Project 1

Pressure Detection System Using STM32F

GitHub repo:

GitHub - Ziaddelsayed/Embedded_systems_online_diploma

Name: Ziad El-Sayed Ibrahim

Introduction:

- In modern embedded systems, pressure detection plays a crucial role in various applications such as industrial automation, automotive systems, and safety devices. Accurate and timely detection of pressure changes can help prevent system failures, alert users to hazardous conditions, or even trigger automated responses. This project focuses on developing a pressure detection system using the STM32F103C8 microcontroller, simulating pressure sensor inputs to demonstrate how a real-world system would function in such scenarios.

- The project utilizes several software modules to read pressure data, process it, and respond based on predefined thresholds. When the pressure exceeds the threshold, the system activates an alert by turning on an LED. By adopting a modular, state-based architecture, the system ensures reliable and continuous pressure monitoring.

- The key modules of this project include the PSENSOR, which is responsible for reading the sensor values, the MainAlg for handling the main logic and threshold comparison, the ALARM_MONITOR for monitoring dangerous pressure levels, and the ACTUATOR_DRIVER, which activates the LED indicator when needed.

- This project demonstrates the practical implementation of pressure detection in an embedded environment, showcasing the use of state machines, modular programming, and GPIO manipulation to simulate real-world scenarios.

-System Architecting/Design Sequence:

1. Case Study:

-Pressure Controller: Assumptions:

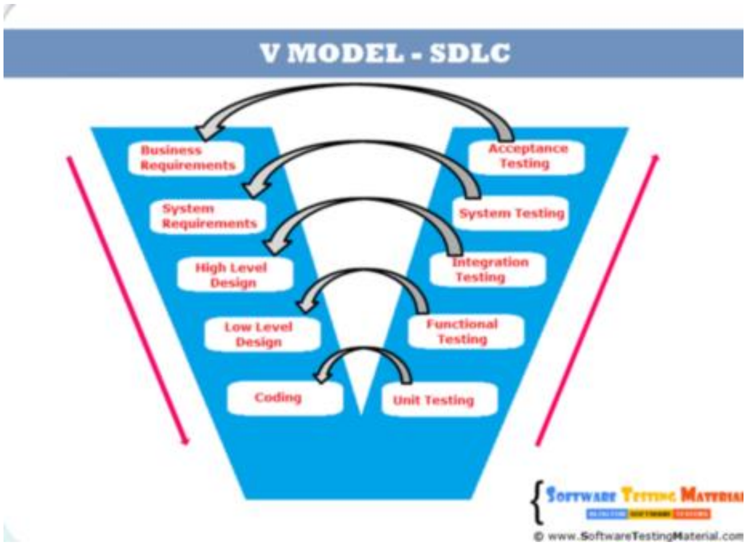
- The controller set up and shutdown procedures are not modeled.
- The controller maintenance is not modeled.
- The pressure sensor never fails.

- The alarm never fails.
- The controller never faces power cut.

-**Versioning** The ”keep track of measured value” option is not modeled in the first version of the design.

2. the Method:

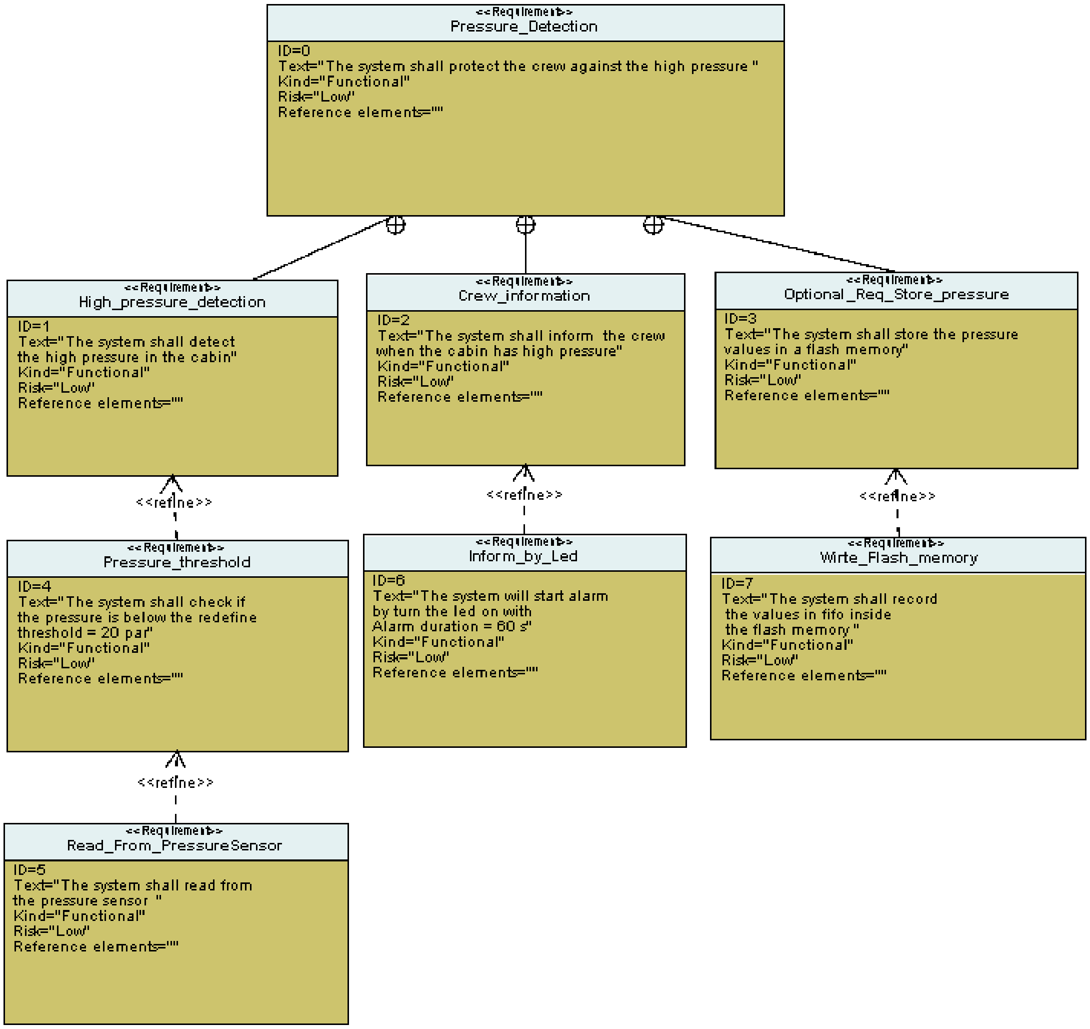
we will use V-Cycle method.



3. Requirement:

- 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.
- Optional: keeps track of the measured values.

-Requirement Diagram:



<<Requirement>>
Pressure_Detection
ID=0
Text="The system shall protect the crew against the high pressure "
Kind="Functional"
Risk="Low"
Reference elements=""

<<Requirement>>
High_pressure_detection
ID=1
Text="The system shall detect the high pressure in the cabin"
Kind="Functional"
Risk="Low"
Reference elements=""

<<Requirement>>
Crew_information
ID=2
Text="The system shall inform the crew when the cabin has high pressure"
Kind="Functional"
Risk="Low"
Reference elements=""

<<Requirement>>
Optional_Req_Store_pressure
ID=3
Text="The system shall store the pressure values in a flash memory"
Kind="Functional"
Risk="Low"
Reference elements=""

<<Requirement>>
Pressure_threshold
ID=4
Text="The system shall check if the pressure is below the redefine threshold = 20 par"
Kind="Functional"
Risk="Low"
Reference elements=""

<<Requirement>>
Inform_by_Led
ID=6
Text="The system will start alarm by turn the led on with Alarm duration = 60 s"
Kind="Functional"
Risk="Low"
Reference elements=""

<<Requirement>>
Wirte_Flash_memory
ID=7
Text="The system shall record the values in fifo inside the flash memory "
Kind="Functional"
Risk="Low"
Reference elements=""

<<Requirement>>
Read_From_PressureSensor
ID=5
Text="The system shall read from the pressure sensor "
Kind="Functional"
Risk="Low"
Reference elements=""

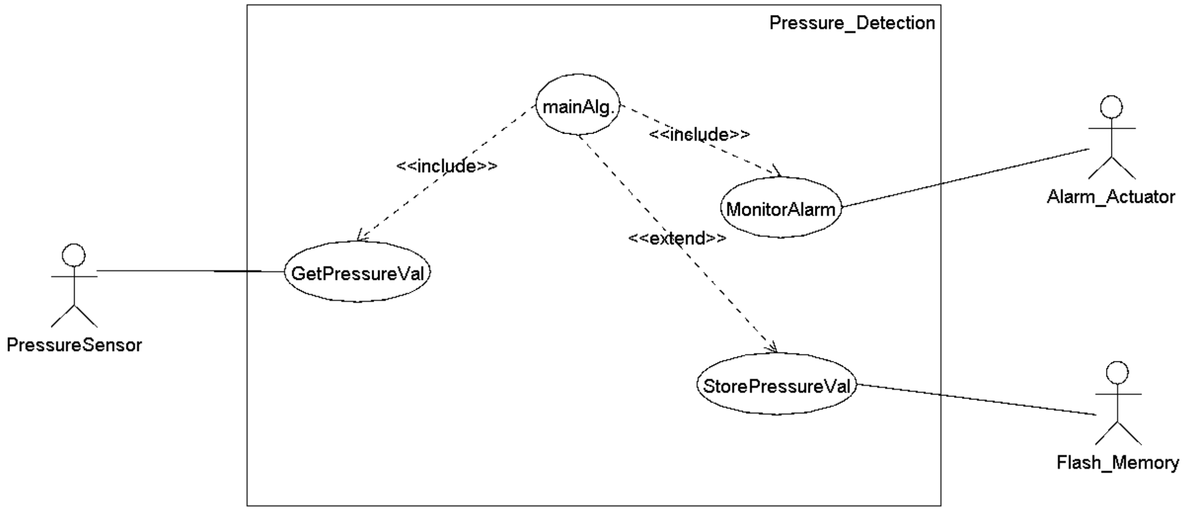
4. Space Exploration/ partitioning:

-we will use STM32F microcontroller.

