Report

First term Project 1

High Pressure Detector

Target:

- In this report we will discuss how to design and implement the project from meeting with the customer(virtually) and passing through all designing processes.
- We will implement the code from scratch respecting to modularity with our own startup.c, linker script and make file.
- We will use Ttool program to describe our system

Steps:

1- Case study

we have an airplane cabin and we want to detect if the pressure exceeds 20 bar an alarm will be activated.

a- Assumptions

after talking to the customer, we should define our assumptions to inform the customer what I will not do and if he want to make any of these assumptions as a requirement he should be informed that is will increase cost,

for example

- the controller setup and shutdown procedures are not modeled

- the pressure sensor never fails.
- the alarm never fails.
- the controller never faces power cut.

b- Versioning

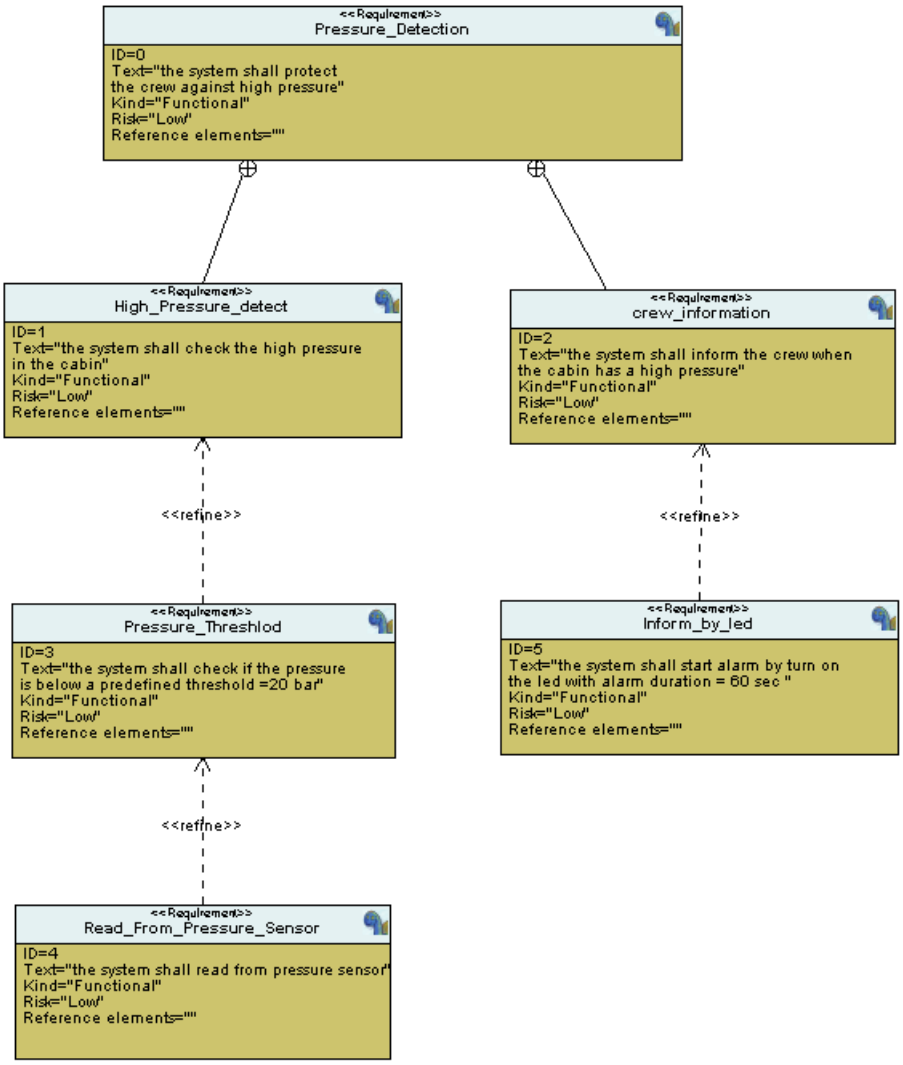
if we want to make different versions according to the time, we to discuss that in this section and inform the customer.

2-Method

in this stage we will determine software development life cycle (SDLC) and software testing life cycle (STLC)
according to the suitable way for our company we choose
V-cycle.

