Mastering Embedded System Online Diploma

➢ **www.learn-in-depth.com**

Project Number	**First Term (Final Project 1)**
Project Name	**Pressure Controller Project**
Name	**Osama Youssef Tawadrous**
My Profile	**https://www.learn-in-depth.com/online-diploma/osamayoussef996%40gmail.com**

➢Table Of Content :-

➢ Case study :

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.
- Keep track of the measured values.

➢ Assumptions :

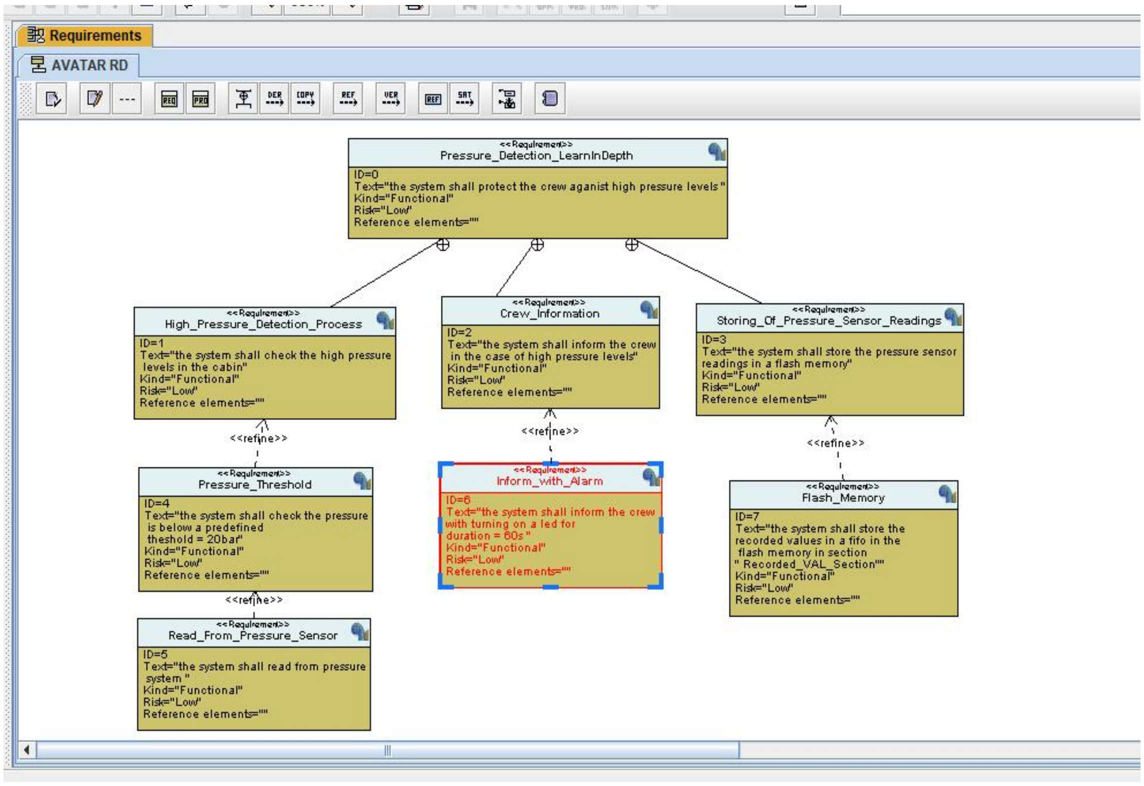
- The system setup and shutdown procedures are not modeled.
- The system maintenance is not modeled.
- The pressure sensor never fails.
- The alarm never fails.
- The system never faces power cut.

➢Requirement Diagram :

We will divide our case study to three main requirements

- ●High Pressure Detection Process .
- ●Inform The Crew at High Pressure Levels .
- ●Optional Storage of Pressure Readings .

Each main requirement has refinement requirements as mentioned below .



<<Requirement>>
Pressure_Detection_LearnInDepth
ID=0
Text="the system shall protect the crew aganist high pressure levels"
Kind="Functional"
Risk="Low"
Reference elements=""

<<Requirement>>
High_Pressure_Detection_Process
ID=1
Text="the system shall check the high pressure levels in the cabin"
Kind="Functional"
Risk="Low"
Reference elements=""

<<Requirement>>
Crew_Information
ID=2
Text="the system shall inform the crew in the case of high pressure levels"
Kind="Functional"
Risk="Low"
Reference elements=""

<<Requirement>>
Storing_Of_Pressure_Sensor_Readings
ID=3
Text="the system shall store the pressure sensor readings in a flash memory"
Kind="Functional"
Risk="Low"
Reference elements=""

<<Requirement>>
Pressure_Threshold
ID=4
Text="the system shall check the pressure is below a predefined theshold = 20bar"
Kind="Functional"
Risk="Low"
Reference elements=""

<<Requirement>>
Inform_with_Alarm
ID=6
Text="the system shall inform the crew with turning on a led for duration = 60s "
Kind="Functional"
Risk="Low"
Reference elements=""