5. System Analysis:

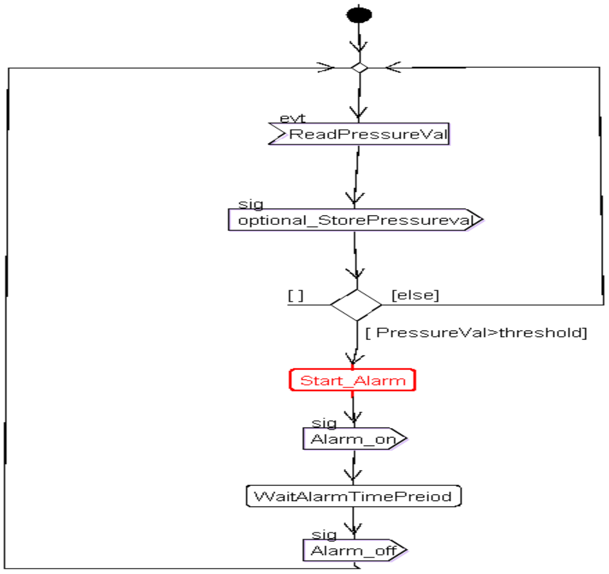
- System boundary and main functions:

  -Case Diagram:



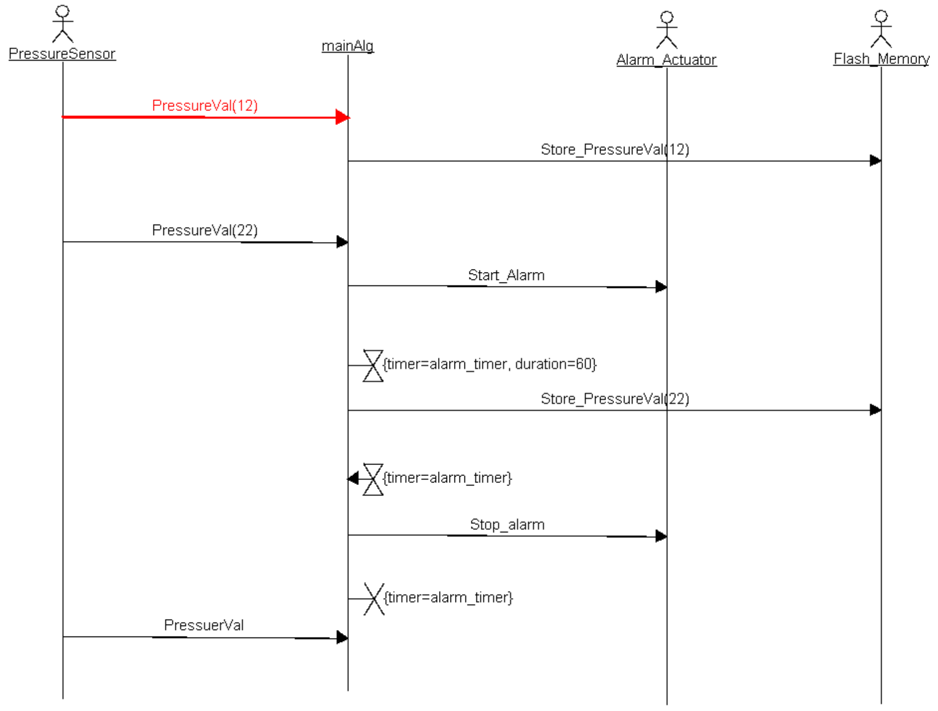
- Relations between main functions:

  -Activity Diagram:



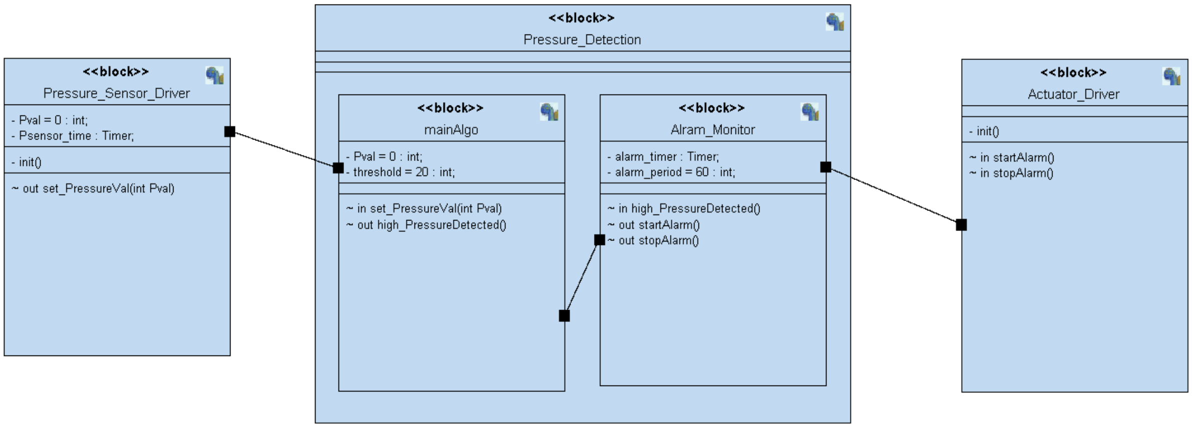
- Communications between main system entities and actors:

  -Sequence Diagram:



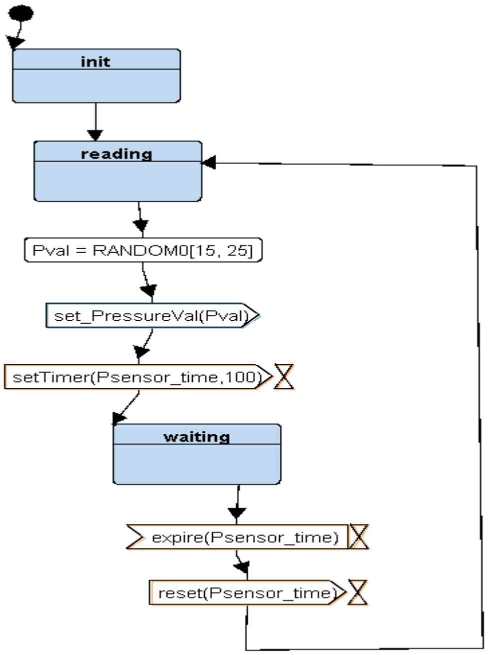
6. System Design:

- Block Diagram:

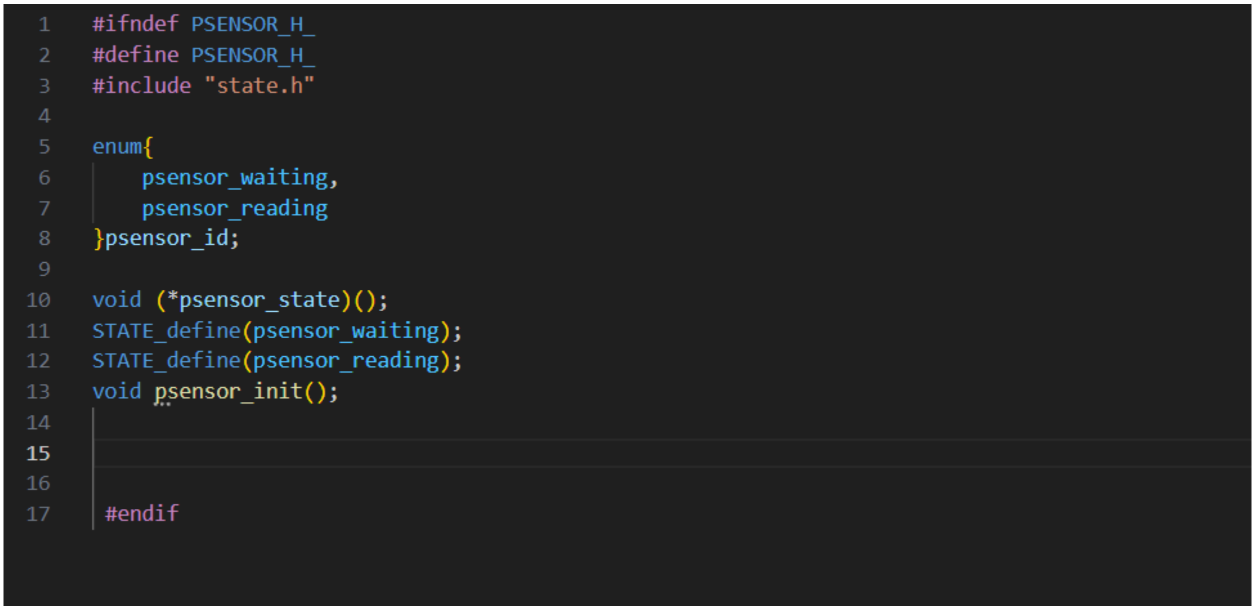


-The State Machines of every module and its codes:

- Pressure Sensor Design:

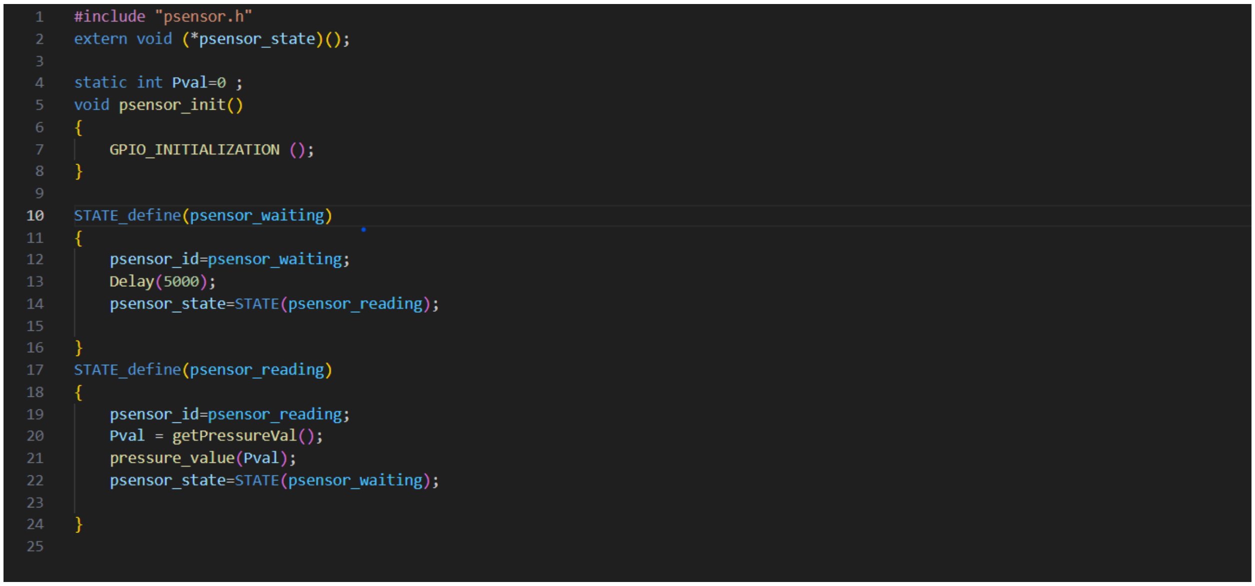


-Psensor.h:



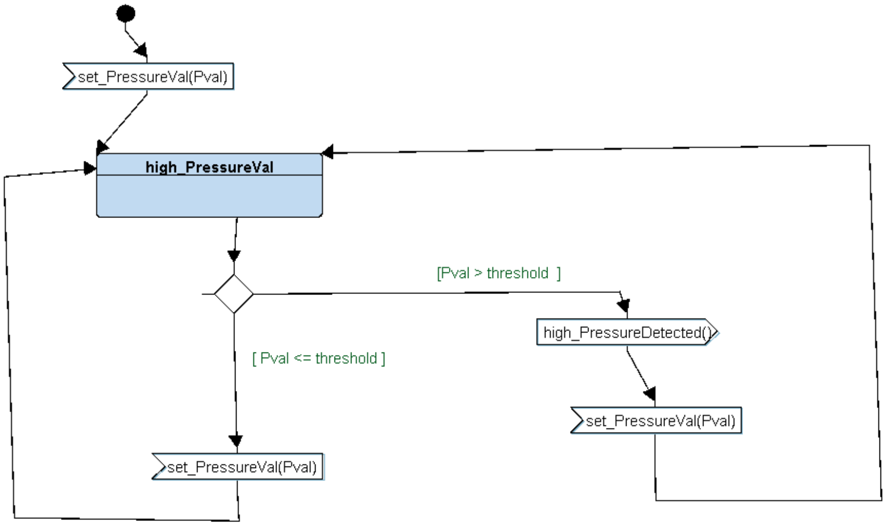
```c
#ifndef PSENSOR_H_
#define PSENSOR_H_
#include "state.h"

enum{
    psensor_waiting,
    psensor_reading
}psensor_id;

void (*psensor_state)();
STATE_define(psensor_waiting);
STATE_define(psensor_reading);
void psensor_init();



 #endif
```

-Psensor.c:

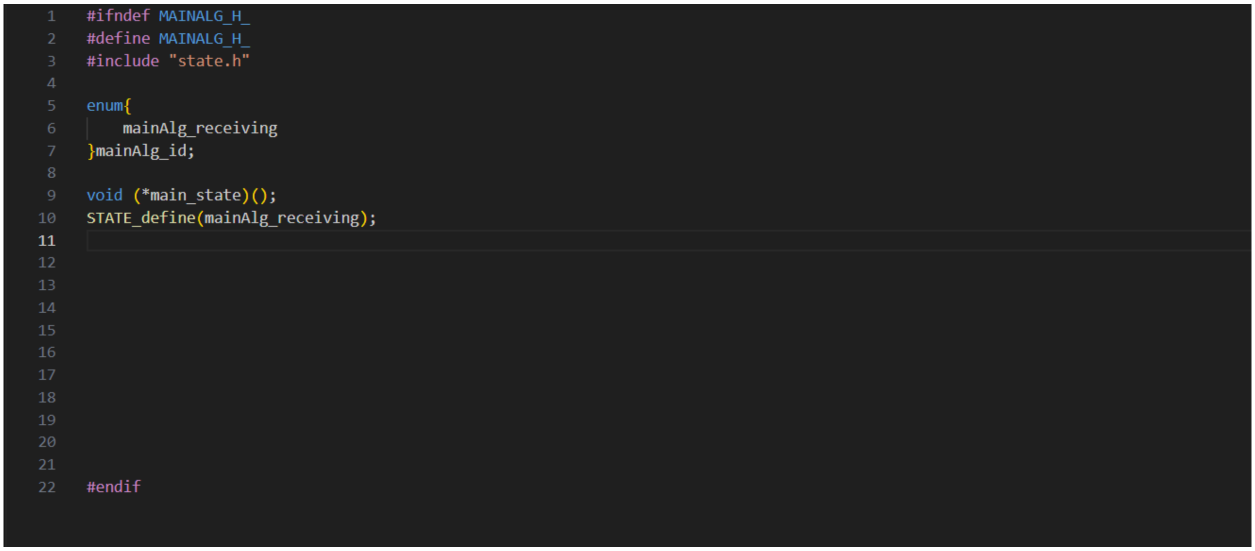


```c
#include "psensor.h"
extern void (*psensor_state)();

static int Pval=0 ;
void psensor_init()
{
    GPIO_INITIALIZATION ();
}

STATE_define(psensor_waiting)
{
    psensor_id=psensor_waiting;
    Delay(5000);
    psensor_state=STATE(psensor_reading);

}
STATE_define(psensor_reading)
{
    psensor_id=psensor_reading;
    Pval = getPressureVal();
    pressure_value(Pval);
    psensor_state=STATE(psensor_waiting);

}

```

- ## Main Algorithm Design:

