Monitor_alarmON,
17      Monitor_waiting
18  };
19
20  //Declare states functions for Pressure Sensor
21
22  STATE_define(AlarmMonitor_alarmOFF);
23  STATE_define(AlarmMonitor_alarmON);
24  STATE_define(AlarmMonitor_waiting);
25
26  //State pointer to function
27  extern void (*pfAlarmMonitor_state)();
28
29
30  #endif /* ALARMMONITOR_H_ */
31
```

Source:

```
#include "AlarmMonitor.h"

//global ENUM Variable

enum NUMMonitor AlarmMonitor_state_id;

//Variables (Scope: only AlarmMonitor.c)

static int alarm_period = 600000;

//STATE pointer to function

void (*pfAlarmMonitor_state)();

//function definition

void Highpressure_detected()
{
    //Set State pointer to AlarmMonitor_alarmON
    pfAlarmMonitor_state = STATE(AlarmMonitor_alarmON);
}

STATE_define(AlarmMonitor_alarmOFF)
{
    //State Name
    AlarmMonitor_state_id = Monitor_alarmOFF;

    //state action (do nothing)
}

STATE_define(AlarmMonitor_alarmON)
{
    //State Name
    AlarmMonitor_state_id = Monitor_alarmON;

    //state action
    //Start Alarm
    StartAlarm();

    //Set State pointer to AlarmMonitor_waiting
    pfAlarmMonitor_state = STATE(AlarmMonitor_waiting);
    pfAlarmMonitor_state();

}
```

```
54  STATE_define(AlarmMonitor_waiting)
55  {
56      //State Name
57      AlarmMonitor_state_id = Monitor_waiting;
58
59      //state action
60      //delay
61      Delay(alarm_period);
62
63      //Stop Alarm
64      StopAlarm();
65
66      //Set State pointer to AlarmMonitor_alarmOFF
67      pfAlarmMonitor_state = STATE(AlarmMonitor_alarmOFF);
68
69  }
70
```

3.4 Alarm_Actuator_Driver (header & source)

Symbol Table:

```
$ arm-none-eabi-nm.exe Alarm_Actuator_Driver.o
00000000 T Alarm_actuator_init
00000000 B Alarm_actuator_state_id
00000004 B pfAlarm_actuator_state
         U Set_Alarm_actuator
0000005c T ST_Alarm_actuator_AlarmOFF
00000084 T ST_Alarm_actuator_AlarmON
00000044 T ST_Alarm_actuator_waiting
00000028 T StartAlarm
0000000c T StopAlarm
```

Header:

```
 7
 8  #ifndef ALARM_ACTUATOR_DRIVER_H_
 9  #define ALARM_ACTUATOR_DRIVER_H_
10
11  #include "state.h"
12
13  //Define states
14  enum NUMAlarm_Act {
15      Alarm_actuator_AlarmOFF,
16      Alarm_actuator_AlarmON,
17      Alarm_actuator_waiting
18  };
19
20  //Declare states functions for Pressure Sensor
21
22  STATE_define(Alarm_actuator_waiting);
23  STATE_define(Alarm_actuator_AlarmOFF);
24  STATE_define(Alarm_actuator_AlarmON);
25
26
27  void Alarm_actuator_init();
28
29  //State pointer to function
30  extern void (*pfAlarm_actuator_state)();
31
32  #endif /* ALARM_ACTUATOR_DRIVER_H_ */
33
```

Source:

```
#include "Alarm_Actuator_Driver.h"

//global ENUM Variable

enum NUMAlarm_Act Alarm_actuator_state_id;

//STATE pointer to function

void (*pfAlarm_actuator_state)();

//function definition

void Alarm_actuator_init()
{
    //init Alarm actuator
}

void StopAlarm()
{
    //Set State pointer to Alarm_actuator_AlarmOFF
    pfAlarm_actuator_state = STATE(Alarm_actuator_AlarmOFF);
    pfAlarm_actuator_state();
}

void StartAlarm()
{
    //Set State pointer to Alarm_actuator_AlarmON
    pfAlarm_actuator_state = STATE(Alarm_actuator_AlarmON);
    pfAlarm_actuator_state();
}

