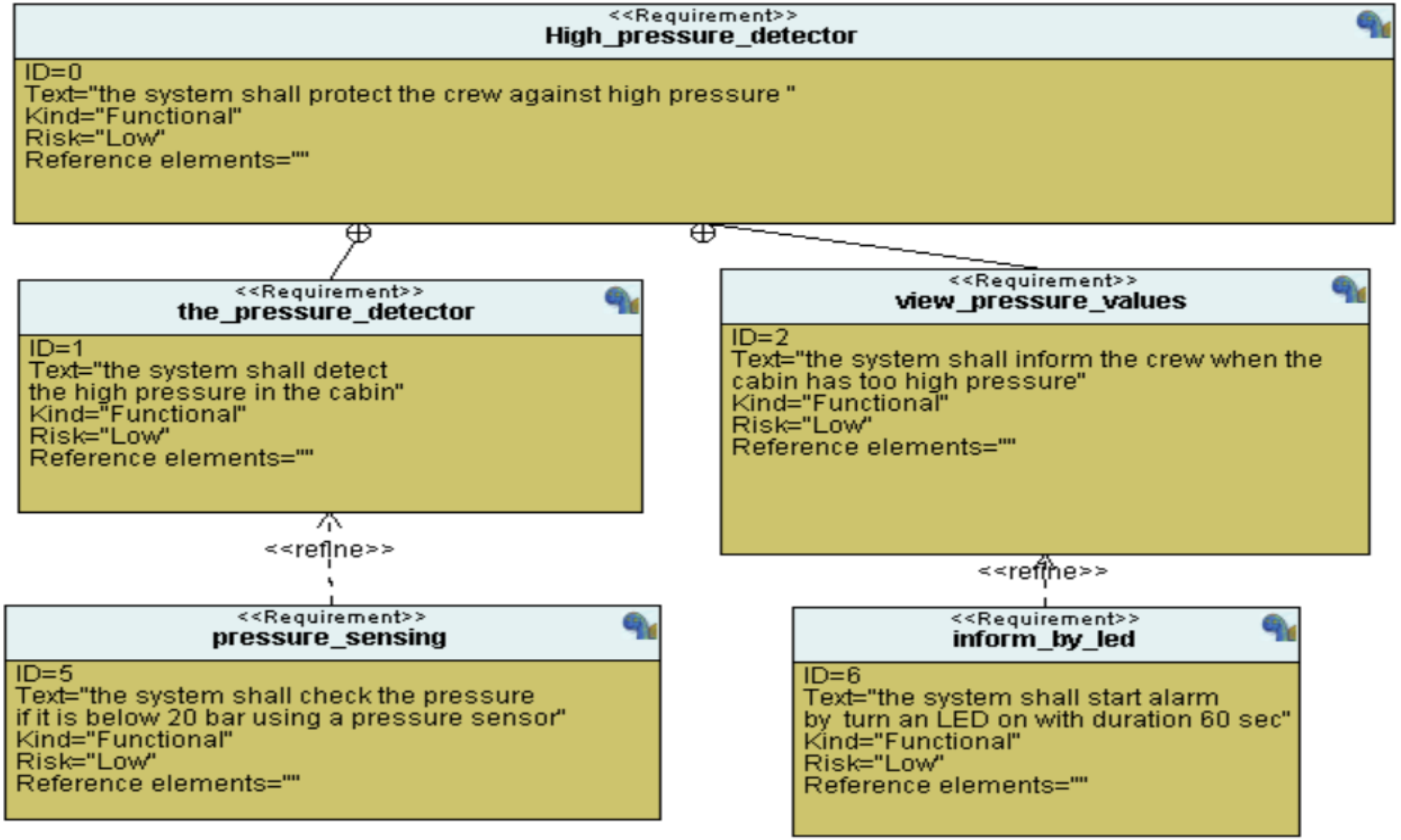
Mastering Embedded System Diploma
learn-in-depth.com