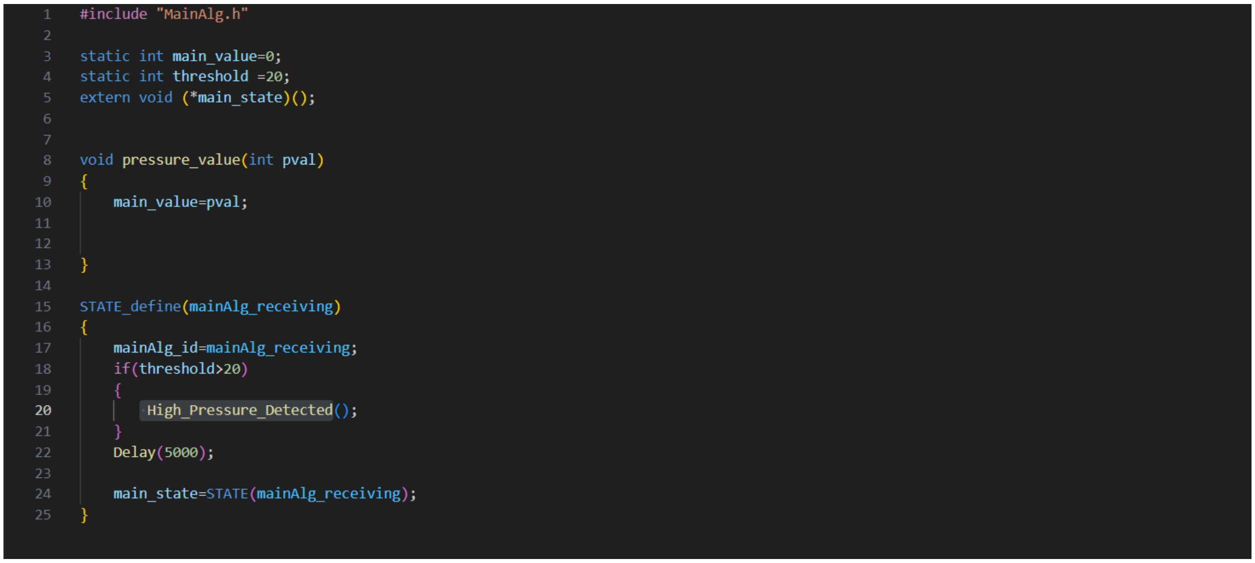


-MainAlg.h:



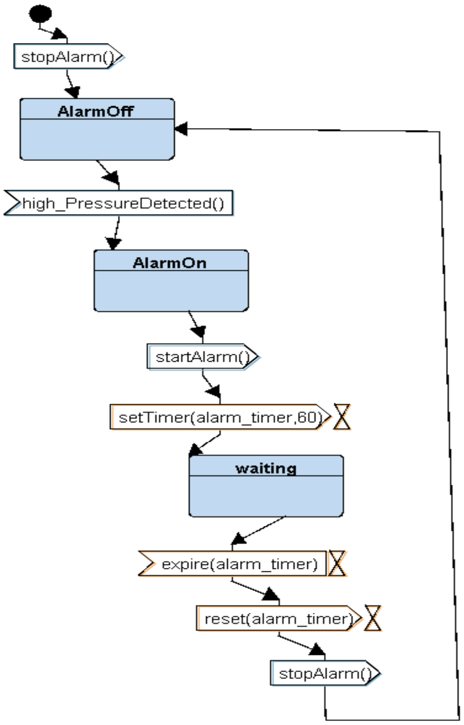
```
#ifndef MAINALG_H_
#define MAINALG_H_
#include "state.h"

enum{
    mainAlg_receiving
}mainAlg_id;

void (*main_state)();
STATE_define(mainAlg_receiving);











#endif
```

-MianAlg.c:

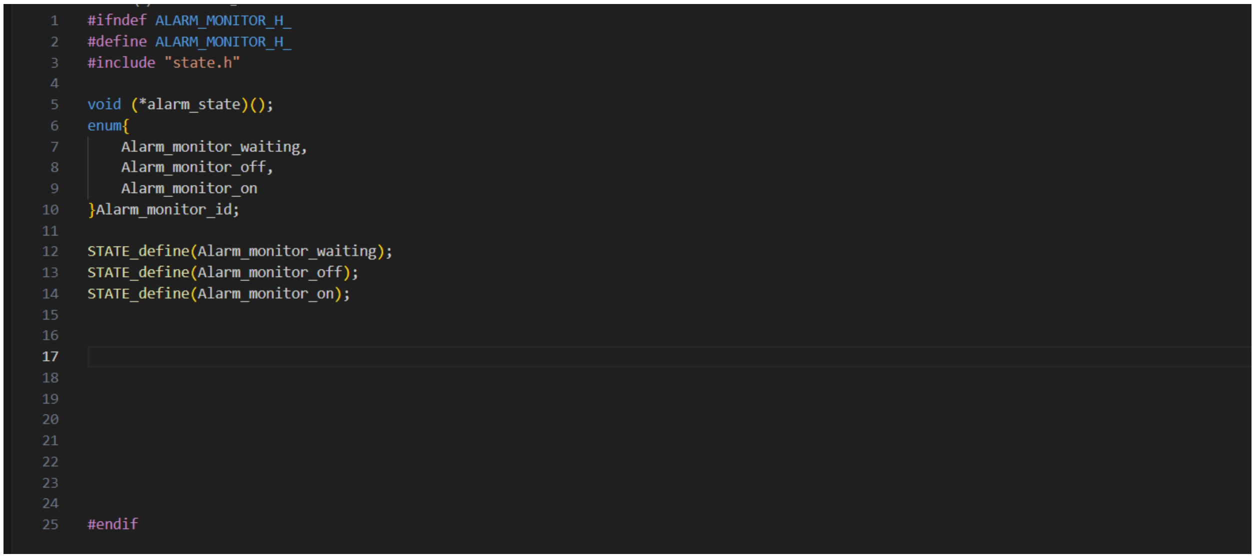


```
#include "MainAlg.h"

static int main_value=0;
static int threshold =20;
extern void (*main_state)();


void pressure_value(int pval)
{
    main_value=pval;


}

STATE_define(mainAlg_receiving)
{
    mainAlg_id=mainAlg_receiving;
    if(threshold>20)
    {
        High_Pressure_Detected();
    }
    Delay(5000);

    main_state=STATE(mainAlg_receiving);
}
```

- ## Alarm Monitor Design:

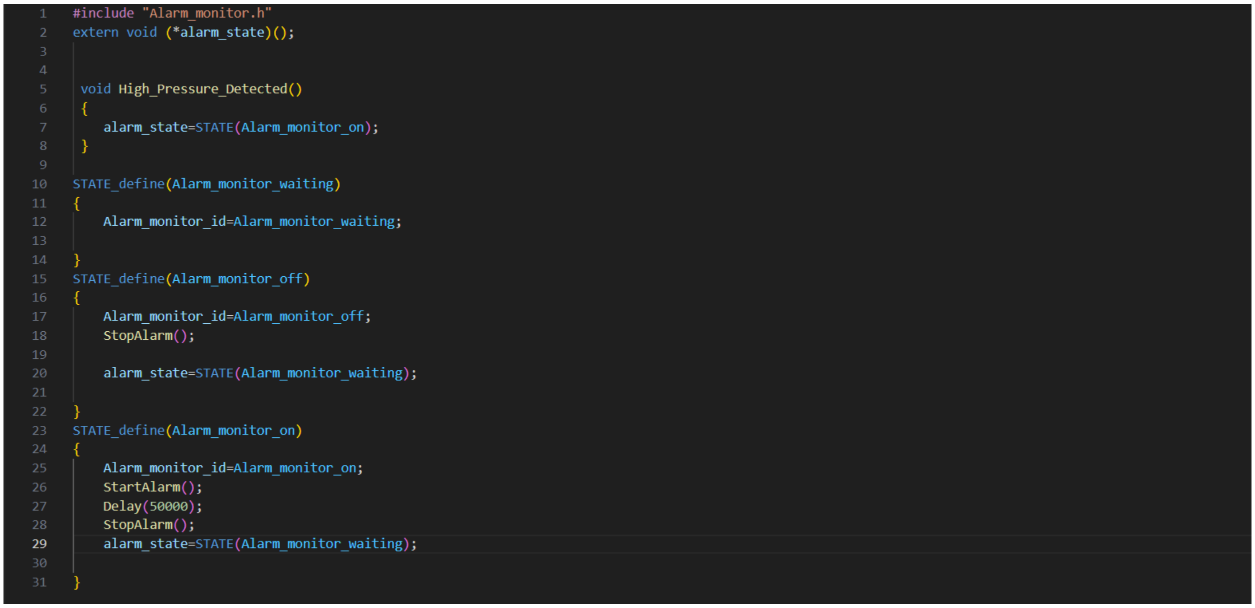


-Alarm_Monitor.h:



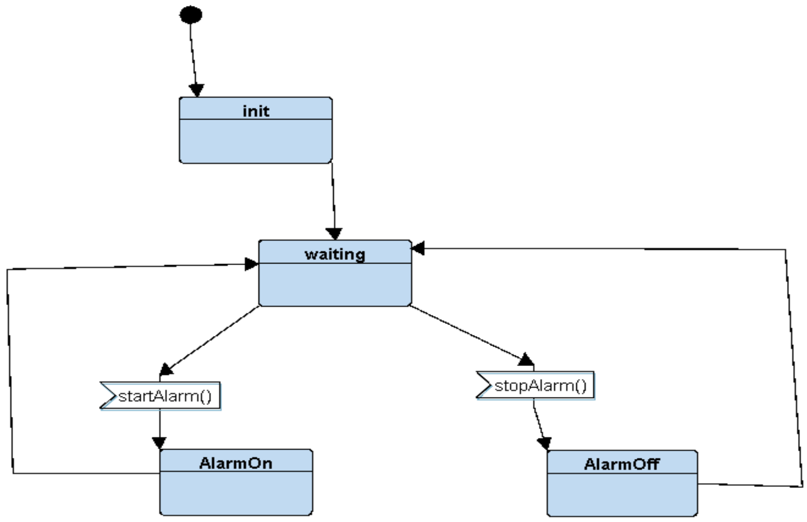
```
#ifndef ALARM_MONITOR_H_
#define ALARM_MONITOR_H_
#include "state.h"

void (*alarm_state)();
enum{
    Alarm_monitor_waiting,
    Alarm_monitor_off,
    Alarm_monitor_on
}Alarm_monitor_id;

STATE_define(Alarm_monitor_waiting);
STATE_define(Alarm_monitor_off);
STATE_define(Alarm_monitor_on);










#endif
```