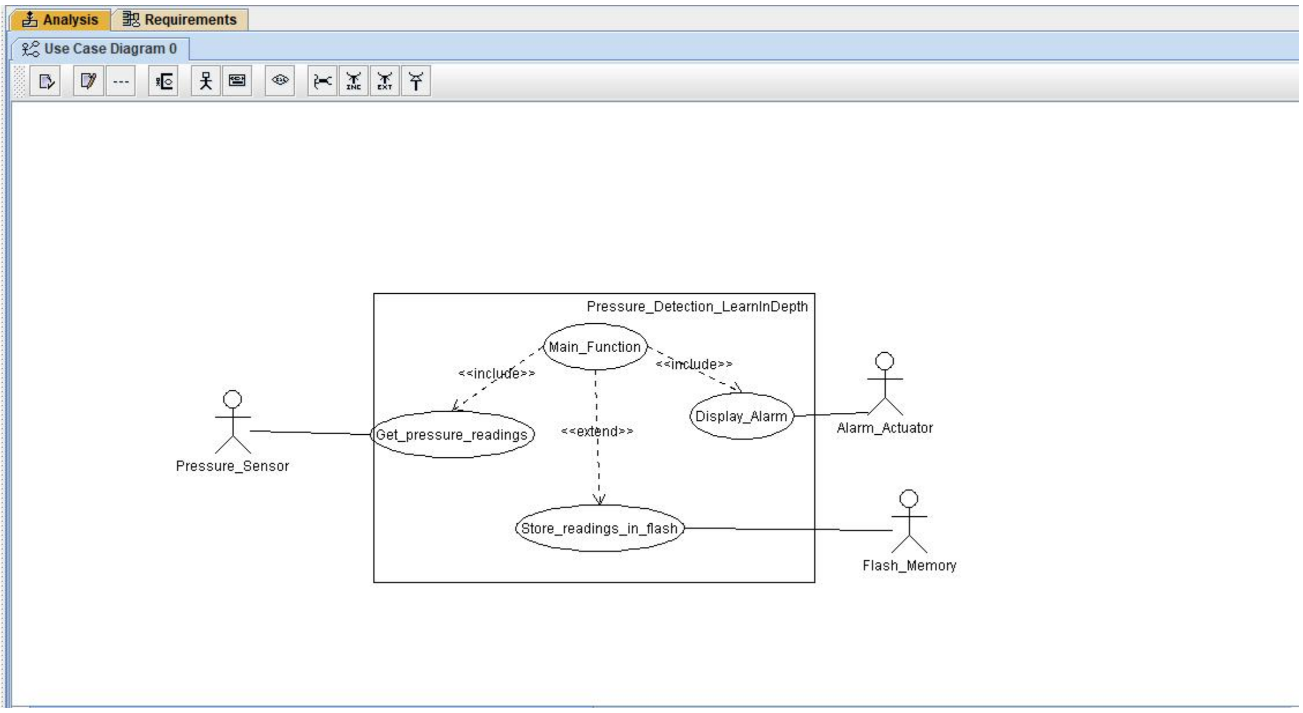
<<Requirement>>
Flash_Memory
ID=7
Text="the system shall store the recorded values in a fifo in the flash memory in section " Recorded_VAL_Section""
Kind="Functional"
Risk="Low"
Reference elements=""

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

➢ System Analysis Diagram :

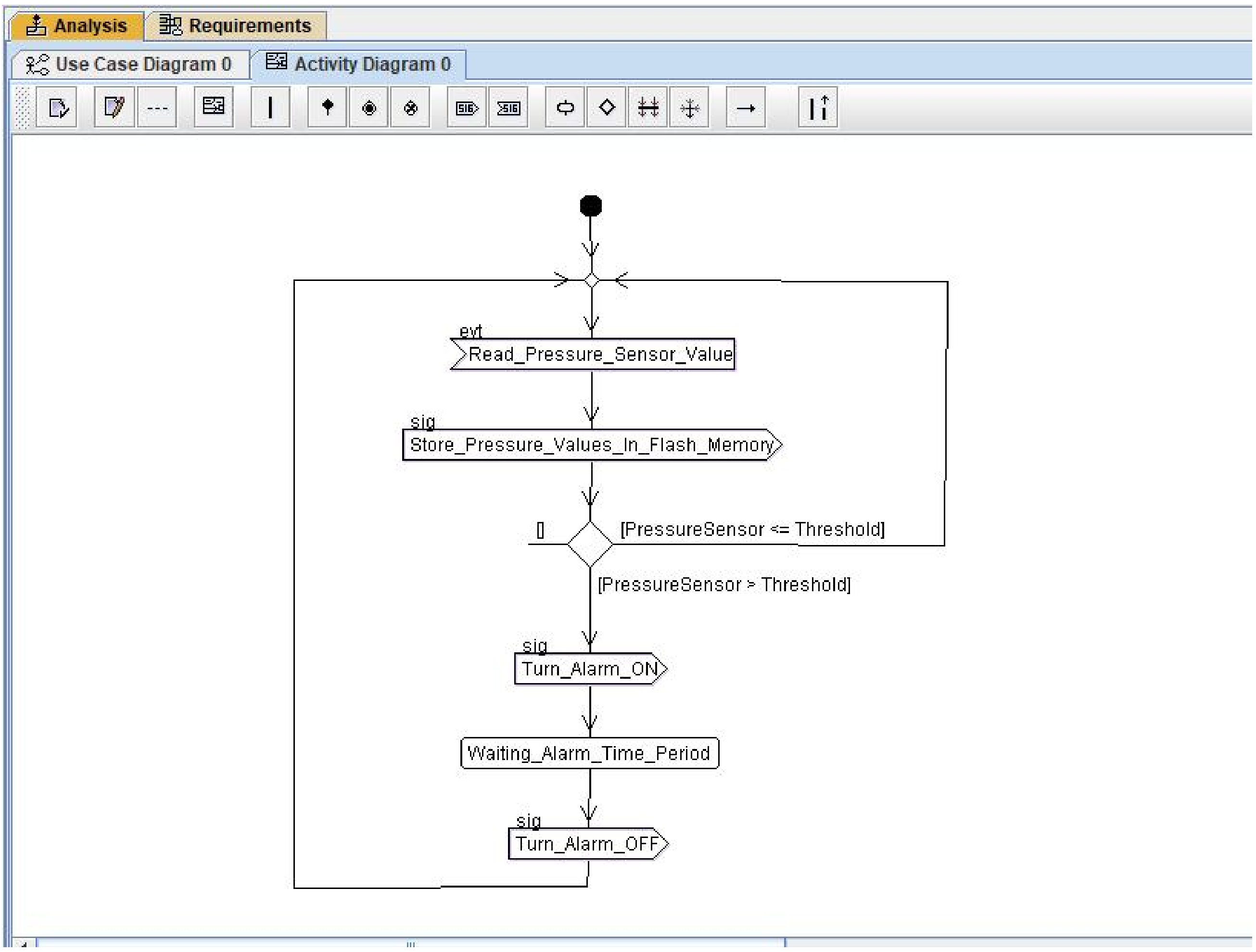
➢ Use Case Diagram :

In use case diagram , we discuss system boundary and main functions , so in our use case digram we will observe that our system boundary include the main function algorithm , Get pressure reading function and display alarm function and exclude the pressure sensor alarm actuator and flash memory for storing readings .