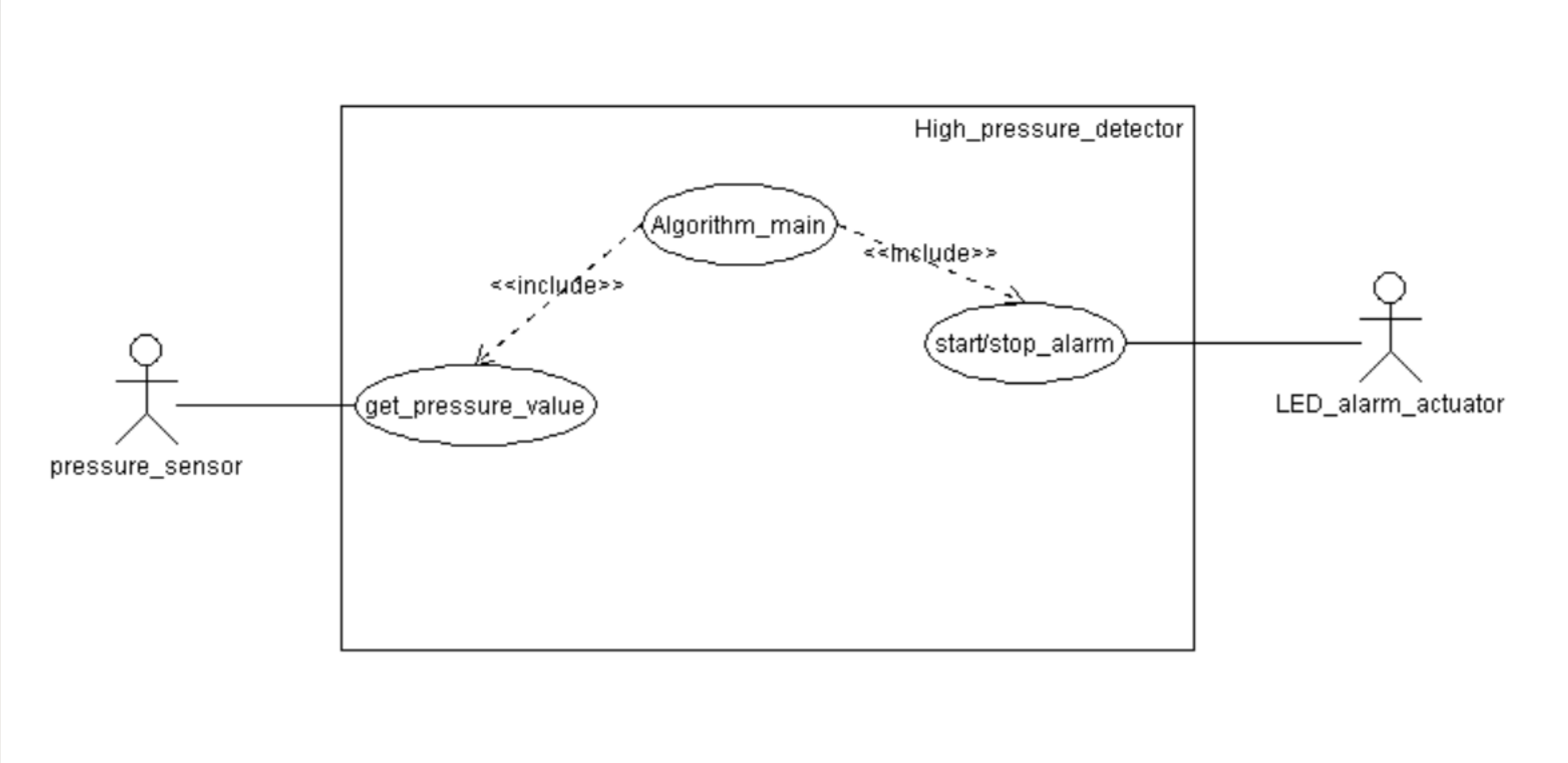
First Term (Final Project 1)
Omar Mohamed Ayman