-Alarm_Monitor.c:

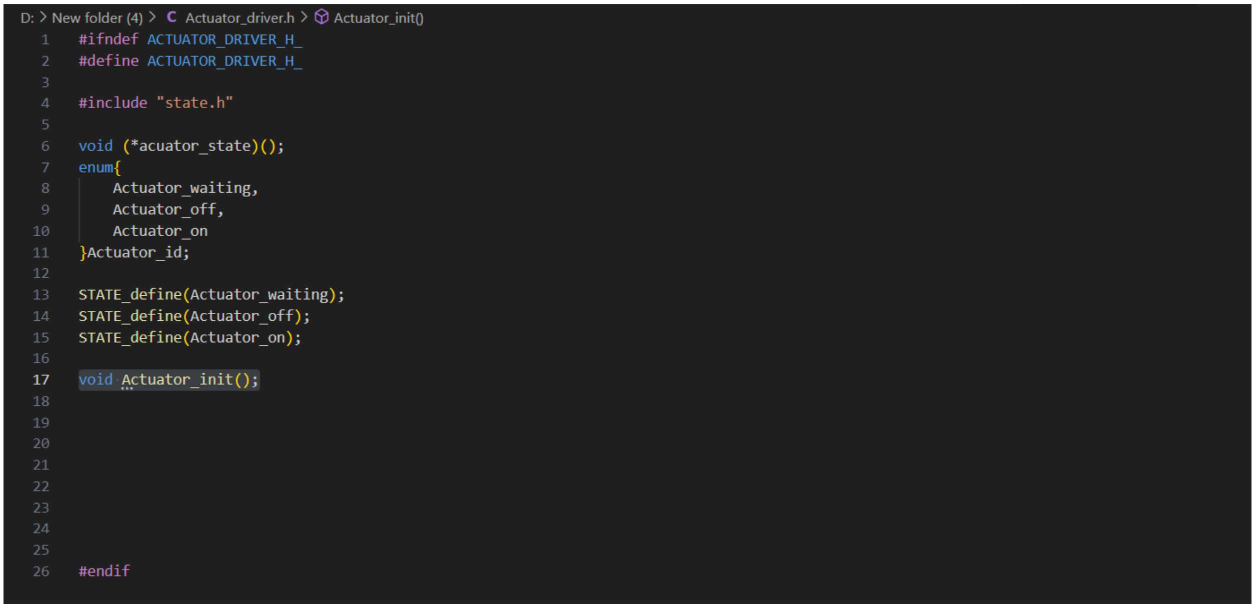


```
#include "Alarm_monitor.h"
extern void (*alarm_state)();


 void High_Pressure_Detected()
 {
    alarm_state=STATE(Alarm_monitor_on);
 }

STATE_define(Alarm_monitor_waiting)
{
    Alarm_monitor_id=Alarm_monitor_waiting;

}
STATE_define(Alarm_monitor_off)
{
    Alarm_monitor_id=Alarm_monitor_off;
    StopAlarm();

    alarm_state=STATE(Alarm_monitor_waiting);

}
STATE_define(Alarm_monitor_on)
{
    Alarm_monitor_id=Alarm_monitor_on;
    StartAlarm();
    Delay(50000);
    StopAlarm();
    alarm_state=STATE(Alarm_monitor_waiting);

}
```

- ## Actuator Driver Design:

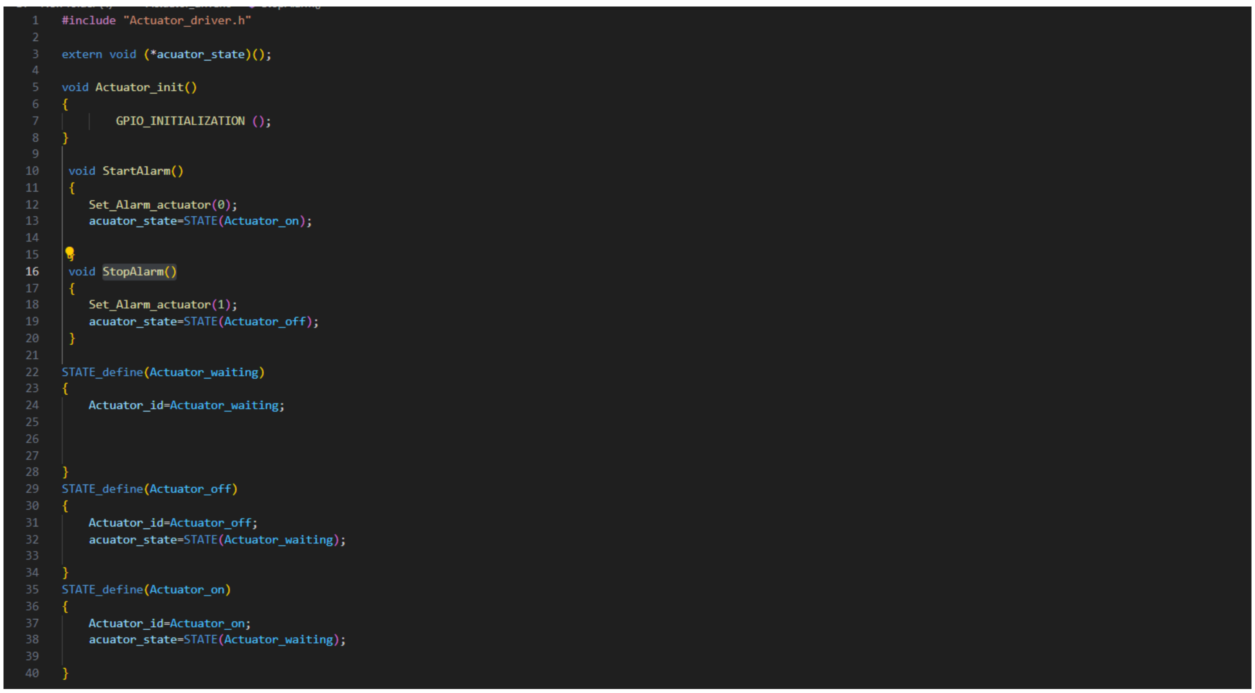


-Actuator_driver.h:



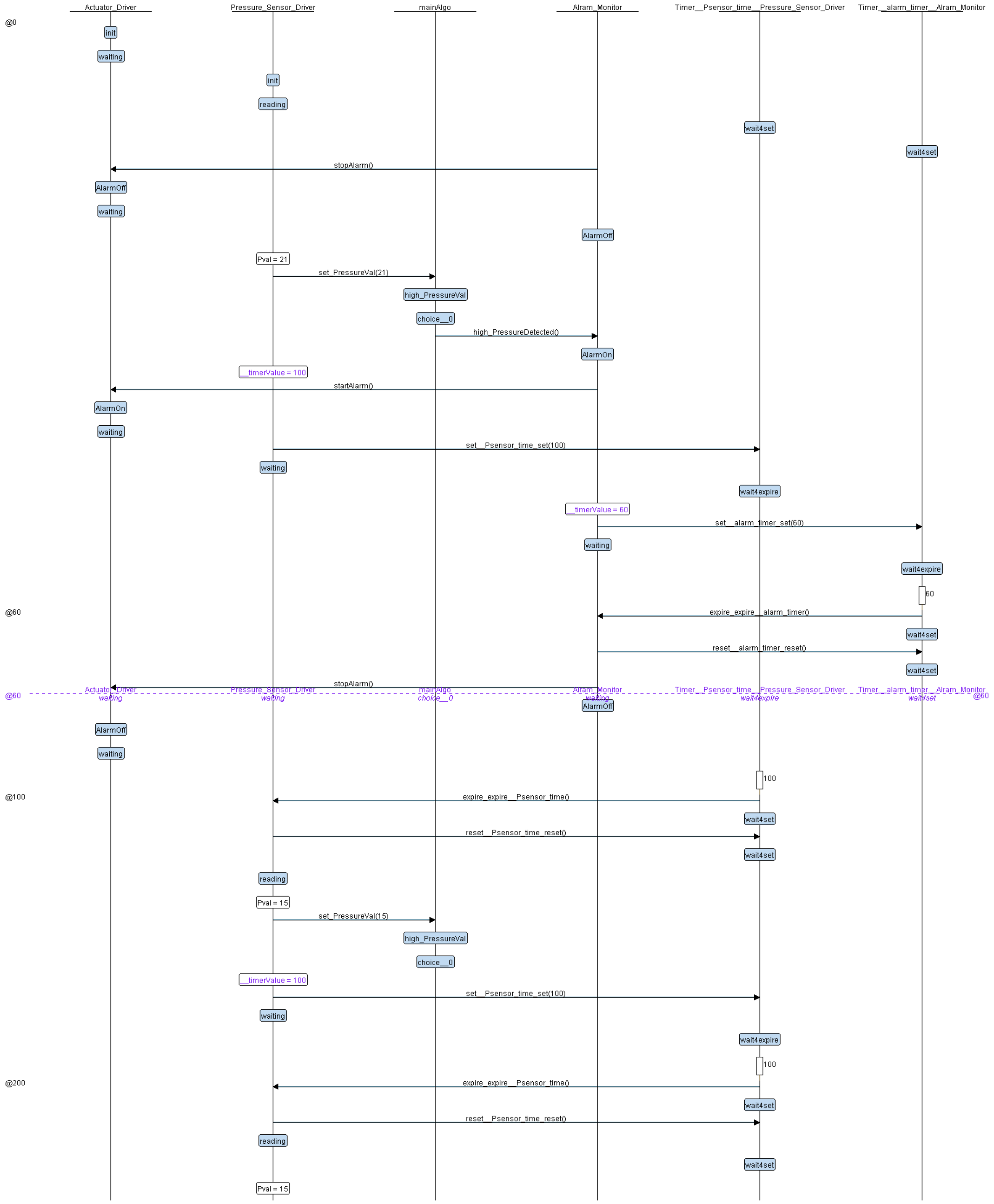
```c
#ifndef ACTUATOR_DRIVER_H_
#define ACTUATOR_DRIVER_H_

#include "state.h"

void (*acuator_state)();
enum{
    Actuator_waiting,
    Actuator_off,
    Actuator_on
}Actuator_id;

STATE_define(Actuator_waiting);
STATE_define(Actuator_off);
STATE_define(Actuator_on);

void Actuator_init();









#endif
```

-Actuator_driver.c:



```c
#include "Actuator_driver.h"

extern void (*acuator_state)();

void Actuator_init()
{
        GPIO_INITIALIZATION ();
}

 void StartAlarm()
 {
    Set_Alarm_actuator(0);
    acuator_state=STATE(Actuator_on);


 void StopAlarm()
 {
    Set_Alarm_actuator(1);
    acuator_state=STATE(Actuator_off);
 }

STATE_define(Actuator_waiting)
{
    Actuator_id=Actuator_waiting;



}
STATE_define(Actuator_off)
{
    Actuator_id=Actuator_off;
    acuator_state=STATE(Actuator_waiting);

}
STATE_define(Actuator_on)
{
    Actuator_id=Actuator_on;
    acuator_state=STATE(Actuator_waiting);

}
```

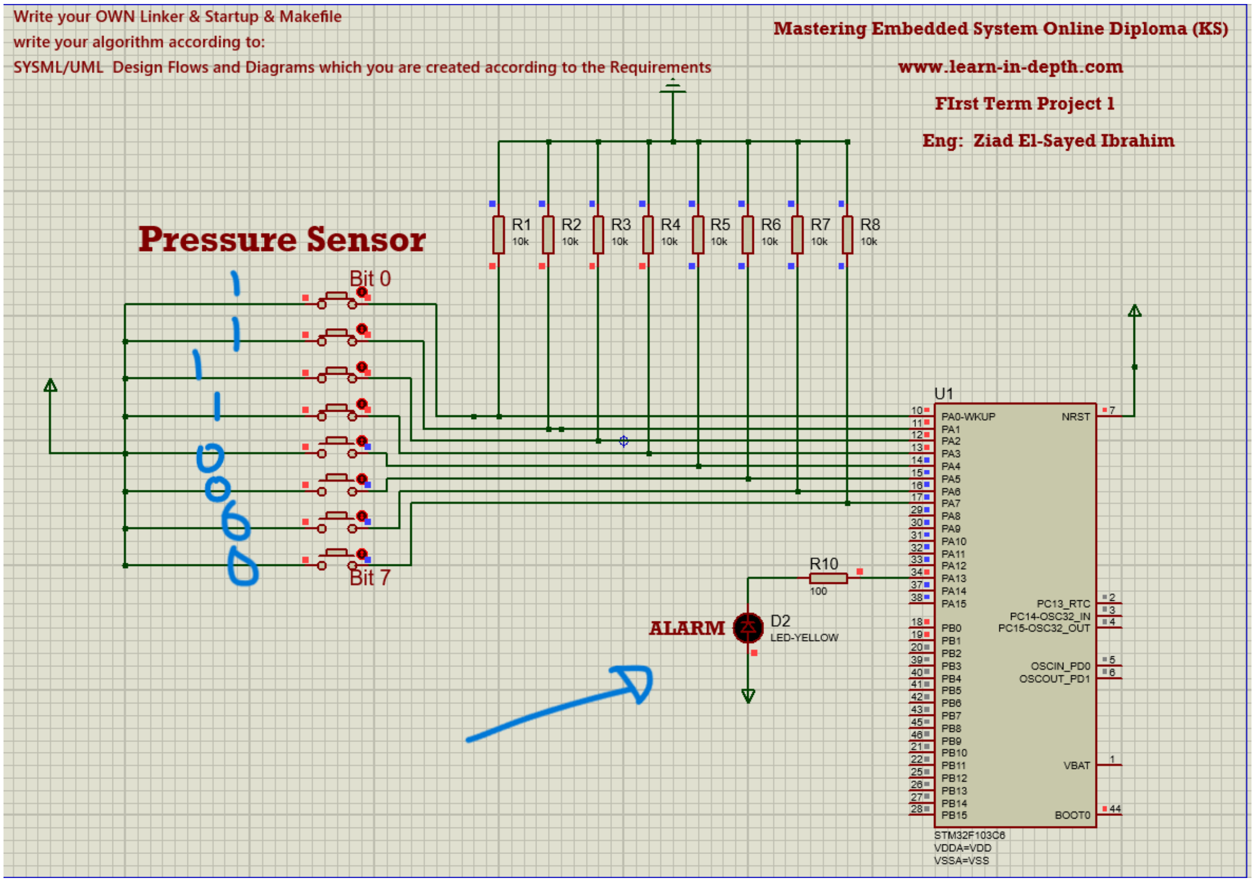
- Design Simulation:



- **Proteus simulation:**

-if we make the pins greater than 20(decimal) the led will be on and off and if we put less than that it will be always off.

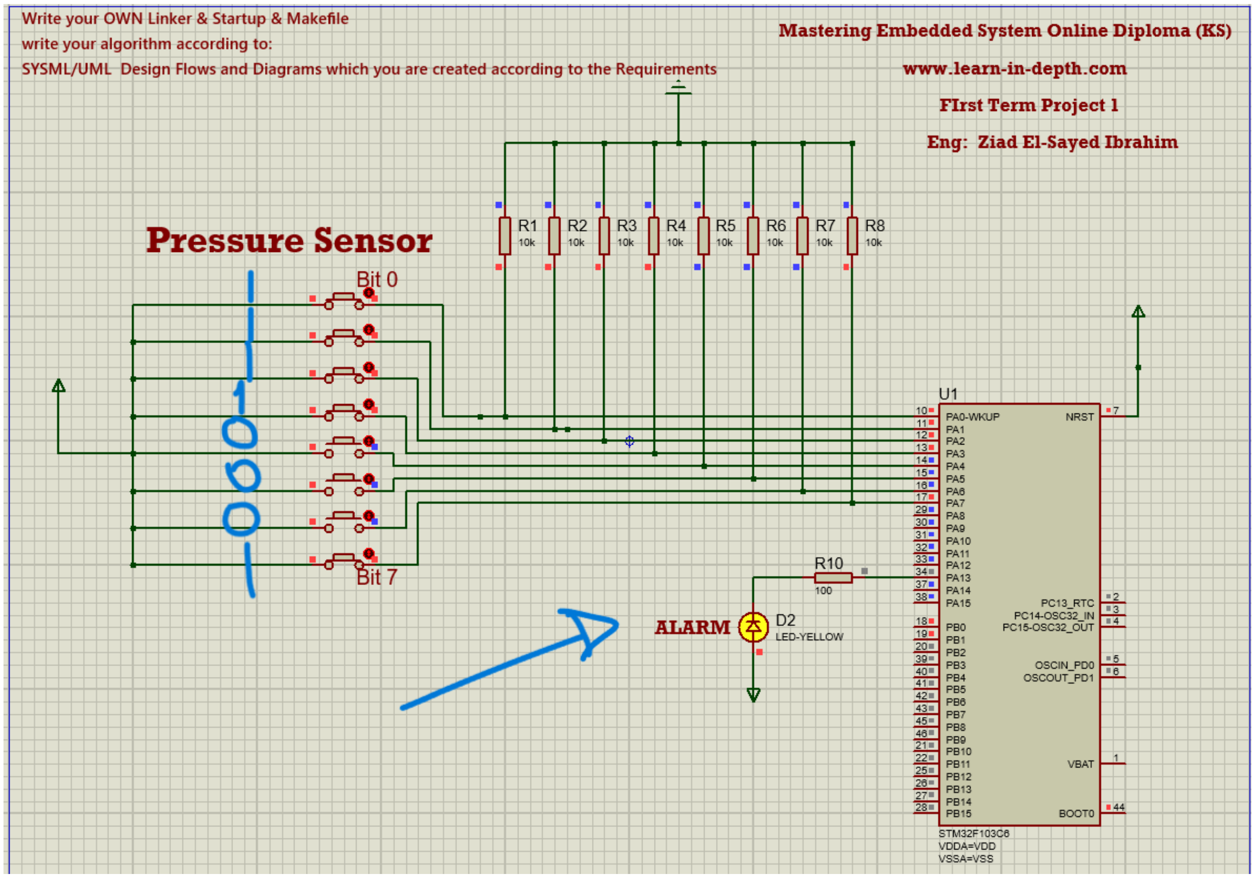
1. Smaller than 20 (00001111=15):