➢ Activity Diagram :

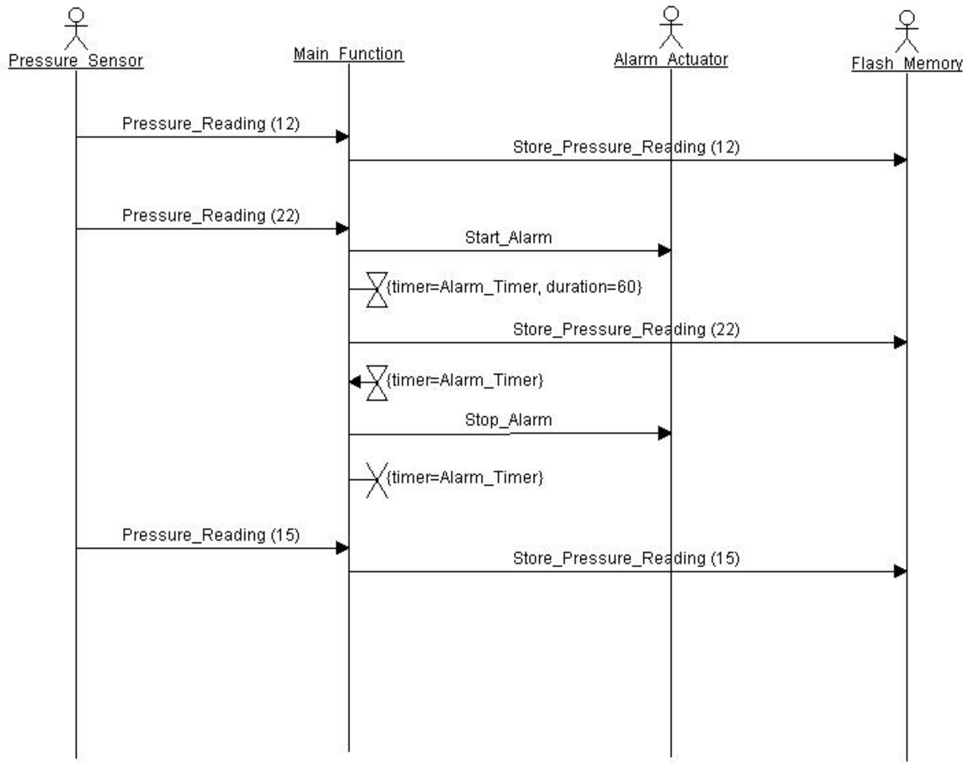
In Activity Diagram , we discuss relation between main functions , as shown below , firstly we will read the pressure value and then store it in flash memory , after that we compare the value of pressure with threshold in case study , if it exceeded this value we will turn on the alarm for 60 seconds to inform the crew that the pressure
exceeded the threshold .



Analysis
Requirements
Use Case Diagram 0
Activity Diagram 0
evt
Read_Pressure_Sensor_Value
sig
Store_Pressure_Values_In_Flash_Memory
[]
[PressureSensor <= Threshold]
[PressureSensor > Threshold]
sig
Turn_Alarm_ON
Waiting_Alarm_Time_Period
sig
Turn_Alarm_OFF

➢ **Sequence Diagram :**

In use Sequence Diagram, we discuss Communication between main system entities and actors ,as we mentioned above in activity diagram , we will see interaction between Pressure sensor actor , main function algorithm , alarm actuator and flash memory



➢**System Design:**

We will divide our case study to FOUR main modules

- **Pressure Sensor Driver Module**.

It is has one function , senses the pressure of the cabin and send it to the controller.

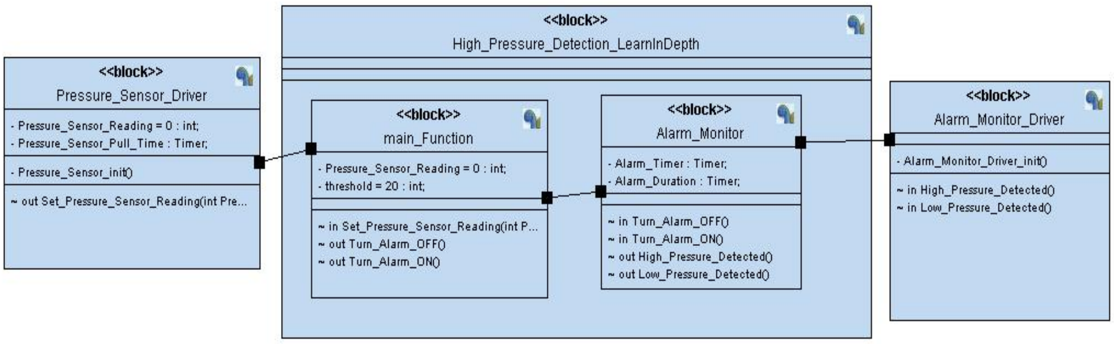
- **Main_Function Module.**

It has three function , the first function is receiving the reading from pressure sensor module , and the second function is checking the value of pressure and take the decision if alarm will turn on or not , the third function is to send the reading to the alarm to display it .