My Profile: https://www.learn-in-depth.com/online-diploma/ommoor642@gmail.com

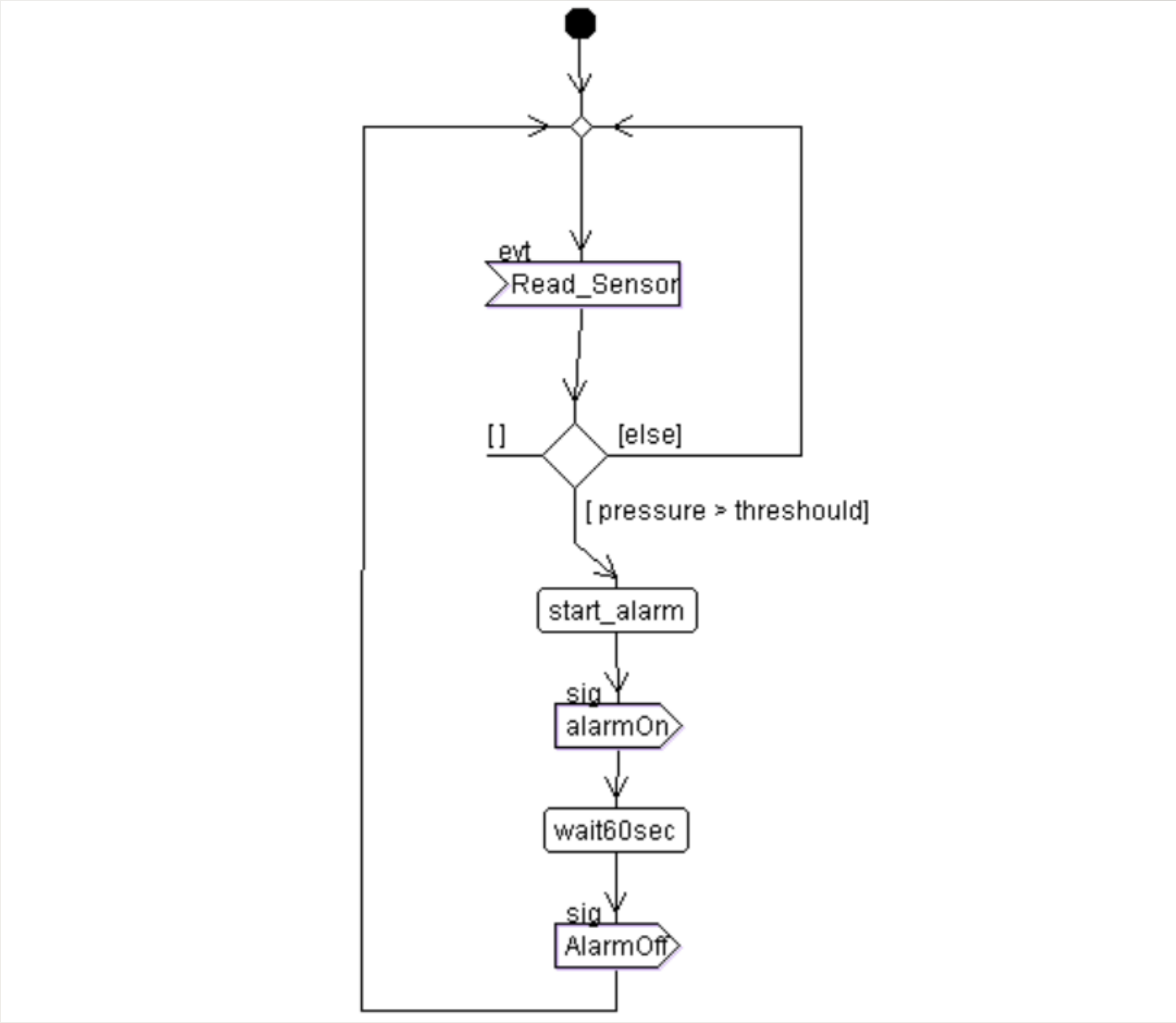
Requirement Diagram



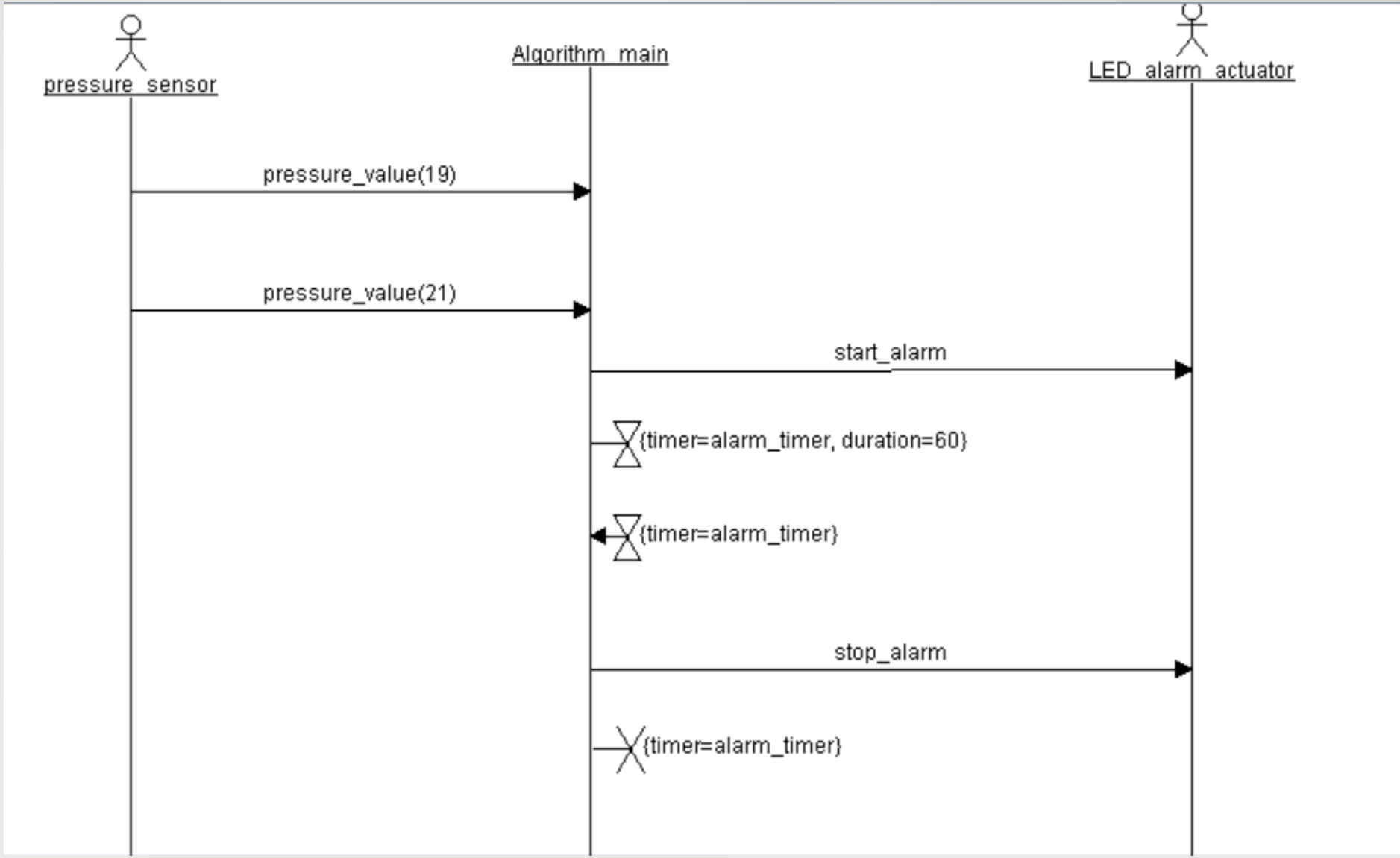
Use Case Diagram



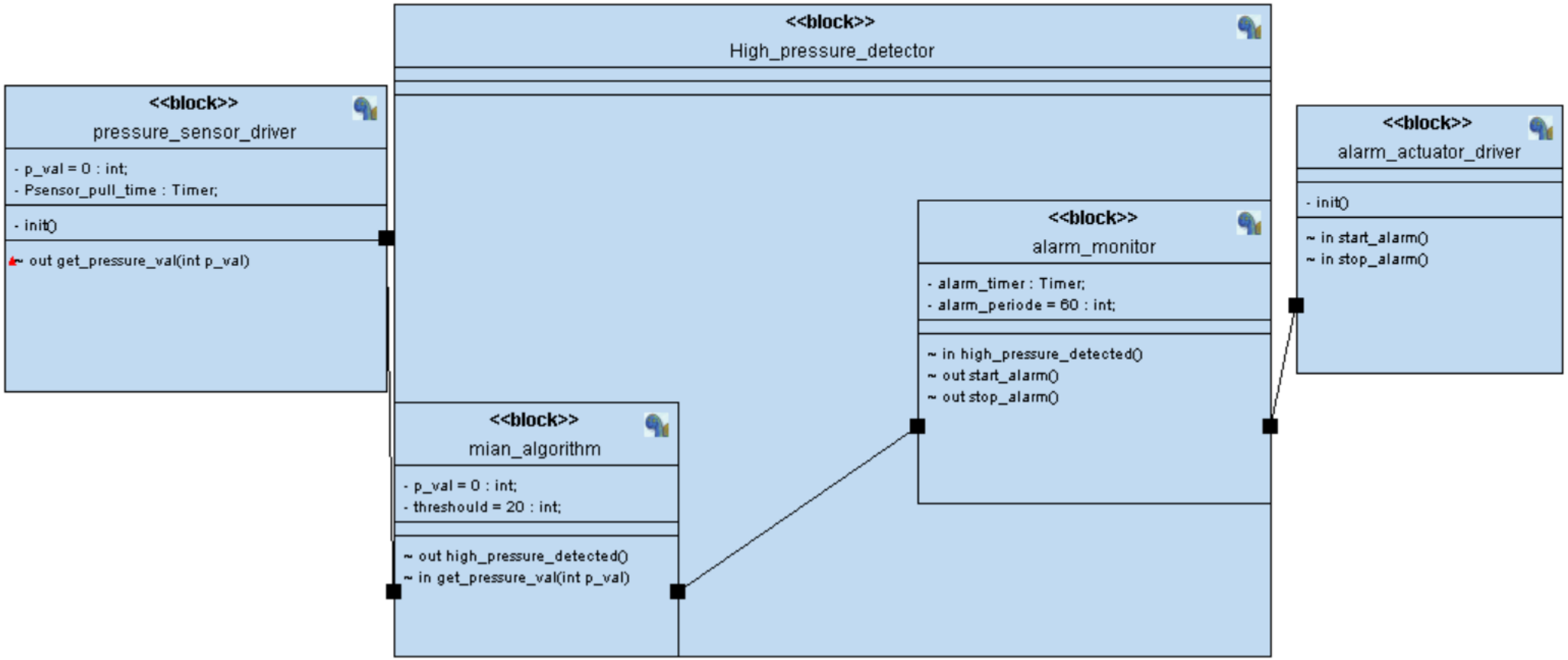
Activity Diagram



Sequence Diagram

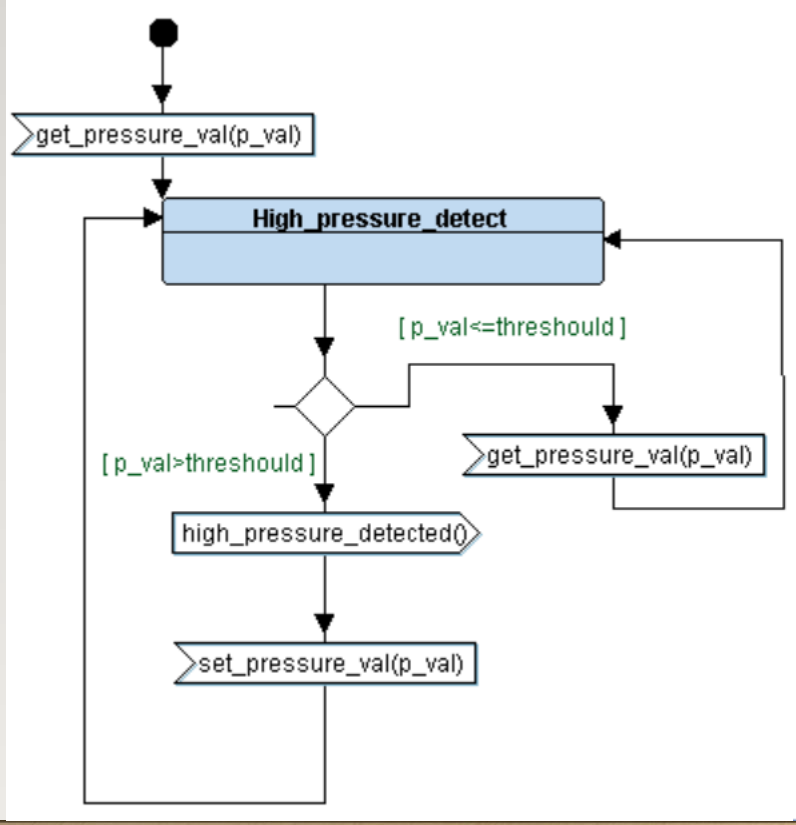


Design Modules

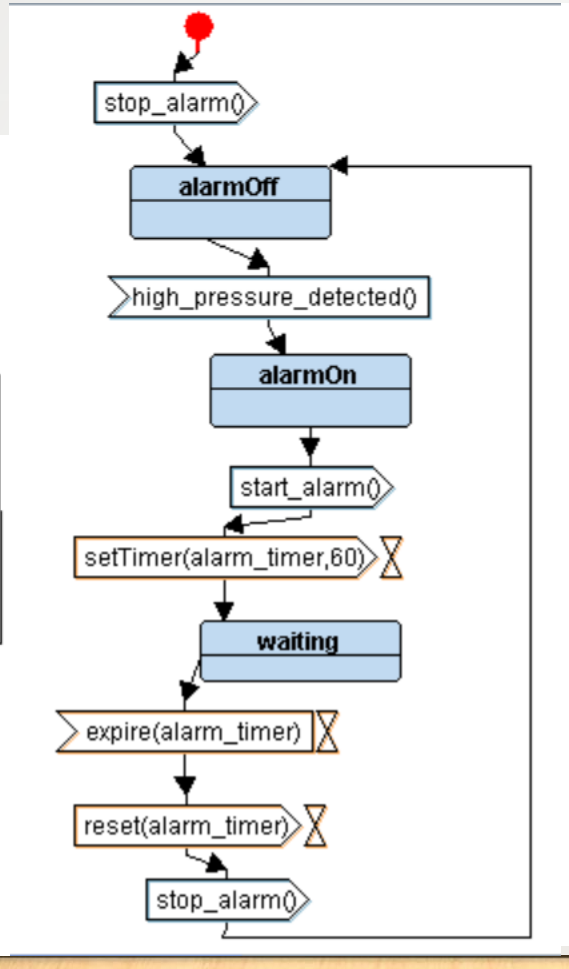


The state machines

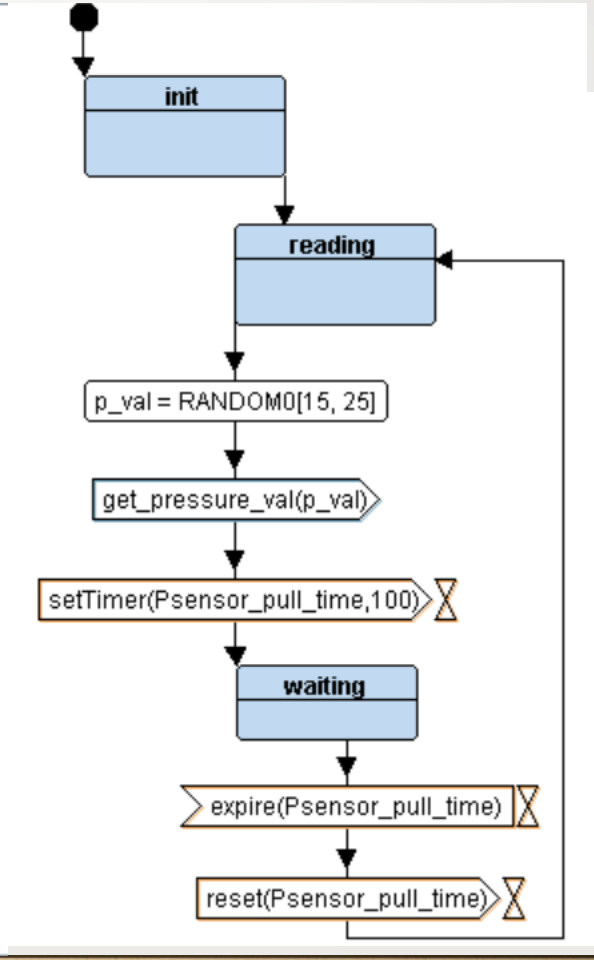
main



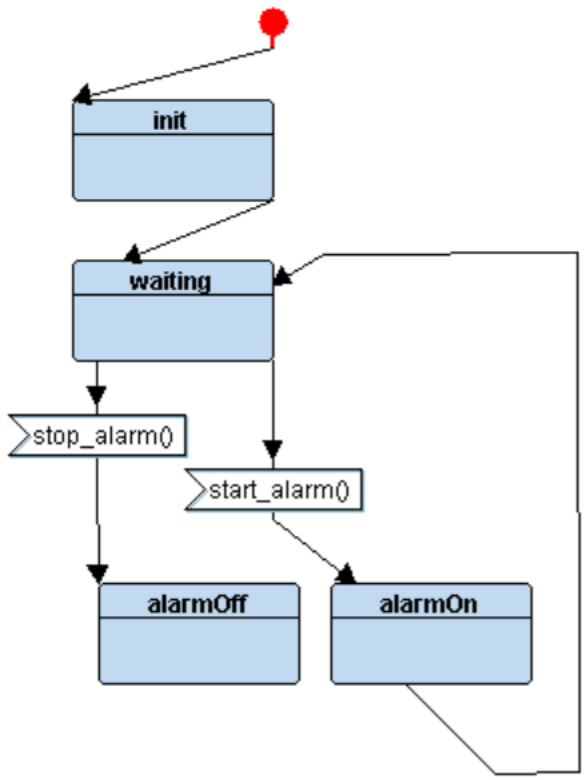
monitor



The sensor



the alarm



Programming :::: Startup.c

```c
/* done by Omar Ayman at  ___5:46___    ___09/30/2022___*/
/*Libraries and MACROS*/
#include"stdint.h"
/*prototypes*/
void Default_handler(void);
void Reset_handler(void);
void NMI_handler(void)__attribute__((weak,alias("Default_handler")));;