3-Requirement diagram

After the previous steps and many meetings with the customer and And our teams that produced a final vision about the customer requirements and we added the related physical requirements
We will describe this vision using requirement diagram as shown below:



4- Design space exploration / partitioning

In this part we should find the best option to divide the software into small chunks of code with respect to the target machine will be selected and how many SOC we will use, this process requires well-experienced people and very advanced tools to choose the best option, but in our case the project is simple and we will use **Board:** STM32F103C6 based on ARM-cortex-m3.

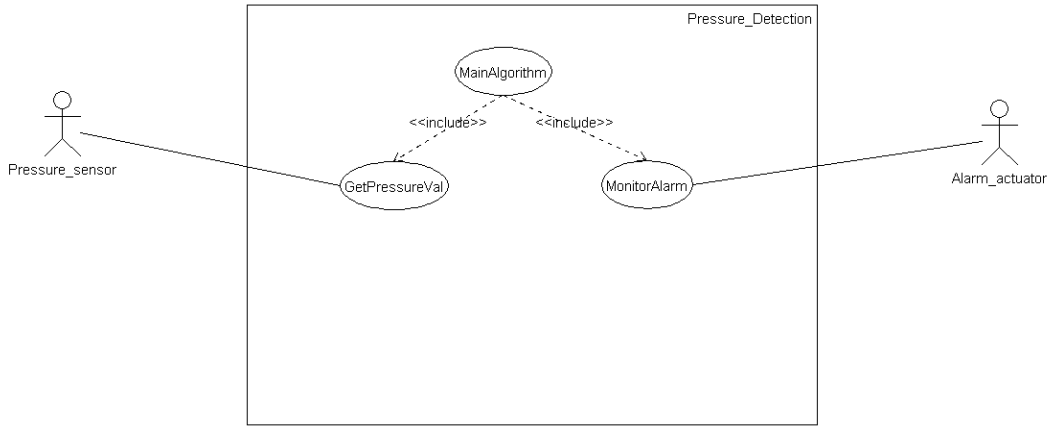
5- System analysis

in this stage we will understand what client wants and the main functionalities of the system to be designed without details.

There are three main things should be described in this stage

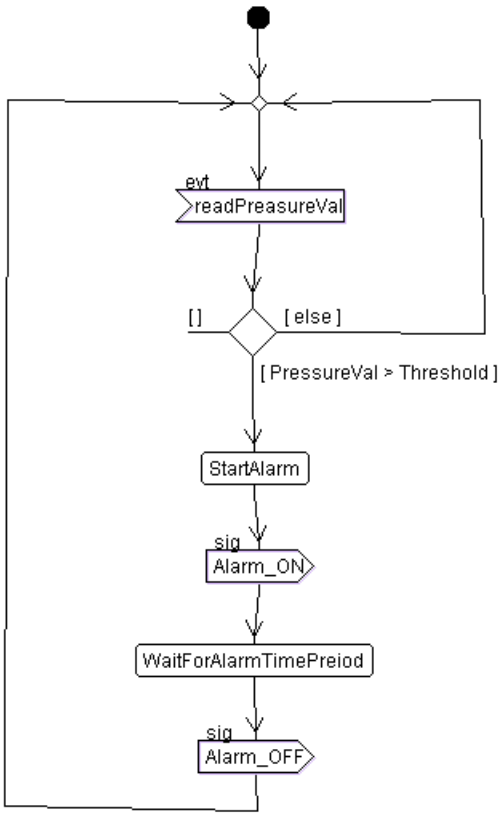
a- Use case diagram

use case diagram describes the boundary of the system, main functions and the interaction between the system and the outer systems or actors like sensors or users etc.