- **Alarm Monitor Module .**

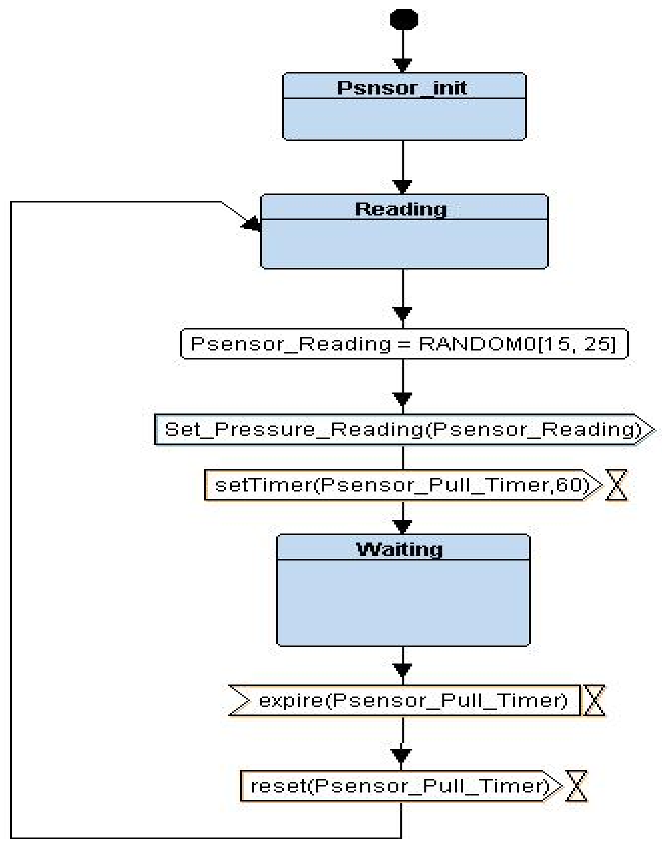
It has two function , the first function is receiving the reading from controller , the second function to send the action will be token to the alarm monitor driver.

- **Alarm Monitor Driver Module.** It has one function , take the action if turning on the alarm or not

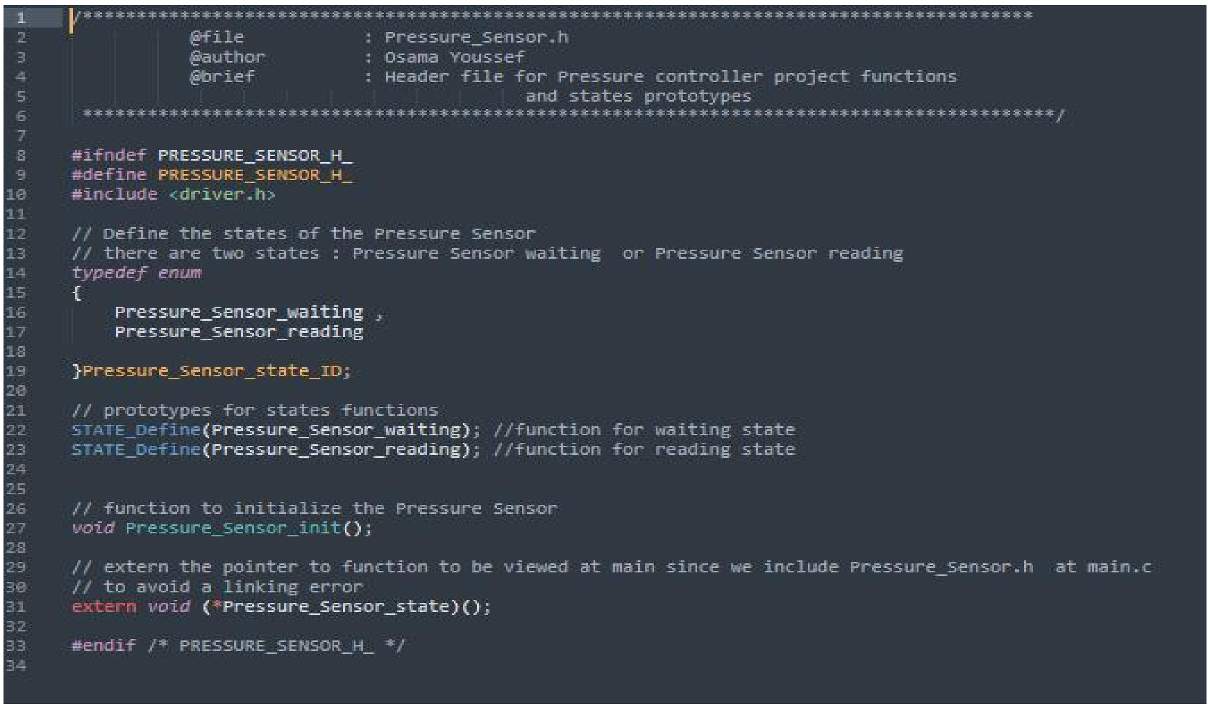

<<block>>
High_Pressure_Detection_LearnInDepth
<<block>>
Pressure_Sensor_Driver
- Pressure_Sensor_Reading = 0 : int;
- Pressure_Sensor_Pull_Time : Timer;
- Pressure_Sensor_init()
~ out Set_Pressure_Sensor_Reading(int Pre...
<<block>>
main_Function
- Pressure_Sensor_Reading = 0 : int;
- threshold = 20 : int;
~ in Set_Pressure_Sensor_Reading(int P...
~ out Turn_Alarm_OFF()
~ out Turn_Alarm_ON()
<<block>>
Alarm_Monitor
- Alarm_Timer : Timer;
- Alarm_Duration : Timer;
~ in Turn_Alarm_OFF()
~ in Turn_Alarm_ON()
~ out High_Pressure_Detected()
~ out Low_Pressure_Detected()
<<block>>
Alarm_Monitor_Driver
- Alarm_Monitor_Driver_init()
~ in High_Pressure_Detected()
~ in Low_Pressure_Detected()

➢ Source files with Block Diagrams :