Write your OWN Linker & Startup & Makefile
write your algorithm according to:
SYSML/UML Design Flows and Diagrams which you are created according to the Requirements

Mastering Embedded System Online Diploma (KS)
www.learn-in-depth.com
FIrst Term Project 1
Eng: Ziad El-Sayed Ibrahim

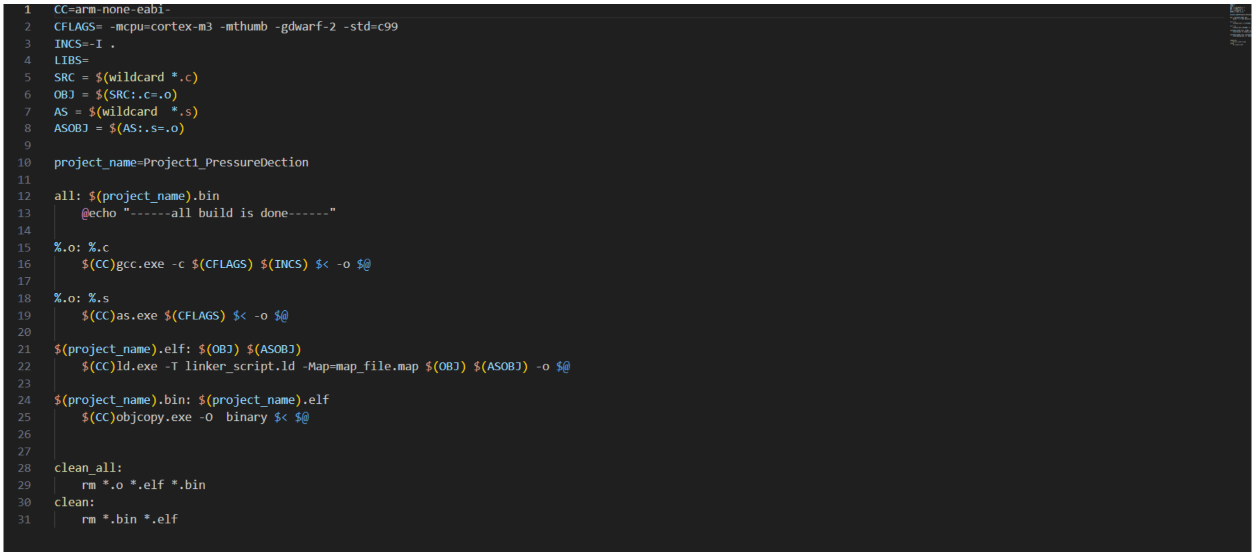
1. Greater than 20 (1000111):



Write your OWN Linker & Startup & Makefile
write your algorithm according to:
SYSML/UML Design Flows and Diagrams which you are created according to the Requirements

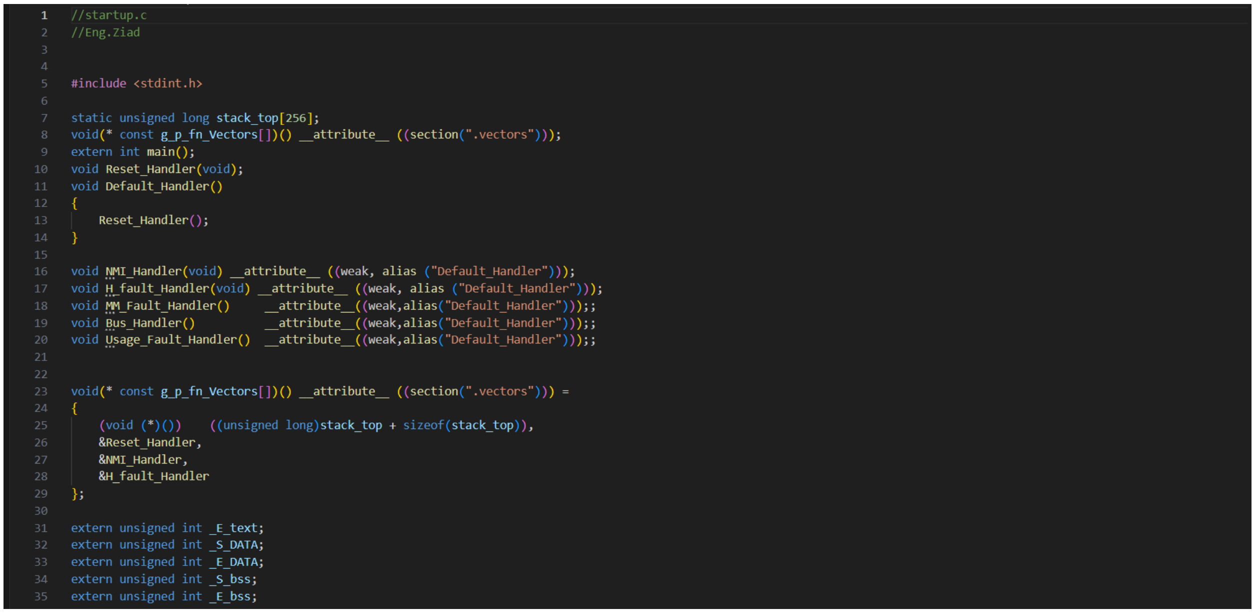
Mastering Embedded System Online Diploma (KS)
www.learn-in-depth.com
FIrst Term Project 1
Eng: Ziad El-Sayed Ibrahim