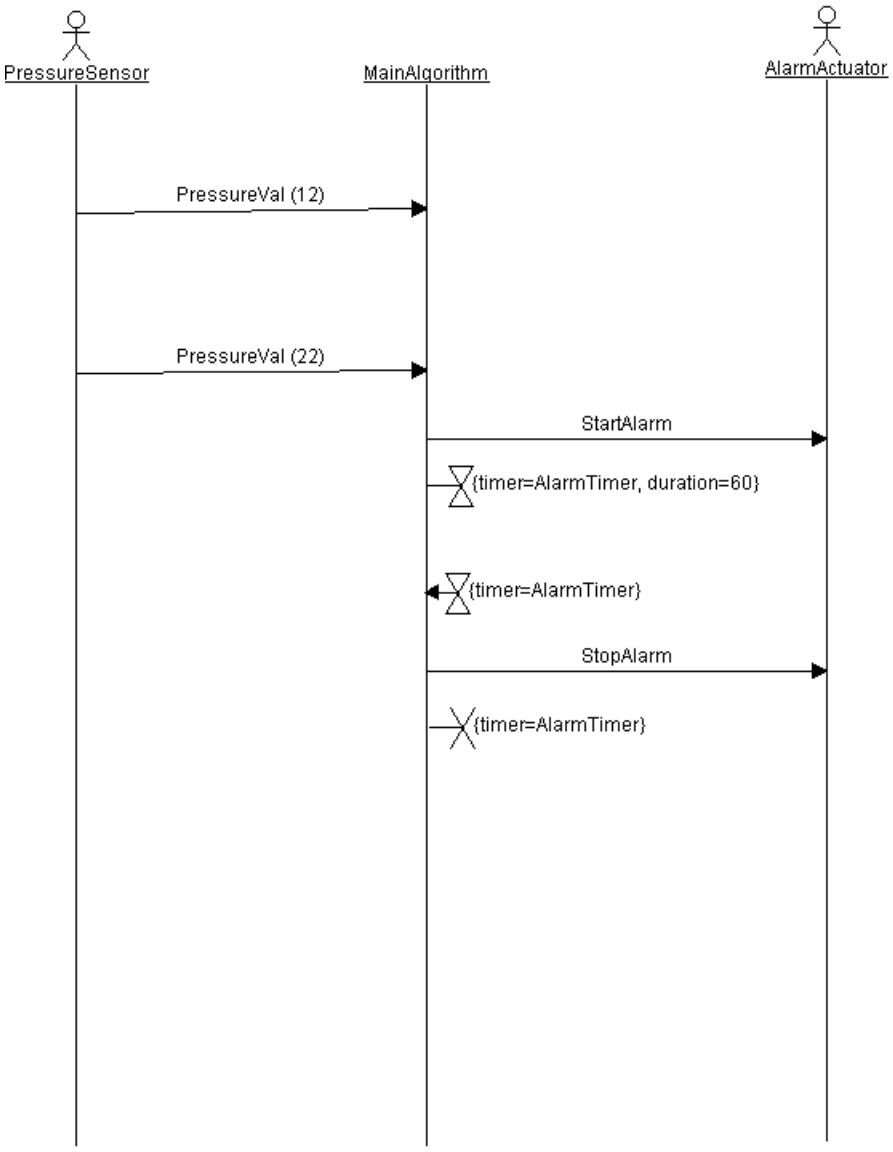
b- Activity diagram

Activity diagram describes the workflow behavior of the system



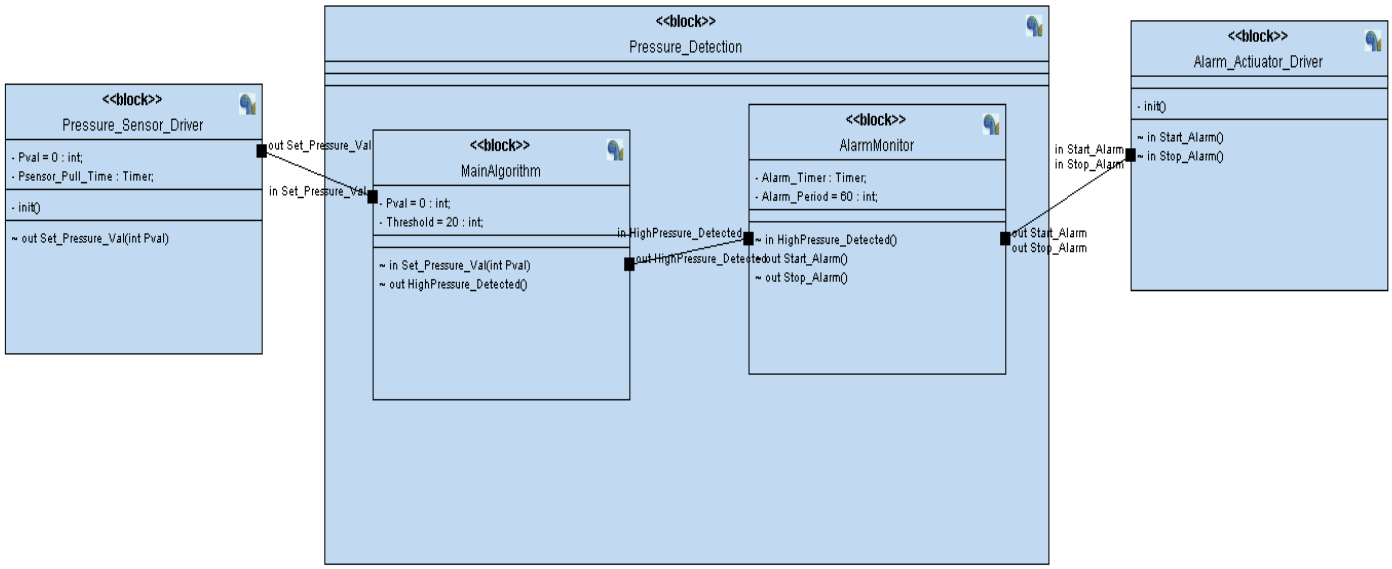
c- Sequence diagram

An interact diagram that details how operations are carried out according time.



6- System design

This stage helps the developer to implement the code