void HardFault_handler(void)__attribute__((weak,alias("Default_handler")));;
void MemManage_handler(void)__attribute__((weak,alias("Default_handler")));;
void BusFault_handler(void)__attribute__((weak,alias("Default_handler")));;
void UsageFault_handler(void)__attribute__((weak,alias("Default_handler")));;
extern int main(void);
/*pointers to transfer .data from flash to ram and to initialize .bss in ram*/
extern uint32_t  _E_text;
extern uint32_t _S_DATA,_E_DATA;
extern uint32_t _S_BSS,_E_BSS;
extern uint32_t _stack_top;
/*Vector Table*/
uint32_t Vectors[] __attribute__((section(".vectors")))=
{
        (uint32_t)&_stack_top,
        (uint32_t)Reset_handler,
        (uint32_t)NMI_handler,
        (uint32_t)HardFault_handler,
        (uint32_t)MemManage_handler,
        (uint32_t)BusFault_handler,
        (uint32_t)UsageFault_handler,
};
```

```c
/*The definitions of handlers*/
void Reset_handler(void)
        {

        uint32_t DataSize = (uint8_t*)&_E_DATA - (uint8_t*)&_S_DATA;
        uint32_t BssSize =  (uint8_t*)&_E_BSS  - (uint8_t*)&_S_BSS;

        uint8_t *P_src = (uint8_t*)&_E_text;
        uint8_t *P_dst = (uint8_t*)&_S_DATA;
        uint8_t *Pbss = (uint8_t*)&_S_BSS;

        for(uint32_t i = 0 ;i<DataSize;i++)
        {
        *((uint8_t*)P_dst++) = *((uint8_t*)P_src++);
              }
        for(uint32_t i = 0 ;i<BssSize;i++)
        {
              *((uint8_t*)Pbss++) = (uint8_t)0;
        }
              main();
        }
void Default_handler(void)      {Reset_handler();}
```