- **Pressure Sensor Diagram :**

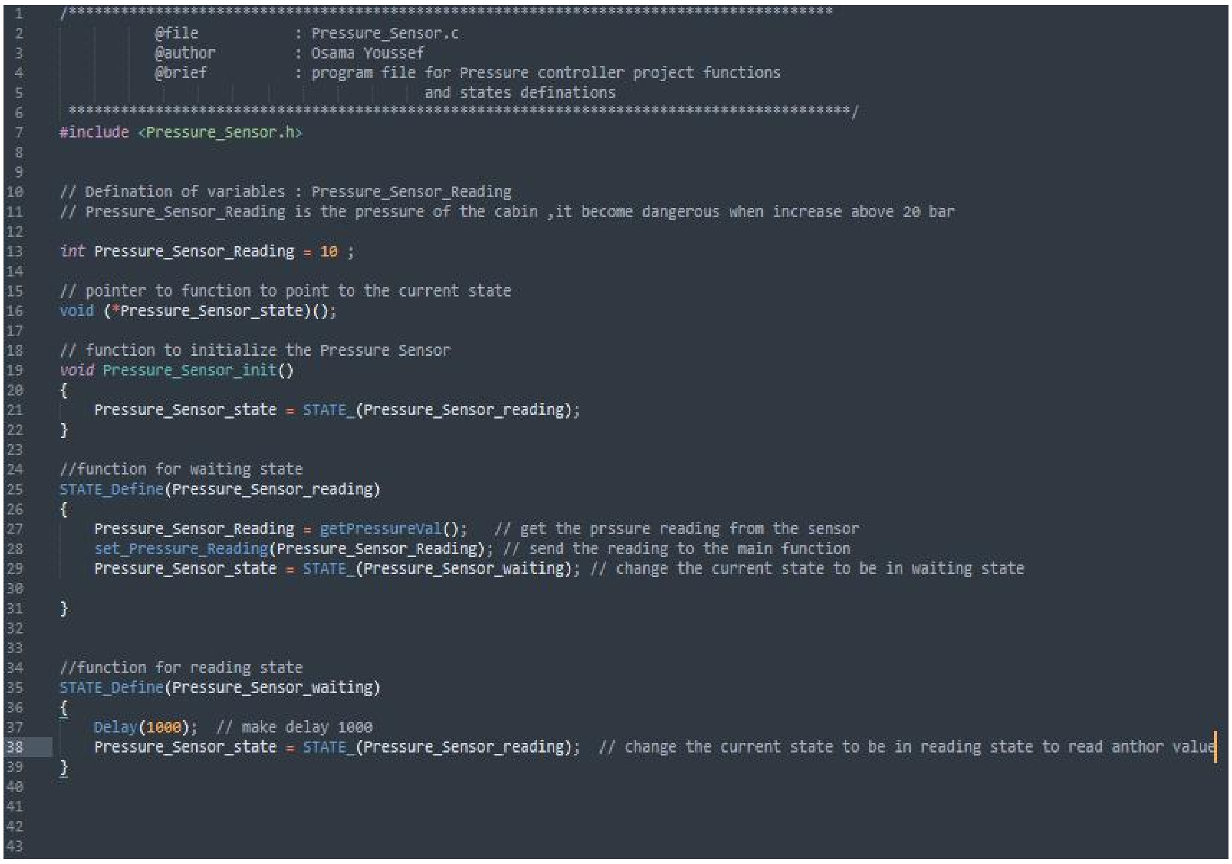

Psnsor_init
Reading
Psensor_Reading = RANDOM0[15, 25]
Set_Pressure_Reading(Psensor_Reading)
setTimer(Psensor_Pull_Timer,60)
Waiting
expire(Psensor_Pull_Timer)
reset(Psensor_Pull_Timer)

- **Pressure Sensor Header File :**



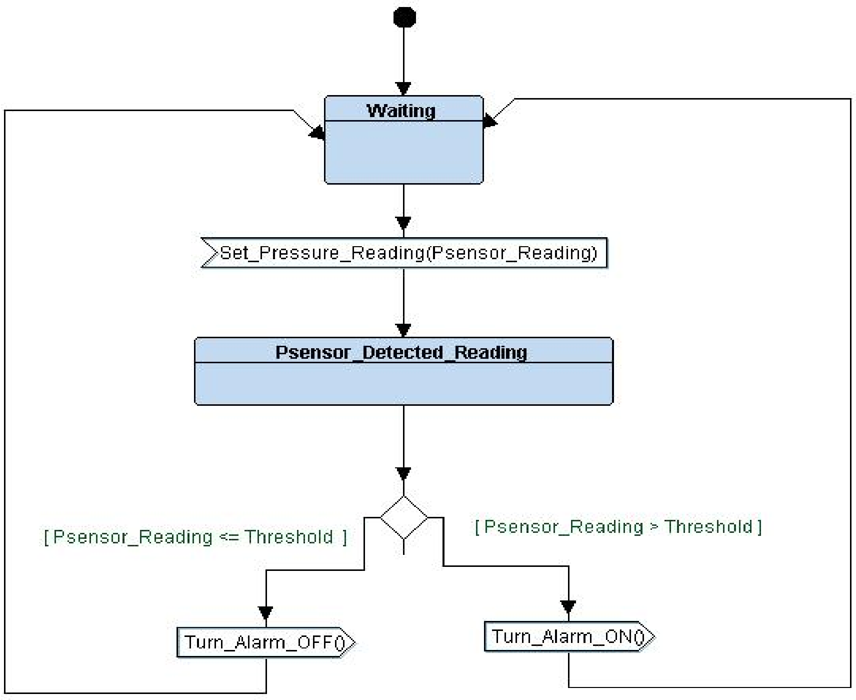
```c
/******************************************************************************************
            @file           : Pressure_Sensor.h
            @author         : Osama Youssef
            @brief          : Header file for Pressure controller project functions
                                          and states prototypes
 ******************************************************************************************/

#ifndef PRESSURE_SENSOR_H_
#define PRESSURE_SENSOR_H_
#include <driver.h>

// Define the states of the Pressure Sensor
// there are two states : Pressure Sensor waiting  or Pressure Sensor reading
typedef enum
{
    Pressure_Sensor_waiting ,
    Pressure_Sensor_reading

}Pressure_Sensor_state_ID;

// prototypes for states functions
STATE_Define(Pressure_Sensor_waiting); //function for waiting state
STATE_Define(Pressure_Sensor_reading); //function for reading state


// function to initialize the Pressure Sensor
void Pressure_Sensor_init();

// extern the pointer to function to be viewed at main since we include Pressure_Sensor.h  at main.c
// to avoid a linking error
extern void (*Pressure_Sensor_state)();

#endif /* PRESSURE_SENSOR_H_ */