by divide the software into blocks or modules and this step is not allowed for user to be shown because it describes how to implement the code.

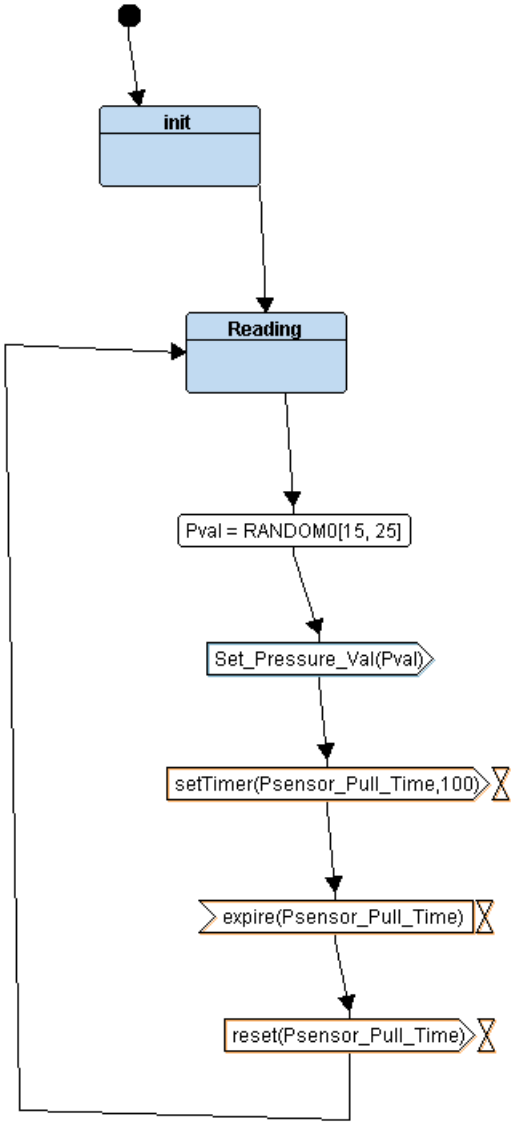


We have 4 blocks and the main block each block has its own variables, methods and link to another block called signals.

Let's see each block state machine:

1- Pressure sensor driver

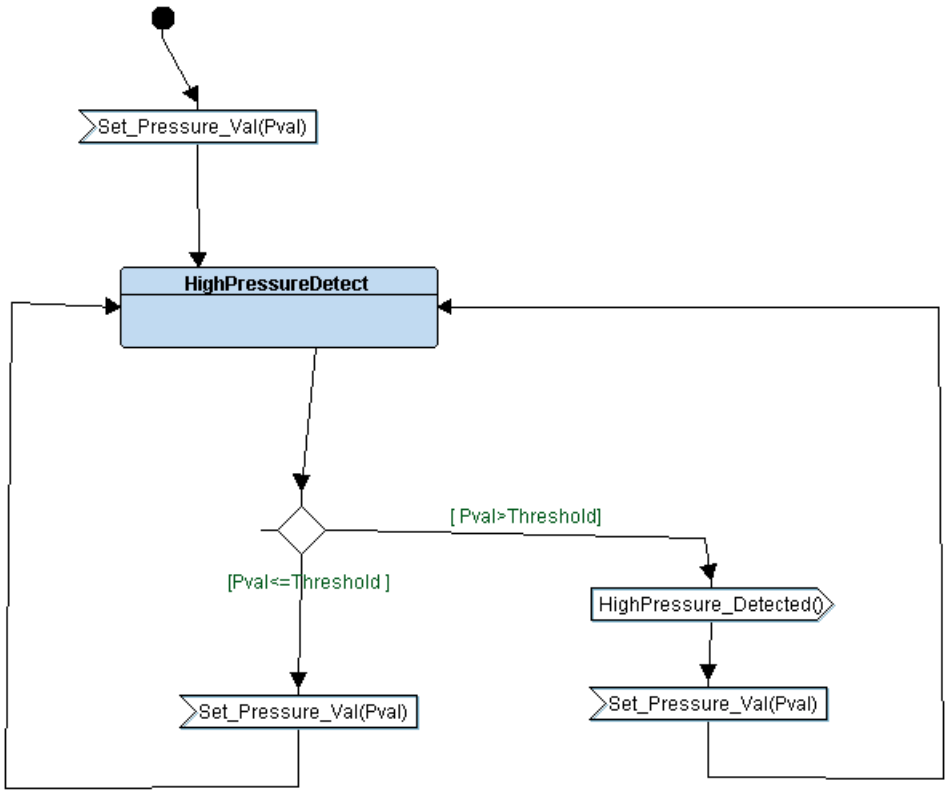
Starts with init state and then goes to reading state to call sensor driver to read the pressure value.



2- Main algorithm block

After receiving signal with measured pressure value, it will jump to

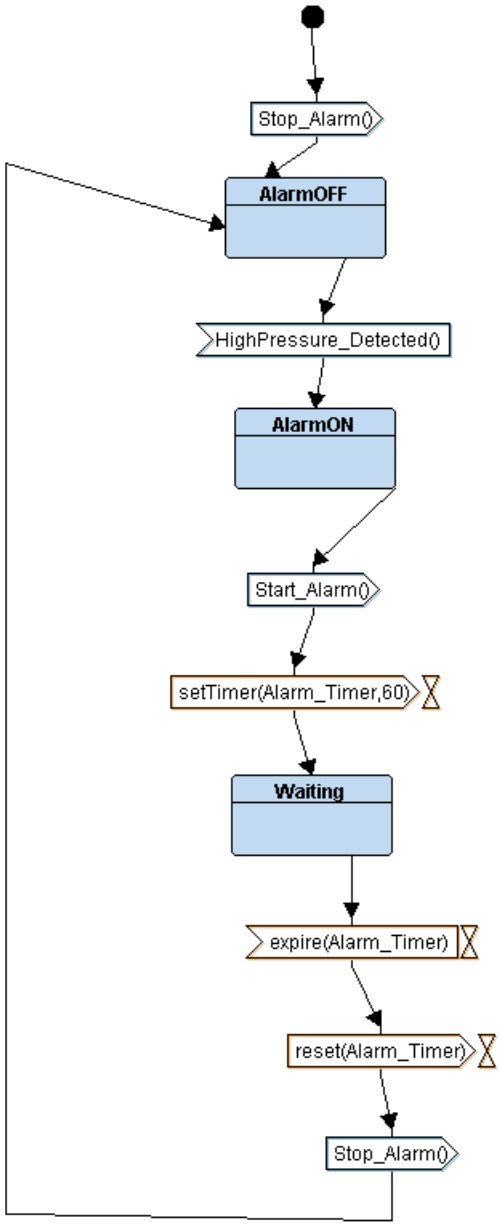
” high pressure detects “state to check if the pressure value exceeds the threshold (20 bar) or not.