Programming :::: LinkerScript.ld & makeFile

```
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)
            . = ALIGN(4) ;
            _E_DATA = . ;
    }> sram AT> flash

    .bss : {
            _S_BSS = . ;
            *(.bss)
            _E_BSS = . ;
            . = ALIGN(4) ;
            . = . + 0x1000 ;
            _stack_top = . ;
    }> sram
}
```

```
#@copyright : OMAR
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 = cortexM3

all: $(project_name).bin
    @echo "M   a   k   e    i s    d   o   n   e"
startup.o: startup.s
    $(CC)as.exe $(CFLAGS) $< -o $@

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

$(project_name).elf: $(OBJ) $(AsOBJ)
    $(CC)ld -T LinkerScript.ld $(LIBS) $(OBJ) $(AsOBJ) -o $@ -Map=mapFile.map

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

clean_all:
    rm  *.o *.bin *.elf *.map *.hex
clean:
    rm *.bin *.elf
```

Programming :::: bsp.h

```c
/* bsp.h  Created on: Sep 28, 2022   Author: OMAR   */
#ifndef BSP_H_
#define BSP_H_
#include "stdint.h"
#define RCC_BASE    0x40021000U
#define GPIOA_BASE  0x40010800U
#define RCC_APB2ENR  (*((volatile uint32_t  *)(RCC_BASE+0x18U)))
#define GPIOA_CRL    (*((volatile uint32_t  *)(GPIOA_BASE+0x00U)))
#define GPIOA_CRH    (*((volatile uint32_t  *)(GPIOA_BASE+0x04U)))
#define GPIOA_IDR    (*((volatile uint32_t  *)(GPIOA_BASE+0x08U)))
#define GPIOA_ODR    (*((volatile uint32_t  *)(GPIOA_BASE+0x0CU)))
#define IOPAEN (0x04U)
#define WAIT(x) for(uint64_t i =0;i<=x;i++)//wait some time
#define LEFT_LIMIT  0x8000
#define RIGHT_LIMIT 0x0100

#define GPIOA_ODR00  (1U<<0)
#define GPIOA_ODR01  (1U<<1)
#define GPIOA_ODR02  (1U<<2)
#define GPIOA_ODR03  (1U<<3)
#define GPIOA_ODR04  (1U<<4)
#define GPIOA_ODR05  (1U<<5)
#define GPIOA_ODR06  (1U<<6)
#define GPIOA_ODR07  (1U<<7)
#define GPIOA_ODR08  (1U<<8)
#define GPIOA_ODR09  (1U<<9)
#define GPIOA_ODR10 (1U<<10)
#define GPIOA_ODR11 (1U<<11)
#define GPIOA_ODR12 (1U<<12)
#define GPIOA_ODR13 (1U<<13)
#define GPIOA_ODR14 (1U<<14)
#define GPIOA_ODR15 (1U<<15)
```

```c
typedef union{
    uint32_t volatile GPIOA_ODR_REG;
    struct{
        uint32_t volatile ODR00 :1;
        uint32_t volatile ODR01 :1;
        uint32_t volatile ODR02 :1;
        uint32_t volatile ODR03 :1;
        uint32_t volatile ODR04 :1;
        uint32_t volatile ODR05 :1;
        uint32_t volatile ODR06 :1;
        uint32_t volatile ODR07 :1;
        uint32_t volatile ODR08 :1;
        uint32_t volatile ODR09 :1;
        uint32_t volatile ODR10 :1;
        uint32_t volatile ODR11 :1;
        uint32_t volatile ODR12 :1;
        uint32_t volatile ODR13 :1;
        uint32_t volatile ODR14 :1;
        uint32_t volatile ODR15 :1;
        uint32_t volatile reserved:16;
    }ODR;}GPIOA_ODR_t;
//volatile GPIOA_ODR_t * GPIOA_ODR_REG = ((volatile GPIOA_ODR_t *)(GPIOA_BASE+0x0CU));
typedef union{
    uint32_t volatile GPIOA_IDR_REG;
    struct{
        uint32_t volatile IDR00 :1;
        uint32_t volatile IDR01 :1;
        uint32_t volatile IDR02 :1;
        uint32_t volatile IDR03 :1;
        uint32_t volatile IDR04 :1;
        uint32_t volatile IDR05 :1;
        uint32_t volatile IDR06 :1;
        uint32_t volatile IDR07 :1;
        uint32_t volatile IDR08 :1;
        uint32_t volatile IDR09 :1;
        uint32_t volatile IDR10 :1;
        uint32_t volatile IDR11 :1;
        uint32_t volatile IDR12 :1;
        uint32_t volatile IDR13 :1;
        uint32_t volatile IDR14 :1;
        uint32_t volatile IDR15 :1;
        uint32_t volatile reserved:16;
    }IDR;}GPIOA_IDR_t;
//volatile GPIOA_IDR_t * GPIOA_IDR_REG = ((volatile GPIOA_IDR_t *)(GPIOA_BASE+0x08U));
#endif /* BSP_H_ */
```