```

● Pressure Sensor Program File :

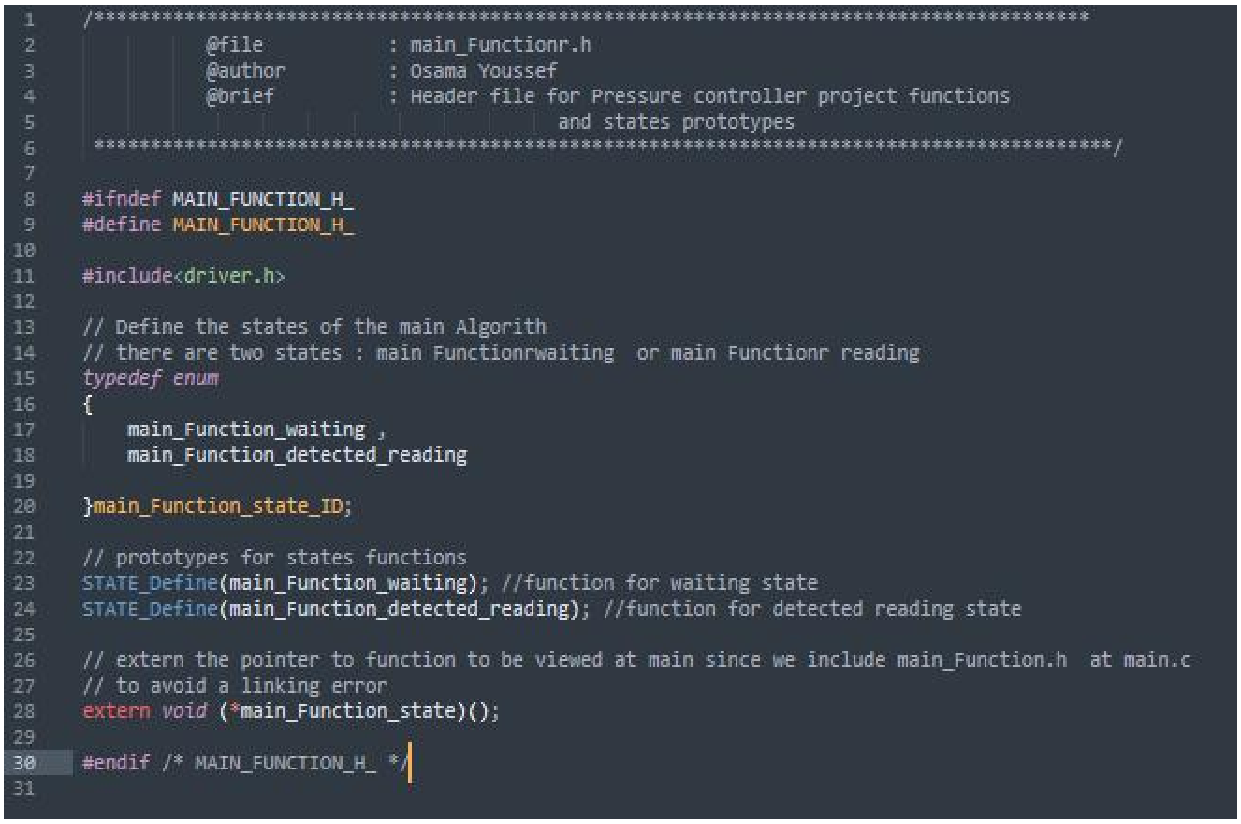


```c
/*******************************************************************************************
            @file           : Pressure_Sensor.c
            @author         : Osama Youssef
            @brief          : program file for Pressure controller project functions
                                           and states definations
 ******************************************************************************************/
#include <Pressure_Sensor.h>


// Defination of variables : Pressure_Sensor_Reading
// Pressure_Sensor_Reading is the pressure of the cabin ,it become dangerous when increase above 20 bar

int Pressure_Sensor_Reading = 10 ;

// pointer to function to point to the current state
void (*Pressure_Sensor_state)();

// function to initialize the Pressure Sensor
void Pressure_Sensor_init()
{
    Pressure_Sensor_state = STATE_(Pressure_Sensor_reading);
}

//function for waiting state
STATE_Define(Pressure_Sensor_reading)
{
    Pressure_Sensor_Reading = getPressureVal();   // get the prssure reading from the sensor
    set_Pressure_Reading(Pressure_Sensor_Reading); // send the reading to the main function
    Pressure_Sensor_state = STATE_(Pressure_Sensor_waiting); // change the current state to be in waiting state

}


//function for reading state
STATE_Define(Pressure_Sensor_waiting)
{
    Delay(1000);  // make delay 1000
    Pressure_Sensor_state = STATE_(Pressure_Sensor_reading);  // change the current state to be in reading state to read anthor value
}
```

● Main Function Diagram :