3- Alarm monitor block

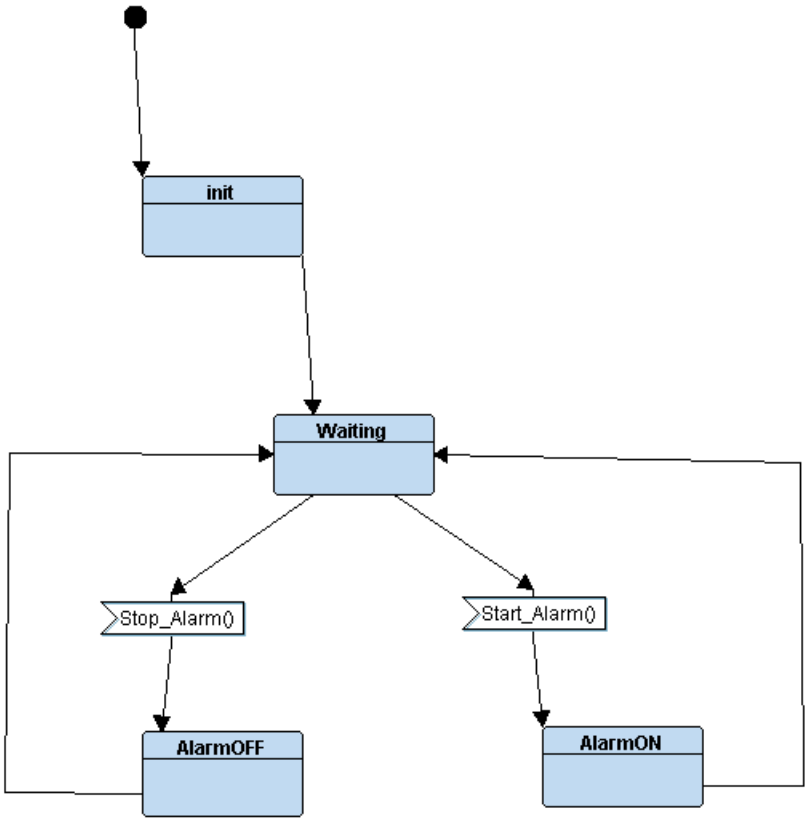
this block will send signal to stop alarm and go to “alarm off” state till it receives signal that high pressure detected it will go to “alarm on”

state that will activate alarm for 60 seconds and back to the “alarm off” state again



4- Alarm actuator block

this block will deal with MC drivers and initialize the GPIO for activate pins of the alarm.



Implementation

After designing our system lets translate these blocks into c code and make our own startup.c and linker script and generic make file.

1- Startup.c

Arm-cortex-m based processors have a beautiful feature that allows us to write a start up using c programming language by assign the stack pointer address into the entry point.
So here with help of linker script we defined the boundry of memories and the mechanism of coping .data section from rom to ram and initialize the .bss section in ram and define interrupt vector table according to specs.

```c
#include <stdint.h>
extern int main(void);
extern unsigned int _E_text ;
extern unsigned int _S_data ;
extern unsigned int _E_data ;
extern unsigned int _S_bss ;
extern unsigned int _E_bss ;
extern unsigned int _stack_top ;
void Reset_Handler()
{    int i ;
    //we need to copy data section from flash to ram
    unsigned int DATA_size = (unsigned char*)&_E_data - (unsigned char*)&_S_data; // casting to tell that is add of char to copy byte by byte
    unsigned char* p_src = (unsigned char*)&_E_text ;
    unsigned char* p_dst = (unsigned char*)&_S_data ;
    for (i=0; i< DATA_size; i++)
    {
        *((unsigned char*)p_dst++) = *((unsigned char*)p_src++);
    }
    // init .bss section in sram = 0
    unsigned int BSS_size = (unsigned char*)&_E_bss - (unsigned char*)&_S_bss;
    p_dst = (unsigned char*)&_S_bss;
    for (i=0; i< BSS_size; i++)
    {
        *((unsigned char*)p_dst++) = (unsigned char)0;
    }

    // jump main
    main();
}
void Default_handler()
{
    Reset_Handler();
}
void NMI_Handler() __attribute__ ((weak,alias("Default_handler")));;
void H_fault_Handler() __attribute__ ((weak,alias("Default_handler")));;
void MM_Fault_Handler() __attribute__ ((weak,alias("Default_handler")));;
void Bus_Fault() __attribute__ ((weak,alias("Default_handler")));;
void Usage_Fault_Handler() __attribute__ ((weak,alias("Default_handler")));;

uint32_t vectors[] __attribute__((section(".vectors")))= {
    (uint32_t) &_stack_top,
    (uint32_t) &Reset_Handler,
    (uint32_t) &NMI_Handler,
    (uint32_t) &H_fault_Handler,
    (uint32_t) &MM_Fault_Handler,
    (uint32_t) &Bus_Fault,
    (uint32_t) &Usage_Fault_Handler
};
```