STATE_define(Alarm_actuator_waiting)
{
    //State Name
    Alarm_actuator_state_id = Alarm_actuator_waiting;

    //state action (do nothing)
}
```

```
48  STATE_define(Alarm_actuator_AlarmOFF)
49  {
50      //State Name
51      Alarm_actuator_state_id = Alarm_actuator_AlarmOFF;
52
53      //state action
54      //Turn off alarm
55      Set_Alarm_actuator(0);
56
57      //Set State pointer to Alarm_actuator_waiting
58      pfAlarm_actuator_state = STATE(Alarm_actuator_waiting);
59  }
60
61  STATE_define(Alarm_actuator_AlarmON)
62  {
63      //State Name
64      Alarm_actuator_state_id = Alarm_actuator_AlarmON;
65
66      //state action
67      //Turn on alarm
68      Set_Alarm_actuator(1);
69
70      //Set State pointer to Alarm_actuator_waiting
71      pfAlarm_actuator_state = STATE(Alarm_actuator_waiting);
72  }
73
```

3.5 Additional Files

3.5.1 main.c

Source & Symbol Table:

```c
#include "driver.h"
#include "MainAlgo.h"
#include "AlarmMonitor.h"
#include "Pressure_Sensor_Driver.h"
#include "Alarm_Actuator_Driver.h"

void setup()
{
    //init all drivers
    //init IRQ ....
    //init HAL Pressure_Sensor_Driver Alarm_Actuator_Driver
    Psensor_init();
    Alarm_actuator_init();


    //Set States pointer for each Block
    pfPsensor_state = STATE(Psensor_reading);
    pfMainAlgo_state = NULL;    /* expected to be set to state high_Pressure_detect after */
    /* set_pressure_val() function is called from Pressure_Sensor_Driver */
    pfAlarmMonitor_state = STATE(AlarmMonitor_alarmOFF);
    pfAlarm_actuator_state = STATE(Alarm_actuator_waiting);

}

int main (){

    GPIO_INITIALIZATION();

    setup();

    StopAlarm();


    while (1)
    {
        pfPsensor_state();
        pfMainAlgo_state();
        pfAlarmMonitor_state();
        pfAlarm_actuator_state();
    }

}
```

```
$ arm-none-eabi-nm.exe main.o
         U Alarm_actuator_init
         U GPIO_INITIALIZATION
00000044 T main
         U pfAlarm_actuator_state
         U pfAlarmMonitor_state
         U pfMainAlgo_state
         U pfPsensor_state
         U Psensor_init
00000000 T setup
         U ST_Alarm_actuator_waiting
         U ST_AlarmMonitor_alarmOFF
         U ST_Psensor_reading
         U StopAlarm
```

3.5.2 states.h

```
#ifndef STATE_H_
#define STATE_H_

#include "stdio.h"
#include "stdlib.h"
#include "driver.h"

//Automatic state function generation

#define STATE_define(_StateFun_) void ST_##_StateFun_()
#define STATE(_StateFun_) ST_##_StateFun_

//States Connection

void set_pressure_val(int Pval);
void Highpressure_detected();
void StartAlarm();
void StopAlarm();


#endif /* STATE_H_ */

```

3.5.3 driver (header & Source)

Symbol Table:

```
$ arm-none-eabi-nm.exe driver.o
00000000 T Delay
00000022 T getPressureVal
00000074 T GPIO_INITIALIZATION
00000038 T Set_Alarm_actuator
```

Header:

```
1   #include <stdint.h>
2   #include <stdio.h>
3
4   #define SET_BIT(ADDRESS,BIT)   ADDRESS |=  (1<<BIT)
5   #define RESET_BIT(ADDRESS,BIT) ADDRESS &= ~(1<<BIT)
6   #define TOGGLE_BIT(ADDRESS,BIT)  ADDRESS ^=  (1<<BIT)
7   #define READ_BIT(ADDRESS,BIT) ((ADDRESS) &   (1<<(BIT)))
8
9
10  #define GPIO_PORTA 0x40010800
11  #define BASE_RCC   0x40021000
12
13  #define APB2ENR   *(volatile uint32_t *)(BASE_RCC + 0x18)
14
15  #define GPIOA_CRL *(volatile uint32_t *)(GPIO_PORTA + 0x00)
16  #define GPIOA_CRH *(volatile uint32_t *)(GPIO_PORTA + 0X04)
17  #define GPIOA_IDR *(volatile uint32_t *)(GPIO_PORTA + 0x08)
18  #define GPIOA_ODR *(volatile uint32_t *)(GPIO_PORTA + 0x0C)
19
20
21  void Delay(int nCount);
22  int getPressureVal();
23  void Set_Alarm_actuator(int i);
24  void GPIO_INITIALIZATION ();
25
```

Source:

```c
#include "driver.h"
#include <stdint.h>
#include <stdio.h>
void Delay(int nCount)
{
    for(; nCount != 0; nCount--);
}

int getPressureVal(){
    return (GPIOA_IDR & 0xFF);
}

void Set_Alarm_actuator(int i){
    if (i == 0){
        SET_BIT(GPIOA_ODR,15);
    }
    else if (i == 1){
        RESET_BIT(GPIOA_ODR,15);
    }
}

void GPIO_INITIALIZATION (){
    SET_BIT(APB2ENR, 2);
    GPIOA_CRL &= 0xFFFFFFFF;
    GPIOA_CRL |= 0x00000000;
    GPIOA_CRH &= 0x0FFFFFFF;
    GPIOA_CRH |= 0x22222222;

}

```

3.6 Final Symbol Table and Header Sections

Final Symbol Table:

```
$ arm-none-eabi-nm.exe Pressure_Detection_LearnInDepth.elf
20000030 B _E_bss
20000008 D _E_DATA
080003f8 T _E_text
20000008 B _S_bss
20000000 D _S_DATA
20001030 B _stack_top
080000b0 T Alarm_actuator_init
20000010 B Alarm_actuator_state_id
20000000 d alarm_period
20000008 B AlarmMonitor_state_id
08000368 W Bus_Fault
08000368 T Default_Handler
08000228 T Delay
0800024a T getPressureVal
0800029c T GPIO_INITIALIZATION
08000368 W H_Fault_Handler
0800001c T Highpressure_detected
0800032c T main
20000018 B MainAlgo_state_id
08000368 W MM_Fault_Handler
08000368 W NMI_Handler
2000001c b P_value
20000014 B pfAlarm_actuator_state
2000000c B pfAlarmMonitor_state
20000020 B pfMainAlgo_state
2000002c B pfPsensor_state
20000028 b Pressure_value
080001b4 T Psensor_init
20000024 B Psensor_state_id
08000374 T Rest_Handler
08000260 T Set_Alarm_actuator
0800015c T set_pressure_val
080002e8 T setup
0800010c T ST_Alarm_actuator_AlarmOFF
08000134 T ST_Alarm_actuator_AlarmON
080000f4 T ST_Alarm_actuator_waiting
08000038 T ST_AlarmMonitor_alarmOFF
08000050 T ST_AlarmMonitor_alarmON
0800007c T ST_AlarmMonitor_waiting
08000188 T ST_high_Pressure_detect
080001c0 T ST_Psensor_reading
080001f8 T ST_Psensor_waiting
080000d8 T StartAlarm
080000bc T StopAlarm
20000004 d threshold
08000368 W Usage_Fault_Handler
08000000 T vectors
```

Header Sections (with debug info)