-We compile with this **Makefile:**

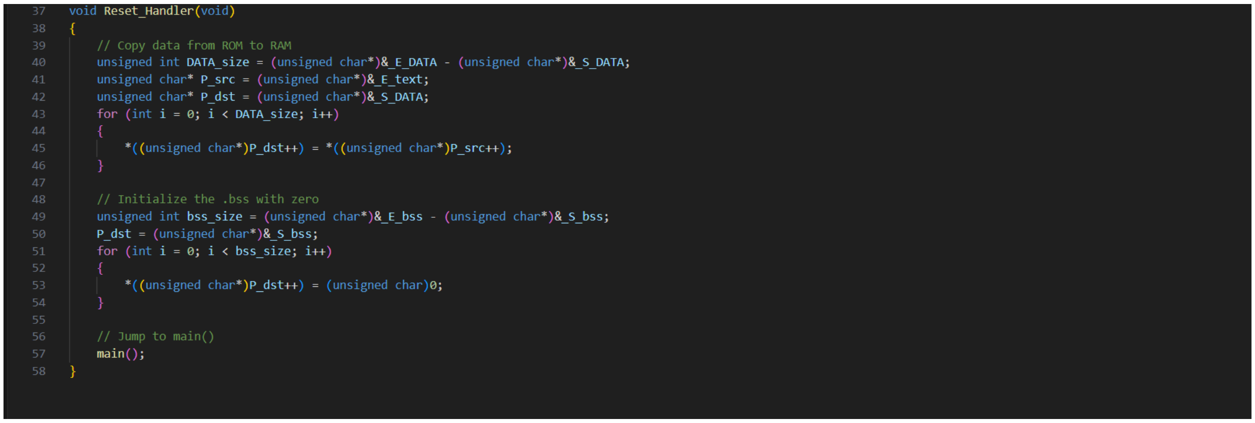


```
CC=arm-none-eabi-
CFLAGS= -mcpu=cortex-m3 -mthumb -gdwarf-2 -std=c99
INCS=-I .
LIBS=
SRC = $(wildcard *.c)
OBJ = $(SRC:.c=.o)
AS = $(wildcard  *.s)
ASOBJ = $(AS:.s=.o)

project_name=Project1_PressureDection

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 $< $@


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

-the Startup.c:

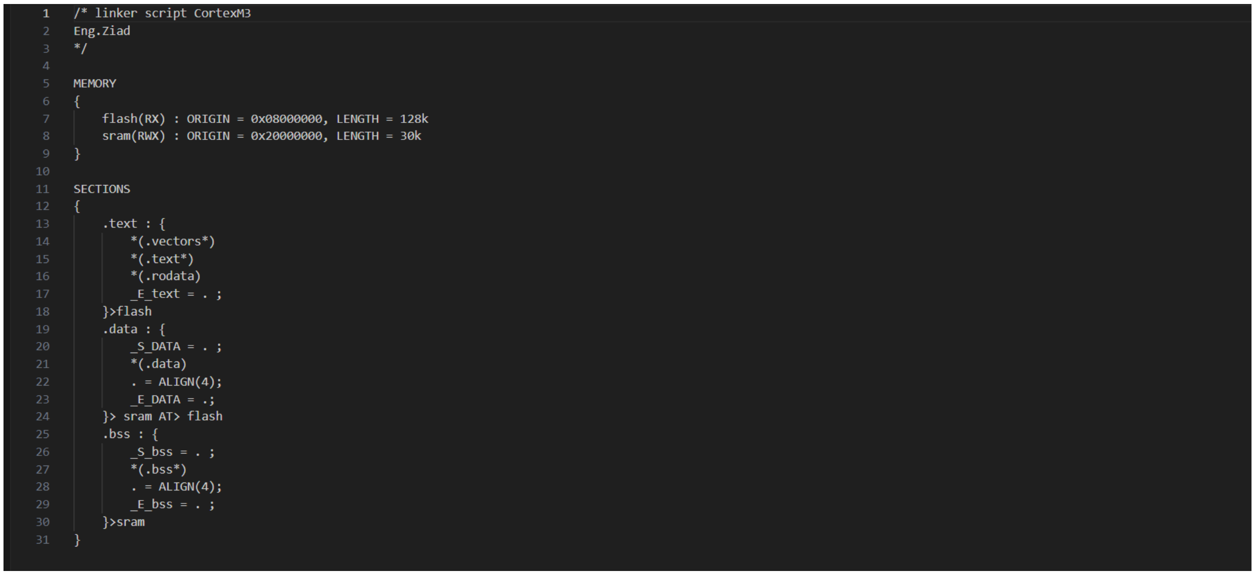


```c
//startup.c
//Eng.Ziad


#include <stdint.h>

static unsigned long stack_top[256];
void(* const g_p_fn_Vectors[])() __attribute__ ((section(".vectors")));
extern int main();
void Reset_Handler(void);
void Default_Handler()
{
    Reset_Handler();
}

void NMI_Handler(void) __attribute__ ((weak, alias ("Default_Handler")));
void H_fault_Handler(void) __attribute__ ((weak, alias ("Default_Handler")));
void MM_Fault_Handler()     __attribute__((weak,alias("Default_Handler")));;
void Bus_Handler()          __attribute__((weak,alias("Default_Handler")));;
void Usage_Fault_Handler()  __attribute__((weak,alias("Default_Handler")));;


void(* const g_p_fn_Vectors[])() __attribute__ ((section(".vectors"))) =
{
    (void (*)())    ((unsigned long)stack_top + sizeof(stack_top)),
    &Reset_Handler,
    &NMI_Handler,
    &H_fault_Handler
};

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;
```



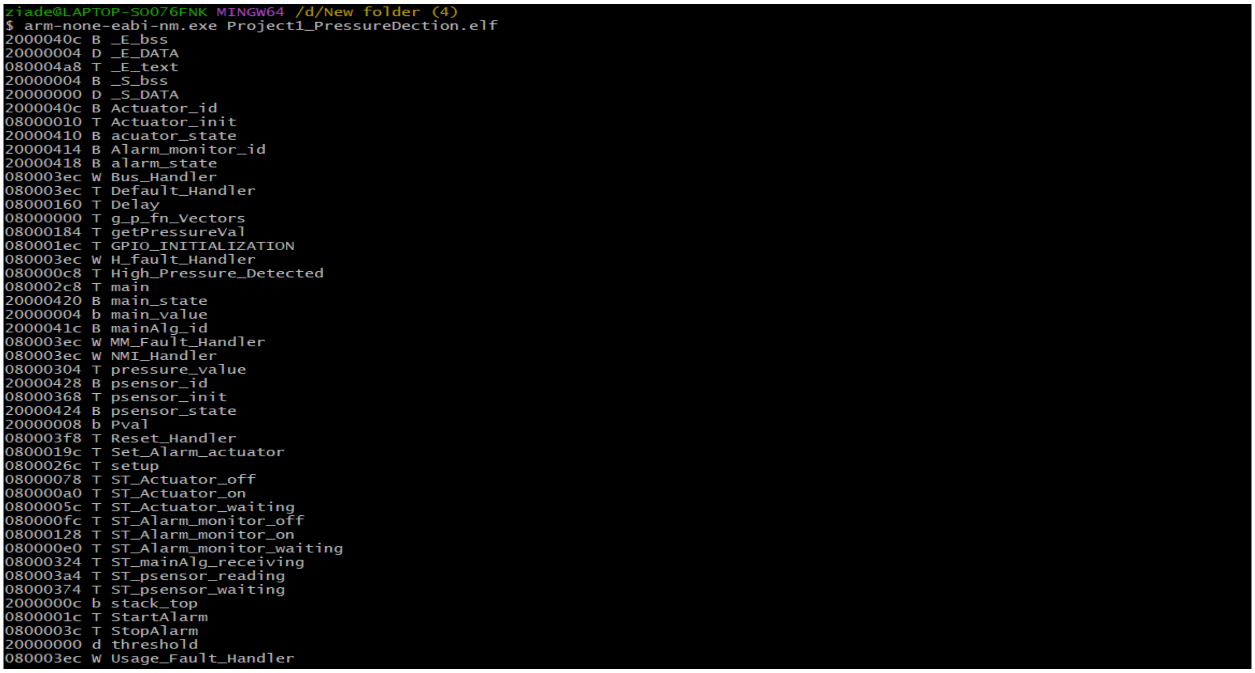
```c
void Reset_Handler(void)
{
    // Copy data from ROM to RAM
    unsigned int DATA_size = (unsigned char*)&_E_DATA - (unsigned char*)&_S_DATA;
    unsigned char* P_src = (unsigned char*)&_E_text;
    unsigned char* P_dst = (unsigned char*)&_S_DATA;
    for (int i = 0; i < DATA_size; i++)
    {
        *((unsigned char*)P_dst++) = *((unsigned char*)P_src++);
    }

    // Initialize the .bss with zero
    unsigned int bss_size = (unsigned char*)&_E_bss - (unsigned char*)&_S_bss;
    P_dst = (unsigned char*)&_S_bss;
    for (int i = 0; i < bss_size; i++)
    {
        *((unsigned char*)P_dst++) = (unsigned char)0;
    }

    // Jump to main()
    main();
}
```

-linker_script.ld:



```
/* linker script CortexM3
Eng.Ziad
*/

MEMORY
{
    flash(RX) : ORIGIN = 0x08000000, LENGTH = 128k
    sram(RWX) : ORIGIN = 0x20000000, LENGTH = 30k
}

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 = . ;
    }>sram
}
```

-Symboltable:



```
ziade@LAPTOP-SO076FNK MINGW64 /d/New folder (4)
$ arm-none-eabi-nm.exe Project1_PressureDection.elf
2000040c B _E_bss
20000004 D _E_DATA
080004a8 T _E_text
20000004 B _S_bss
20000000 D _S_DATA
2000040c B Actuator_id
08000010 T Actuator_init
20000410 B acuator_state
20000414 B Alarm_monitor_id
20000418 B alarm_state
080003ec W Bus_Handler
080003ec T Default_Handler
08000160 T Delay
08000000 T g_p_fn_Vectors
08000184 T getPressureVal
080001ec T GPIO_INITIALIZATION
080003ec W H_fault_Handler
080000c8 T High_Pressure_Detected
080002c8 T main
20000420 B main_state
20000004 b main_value
2000041c B mainAlg_id
080003ec W MM_Fault_Handler
080003ec W NMI_Handler
08000304 T pressure_value
20000428 B psensor_id
08000368 T psensor_init
20000424 B psensor_state
20000008 b Pval
080003f8 T Reset_Handler
0800019c T Set_Alarm_actuator
0800026c T setup
08000078 T ST_Actuator_off
080000a0 T ST_Actuator_on
0800005c T ST_Actuator_waiting
080000fc T ST_Alarm_monitor_off
08000128 T ST_Alarm_monitor_on
080000e0 T ST_Alarm_monitor_waiting
08000324 T ST_mainAlg_receiving
080003a4 T ST_psensor_reading
08000374 T ST_psensor_waiting
2000000c b stack_top
0800001c T StartAlarm
0800003c T StopAlarm
20000000 d threshold
080003ec W Usage_Fault_Handler
```

-That’s all we want you to know about our program thanks for reading this, may we will meet again inshallah!