Programming :::: mainALG.c & pressure.c & pressure.h

```c
#include "pressure_sensor.h"
#include "alarm.h"
#include "alarm_monitor.h"
#include "stdint.h"
static uint32_t p_val;
static const uint32_t threshould = 0x14;

int main(void)
{
    /* Modules initializations*/
    init_sensor();
    init_alarm();
    /* Loop forever */
    for(;;)
    {
    p_val = get_pressure_val();
    if(p_val > threshould) {high_pressure_detected();}
    }
    return 0;
}
```

```c
#include"bsp.h"
#include "pressure_sensor.h"

static uint32_t p_val;
static uint32_t Psensor_pull_time ;
void init_sensor(void)
{
    //Enable clock of GPIOA
    RCC_APB2ENR |=IOPAEN;
    //wait some time
    WAIT(5000);
    //clear all bits 0~7
    GPIOA_CRL &= ~(0xffffffffU);
    //digital inputs
    GPIOA_CRL |=(0x88888888U);
}

uint32_t get_pressure_val(void)
{
    // pull time [do nothing] about 20 second
    while(Psensor_pull_time <=100){WAIT(30000);Psensor_pull_time++;}
    Psensor_pull_time=0;
    //start reading ...
    p_val = GPIOA_IDR;
    return p_val;
}
```

```c
#ifndef PRESSURE_SENSOR_H_
#define PRESSURE_SENSOR_H_

#include "stdint.h"
void init_sensor(void);
uint32_t get_pressure_val(void);

#endif /* PRESSURE_SENSOR_H_ */
```

Programming :::: alarm.c & alarm.h & alarm_monitor.c & alarm_monitor.h

```c
#include"bsp.h"
#include "alarm.h"

void init_alarm()
{
    //Enable clock of GPIOA
    RCC_APB2ENR |=IOPAEN;
    //wait some time
    WAIT(5000);
    GPIOA_CRH &= ~(0xffffffffU);
    //General purpose output Open-drain
    GPIOA_CRH |=(0x22222222U);
    //clear GPIO(bit15~bit08)
    GPIOA_ODR&=~0xff00;
}

void start_alarm(void)
{
    uint8_t m1;
    //LED ON
    GPIOA_ODR |= GPIOA_ODR09;
    //generate teet teet sound on a speaker
    for(m1=0;m1<150;m1++){GPIOA_ODR^= GPIOA_ODR08;WAIT(250);}
    for(m1=0;m1<150;m1++){GPIOA_ODR^= GPIOA_ODR08;WAIT(125);}
}
void stop_alarm(void)
{
    //LED OFF
    GPIOA_ODR &= ~GPIOA_ODR09;
}
```

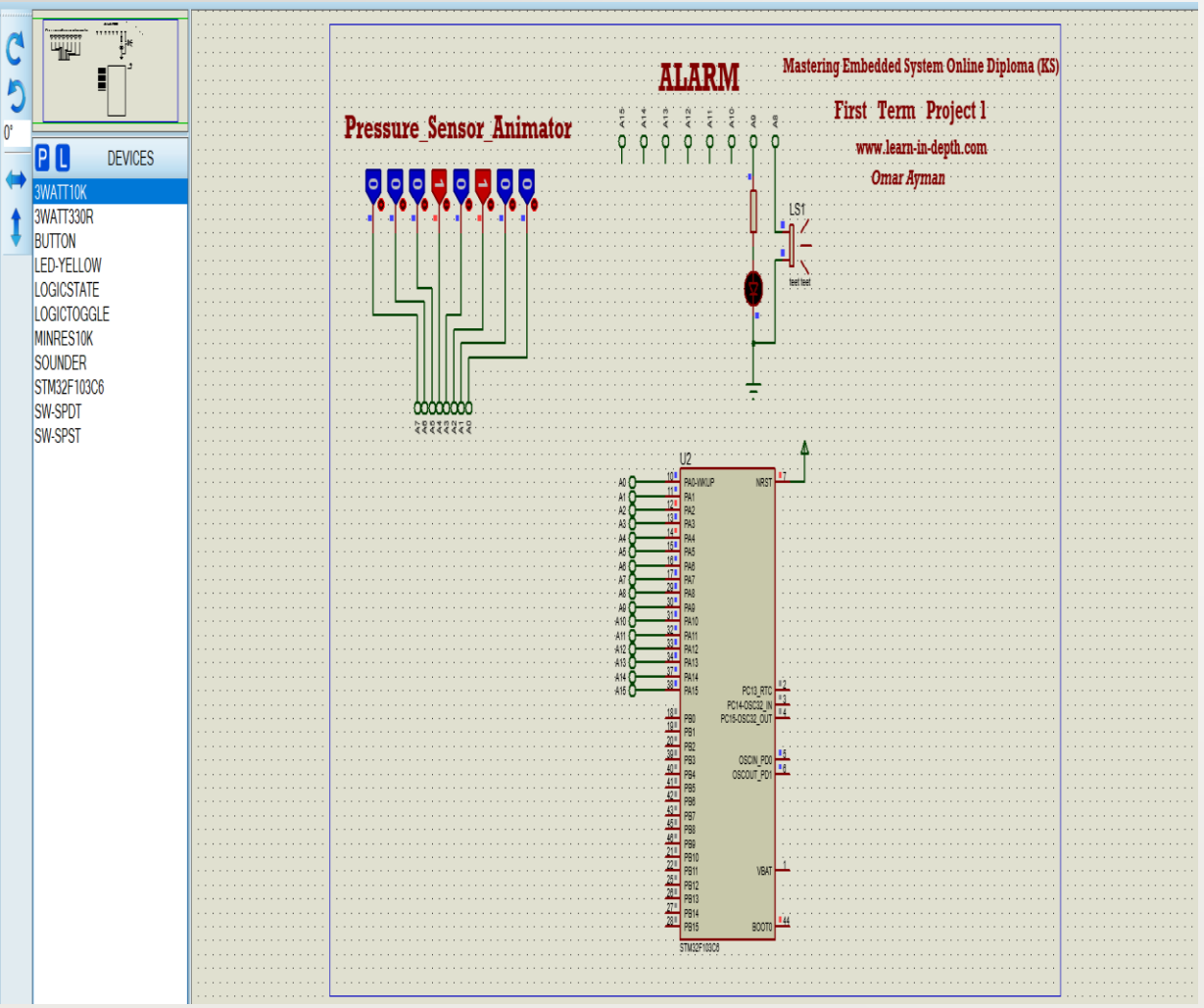
```c
#ifndef ALARM_H_
#define ALARM_H_

void init_alarm();
#endif /* ALARM_H_ */
```

```c
#include "alarm_monitor.h"
#include "alarm.h"

static uint32_t alarm_timer;
static const uint32_t alarm_period = 60;//60 counts are roughly equal to 20 seconds
uint8_t high_pressure_detected(void)
{
    while(alarm_timer<=alarm_period)
    {
    start_alarm();
    alarm_timer++;
    }
    alarm_timer = 0 ;
    stop_alarm();
}
```