```
$ arm-none-eabi-objdump.exe -h Pressure_Detection_LearnInDepth.elf

Pressure_Detection_LearnInDepth.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000003f8  08000000  08000000  00010000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000008  20000000  080003f8  00020000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00001028  20000008  08000400  00020008  2**2
                  ALLOC
  3 .debug_info   0000087a  00000000  00000000  00020008  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  4 .debug_abbrev 00000565  00000000  00000000  00020882  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  5 .debug_loc    00000550  00000000  00000000  00020de7  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  6 .debug_aranges 000000e0  00000000  00000000  00021337  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  7 .debug_line   00000584  00000000  00000000  00021417  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  8 .debug_str    0000047d  00000000  00000000  0002199b  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  9 .comment      00000049  00000000  00000000  00021e18  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 0000002d  00000000  00000000  00021e61  2**0
                  CONTENTS, READONLY
 11 .debug_frame  0000033c  00000000  00000000  00021e90  2**2
                  CONTENTS, READONLY, DEBUGGING, OCTETS
```

Header Sections (without debug info)

```
$ arm-none-eabi-objdump.exe -h Pressure_Detection_LearnInDepth.elf

Pressure_Detection_LearnInDepth.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000003f8  08000000  08000000  00010000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000008  20000000  080003f8  00020000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00001028  20000008  08000400  00020008  2**2
                  ALLOC
  3 .comment      00000049  00000000  00000000  00020008  2**0
                  CONTENTS, READONLY
  4 .ARM.attributes 0000002d  00000000  00000000  00020051  2**0
                  CONTENTS, READONLY
```

4. Design Compilation

4.1 startup.c

```c
/* startup.c
Eng. Abdallah Khater
*/

#include <stdint.h>

extern int main(void);

void Rest_Handler(void);

void Default_Handler()
{
    Rest_Handler();
}

void NMI_Handler(void) __attribute__((weak,alias("Default_Handler")));;
void H_Fault_Handler(void) __attribute__((weak,alias("Default_Handler")));;
void MM_Fault_Handler(void) __attribute__((weak,alias("Default_Handler")));;
void Bus_Fault(void) __attribute__((weak,alias("Default_Handler")));;
void Usage_Fault_Handler(void) __attribute__((weak,alias("Default_Handler")));;

extern unsigned int _stack_top;
uint32_t vectors[] __attribute__((section(".vectors"))) = {
    (uint32_t) &_stack_top,
    (uint32_t) &Rest_Handler,
    (uint32_t) &NMI_Handler,
    (uint32_t) &H_Fault_Handler,
    (uint32_t) &MM_Fault_Handler,
    (uint32_t) &Bus_Fault,
    (uint32_t) &Usage_Fault_Handler
};
```

4.2 linker_script.ld

```
/* linker_script.ld
Eng.Abdallah Khater
*/

MEMORY
{
    flash(RX) : ORIGIN = 0x08000000, LENGTH = 128k
    sram(RWX) : ORIGIN = 0x20000000, LENGTH = 20k
}

SECTIONS
{
    .text :
    {
        *(.vectors*)
        *(.text*)
        *(.rodata)
        _E_text = . ;
    } > flash

    .data :
    {
        _S_DATA = . ;
        *(.data)
        _E_DATA = . ;
    } > sram AT> flash

    .bss :
    {
        _S_bss = . ;
        *(.bss*)
        . = ALIGN(4);
        _E_bss = . ;

        . = ALIGN(4);
        . = . + 0x1000 ;
        _stack_top = . ;

    } > sram
}
```

4.3 Makefile (for build automation)

```
#@copyright: Abdallah
CC=arm-none-eabi-
CFLAGS=-mcpu=cortex-m3 -gdwarf-2
INCS=-I .
LIBS=
SRC=$(wildcard *.c)
OBJ=$(SRC:.c=.o)
As=$(wildcard *.s)
AsOBJ=$(As:.s=.o)
project_name=Pressure_Detection_LearnInDepth

all: $(project_name).bin
    @echo "======Build is done======"

%.o: %.c
    $(CC)gcc.exe -c $(CFLAGS) $(INCS) $< -o $@

$(project_name).elf: $(OBJ) $(AsOBJ)
    $(CC)ld.exe -T linker_script.ld $(OBJ) $(AsOBJ) -o $@ -Map=Map_file.map

$(project_name).bin: $(project_name).elf
    $(CC)objcopy -O binary $< $@

clean_all:
    rm *.o *.elf *.bin

clean:
    rm *.elf *.bin

```