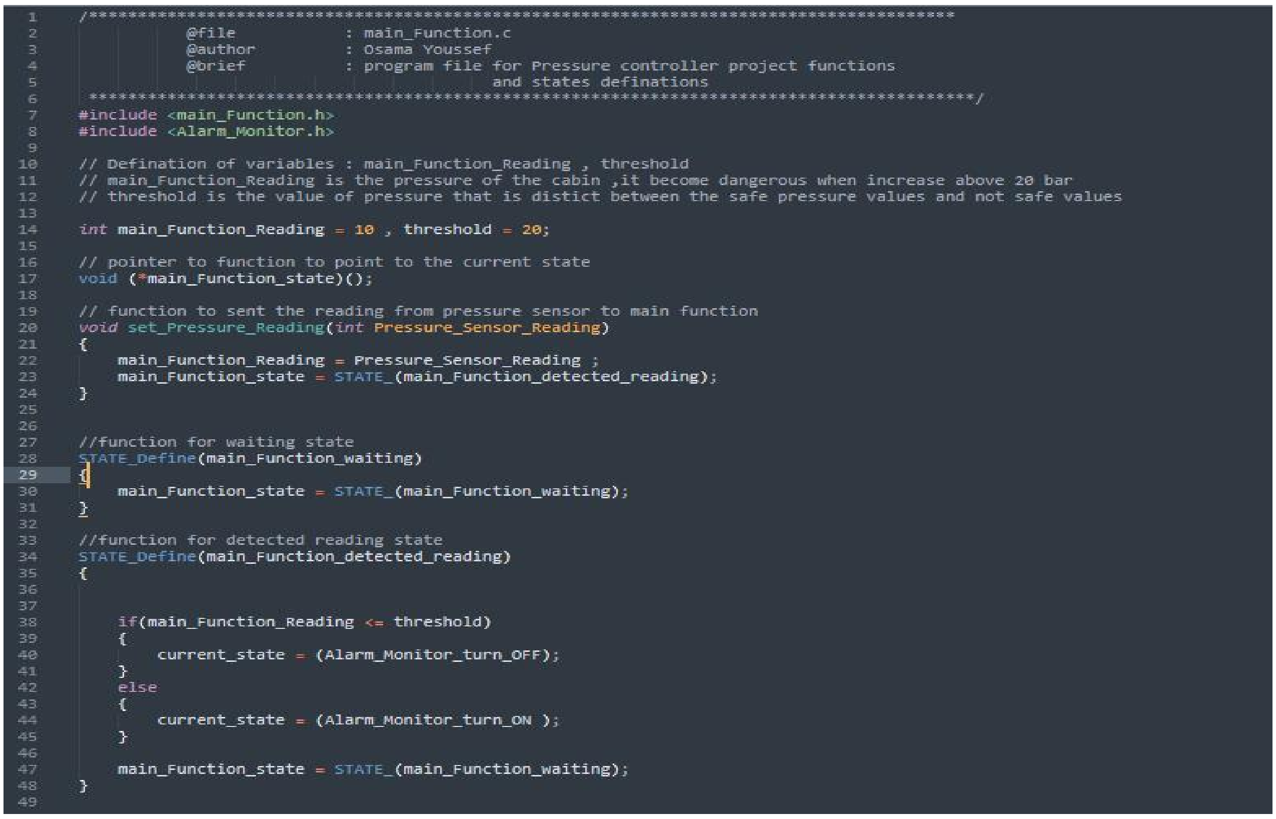

Waiting
Set_Pressure_Reading(Psensor_Reading)
Psensor_Detected_Reading
[Psensor_Reading <= Threshold]
[Psensor_Reading > Threshold]
Turn_Alarm_OFF()
Turn_Alarm_ON()

- **Main Function Header File :**



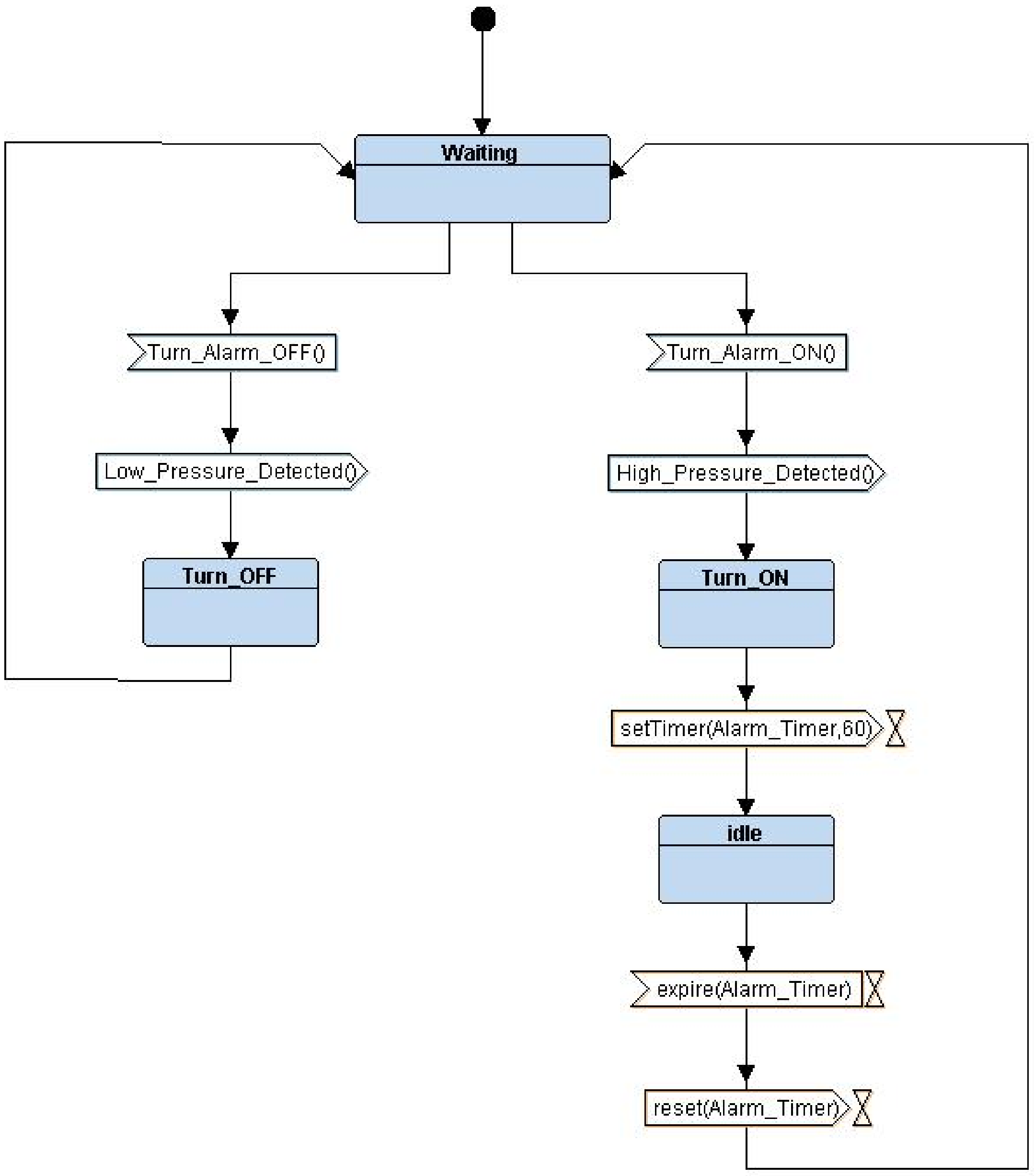
```c
/**********************************************************************************************
            @file           : main_Functionr.h
            @author         : Osama Youssef
            @brief          : Header file for Pressure controller project functions
                                          and states prototypes
 **********************************************************************************************/

#ifndef MAIN_FUNCTION_H_
#define MAIN_FUNCTION_H_

#include<driver.h>

// Define the states of the main Algorith
// there are two states : main Functionrwaiting  or main Functionr reading
typedef enum
{
    main_Function_waiting ,
    main_Function_detected_reading

}main_Function_state_ID;

// prototypes for states functions
STATE_Define(main_Function_waiting); //function for waiting state
STATE_Define(main_Function_detected_reading); //function for detected reading state

// extern the pointer to function to be viewed at main since we include main_Function.h  at main.c
// to avoid a linking error
extern void (*main_Function_state)();

#endif /* MAIN_FUNCTION_H_ */