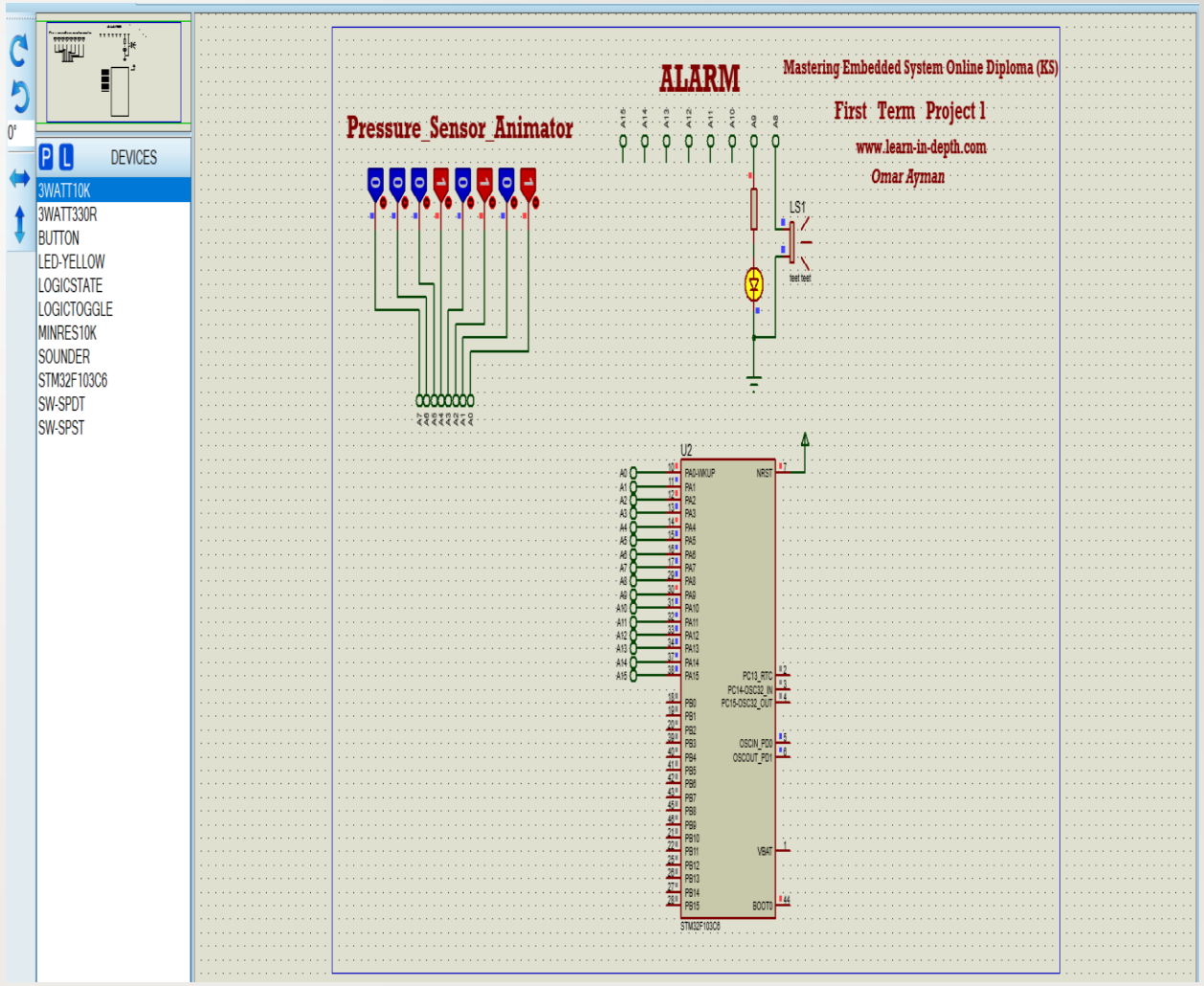
```c
#ifndef ALARM_MONITOR_H_
#define ALARM_MONITOR_H_

#include "stdint.h"
void start_alarm(void);
void stop_alarm(void);

uint8_t high_pressure_detected(void);

#endif /* ALARM_MONITOR_H_ */
```

Test cases

Pressure = 20 :::: alarm is turned off



Pressure = 21 :::: alarm is turned on



Programming :::: simulation in proteus