2- Linker script

```
/*linker_script cortex-m3
Eng.shady mamdouh
*/
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*)
                . = ALIGN(4) ;
                _E_bss = . ;
                . = . + 0x1000;
                _stack_top = . ;

           }> sram
}
```

3- Make file

```
#@copyrights shady mamdouh
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=First_term_project1_Pressure_Detector
all: $(project_name).bin
    @echo "all build is done"
%.o: %.c
    $(CC)gcc.exe -c $(CFLAGS) $(INCS) $< -o $@
%.o: %.s
    $(CC)as.exe $(CFLAGS) $< -o $@
$(project_name).elf: $(OBJ) $(ASOBJ)
    $(CC)ld.exe -T linker_script.ld -Map=map_file.map $(OBJ) $(ASOBJ) -o $@
$(project_name).bin: $(project_name).elf
    $(CC)objcopy.exe -O  binary $< $@
$(project_name).hex: $(project_name).elf
    $(CC)objcopy.exe -O  binary $< $@
clean_all:
    rm *.o *.elf *.bin
clean:
    rm *.bin *.elf

```

4- Main

```
/*
 * main.c
 *
 *  Created on: Feb 21, 2021
 *      Author: Shady mamdouh
 */
#include "Pressure_Sensor_Driver.h"
#include "MainAlgorithm.h"
#include "AlarmMonitor.h"
#include "driver.h"
int Pval;
int main(void)
{    // pointer to functios initializtion
    PS_ptr=PS_init;
    MA_ptr=High_pressure_state;
    AM_ptr=AlarmOff_state;
    while(1)
        {
        (*PS_ptr)(); // refers to Pressure_Sensor_Driver module
        (*MA_ptr)(); // refers to MainAlgorithm module
        (*AM_ptr)(); // refers to AlarmMonitor module
//Delay(10000);   // for embedded delay
        }
}
```

5- **Pressure sensor driver module**

```c
/*
 * Pressure_Sensor_Driver.c
 *
 *  Created on: Feb 21, 2021
 *      Author: Shady mamdouh
 */

#include"driver.h"
#include"Pressure_Sensor_Driver.h"
extern int Pval;
void (*PS_ptr)();
extern void set_pressureValue();
void PS_init()
{
    GPIO_INITIALIZATION ();
    reading_state();
}

void reading_state()
{ PS_ptr=reading_state;
    Pval=getPressureVal();
    set_pressureValue();

}
```

```c
/*
 * Pressure_Sensor_Driver.h
 *
 *  Created on: Feb 21, 2021
 *      Author: Shady mamdouh
 */

#ifndef PRESSURE_SENSOR_DRIVER_H_
#define PRESSURE_SENSOR_DRIVER_H_

void PS_init();
void reading_state();
extern void (*PS_ptr)();

#endif /* PRESSURE_SENSOR_DRIVER_H_ */
```

6- **Main algorithm module**

```c
/*
 * MainAlgorithm.h
 *
 *  Created on: Feb 21, 2021
 *      Author: Shady mamdouh
 */

#ifndef MAINALGORITHM_H_
#define MAINALGORITHM_H_
void High_pressure_state(void);
void set_pressureValue(void);
void High_pressure_state(void);
extern void (*MA_ptr)();
extern void (*AM_ptr)();
extern void High_pressure_detect();
#endif /* MAINALGORITHM_H_ */
```

```c
/*
 *  MainAlgorithm.c
 *
 *  Created on: Feb 21, 2021
 *      Author: Shady mamdouh
 */
#include"MainAlgorithm.h"
void High_pressure_state(void);


extern int Pval;
void (*MA_ptr)();
#define Threshold 20
void set_pressureValue(void)
{
MA_ptr=High_pressure_state;
}
void High_pressure_state(void)
 {
    if(Pval <= Threshold)
        {
        MA_ptr=High_pressure_state;

        }
    else
        {
        High_pressure_detect();
        MA_ptr=High_pressure_state;
        }
 }
```

7- Alarm monitor module

```c
/*
 * AlarmMonitor.c
 *
 *  Created on: Feb 21, 2021
 *      Author: shady mamdouh
 */
#include "AlarmMonitor.h"
int x;
void (*AM_ptr)();
void High_pressure_detect()
{
    AM_ptr=AlarmOn_state;

}
void AlarmOff_state()
{
stopAlarm();
AM_ptr=AlarmOff_state;
}
void AlarmOn_state()
{   for(x=0;x<500;x++) // period of blinking
    {startAlarm();
    Delay(10000);
    stopAlarm();
    Delay(10000);
    }
    AM_ptr=AlarmOff_state;

}
```

```c
/*
 * AlarmMonitor.h
 *
 *  Created on: Feb 21, 2021
 *      Author: Shady mamdouh
 */


#ifndef ALARMMONITOR_H_
#define ALARMMONITOR_H_
#include"AlarmActuatorDriver.h"
#include"driver.h"
void High_pressure_detect();
void AlarmOff_state();
void AlarmOn_state();
extern void stopAlarm();
extern void startAlarm();
extern void (*AM_ptr)();
#endif /* ALARMMONITOR_H_ */
```

8-**Alarm actuator module**

```c
/*
 * AlarmActuatorDriver.c
 *
 *  Created on: Feb 21, 2021
 *      Author: Shady mamdouh
 */
#include"AlarmActuatorDriver.h"
#include "driver.h"

void startAlarm()
{
    Set_Alarm_actuator(1);

}
void stopAlarm()
{
    Set_Alarm_actuator(0);

}
```

```c
/*
 * AlarmActuatorDriver.h
 *
 *  Created on: Feb 21, 2021
 *      Author: Shady mamdouh
 */

#ifndef ALARMACTUATORDRIVER_H_
#define ALARMACTUATORDRIVER_H_

#include"driver.h"
void stopAlarm();
void startAlarm();

#endif /* ALARMACTUATORDRIVER_H_ */
```

9-**Driver module**

```c
/*
 * driver.c
 *
 *  Created on: Feb 21, 2021
 *      Author: Shady mamdouh
 */
#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 == 1){
        SET_BIT(GPIOA_ODR,13);
    }
    else if (i == 0){
        RESET_BIT(GPIOA_ODR,13);
    }
}

void GPIO_INITIALIZATION (){
    SET_BIT(APB2ENR, 2);
    GPIOA_CRL &= 0xFF0FFFFF;
    GPIOA_CRL |= 0x00000000;
    GPIOA_CRH &= 0xFF0FFFFF;
    GPIOA_CRH |= 0x22222222;
}
```

```c
/*
 * driver.h
 *
 *  Created on: Feb 21, 2021
 *      Author: SHADY MAMDOUH
 */

#ifndef DRIVER_H_
#define DRIVER_H_

#include <stdint.h>
#include <stdio.h>

#define SET_BIT(ADDRESS,BIT)    ADDRESS |=  (1<<BIT)
#define RESET_BIT(ADDRESS,BIT) ADDRESS &= ~(1<<BIT)
#define TOGGLE_BIT(ADDRESS,BIT)  ADDRESS ^=  (1<<BIT)
#define READ_BIT(ADDRESS,BIT) ((ADDRESS) &   (1<<(BIT)))


#define GPIO_PORTA 0x40010800
#define BASE_RCC   0x40021000

#define APB2ENR   *(volatile uint32_t *)(BASE_RCC + 0x18)

#define GPIOA_CRL *(volatile uint32_t *)(GPIO_PORTA + 0x00)
#define GPIOA_CRH *(volatile uint32_t *)(GPIO_PORTA + 0X04)
#define GPIOA_IDR *(volatile uint32_t *)(GPIO_PORTA + 0x08)
#define GPIOA_ODR *(volatile uint32_t *)(GPIO_PORTA + 0x0C)


void Delay(int nCount);
int getPressureVal();
void Set_Alarm_actuator(int i);
void GPIO_INITIALIZATION ();

#endif /* DRIVER_H_ */
```

Final output