```

- **Main Function Program File :**

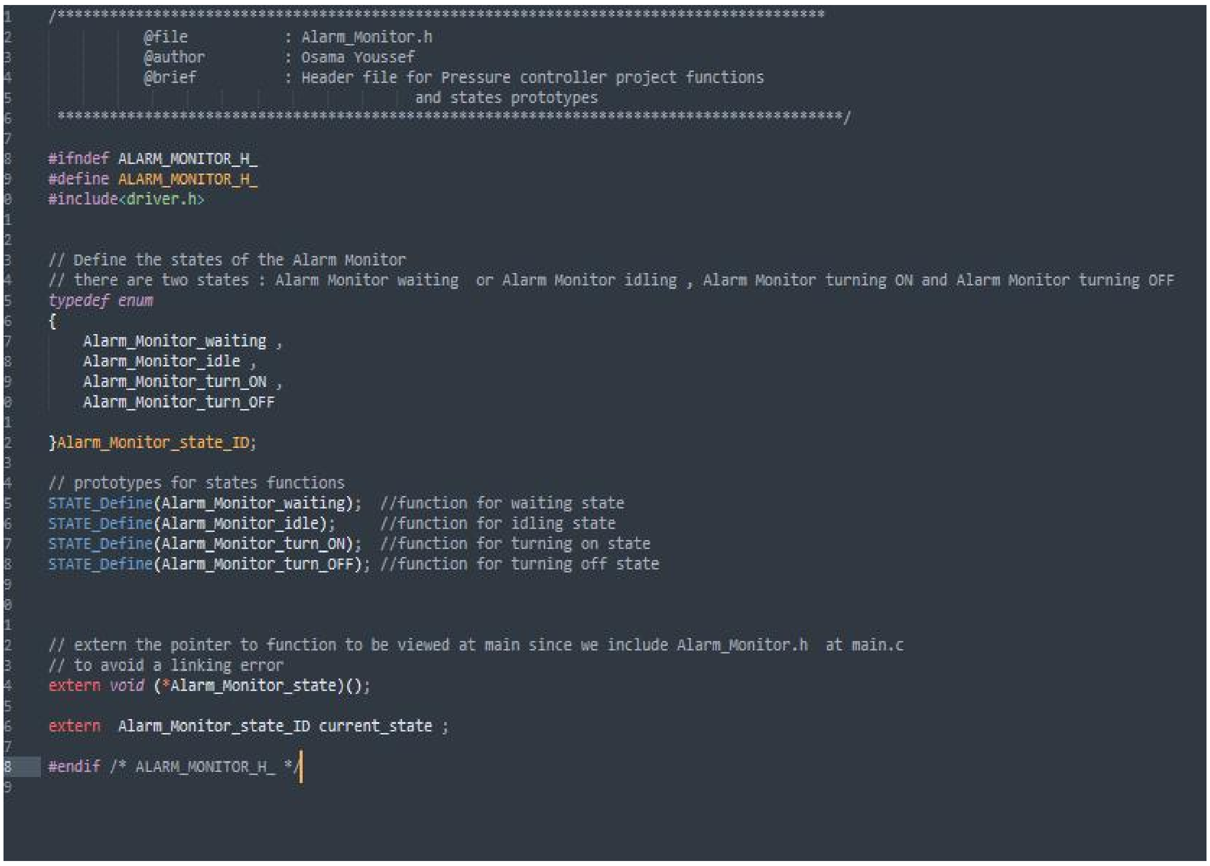


```c
/**********************************************************************************************
            @file           : main_Function.c
            @author         : Osama Youssef
            @brief          : program file for Pressure controller project functions
                                          and states definations
 **********************************************************************************************/
#include <main_Function.h>
#include <Alarm_Monitor.h>

// Defination of variables : main_Function_Reading , threshold
// main_Function_Reading is the pressure of the cabin ,it become dangerous when increase above 20 bar
// threshold is the value of pressure that is distict between the safe pressure values and not safe values

int main_Function_Reading = 10 , threshold = 20;

// pointer to function to point to the current state
void (*main_Function_state)();

// function to sent the reading from pressure sensor to main function
void set_Pressure_Reading(int Pressure_Sensor_Reading)
{
    main_Function_Reading = Pressure_Sensor_Reading ;
    main_Function_state = STATE_(main_Function_detected_reading);
}


//function for waiting state
STATE_Define(main_Function_waiting)
{
    main_Function_state = STATE_(main_Function_waiting);
}

//function for detected reading state
STATE_Define(main_Function_detected_reading)
{


    if(main_Function_Reading <= threshold)
    {
        current_state = (Alarm_Monitor_turn_OFF);
    }
    else
    {
        current_state = (Alarm_Monitor_turn_ON );
    }

    main_Function_state = STATE_(main_Function_waiting);
}

```

- **Alarm Monitor Diagram :**

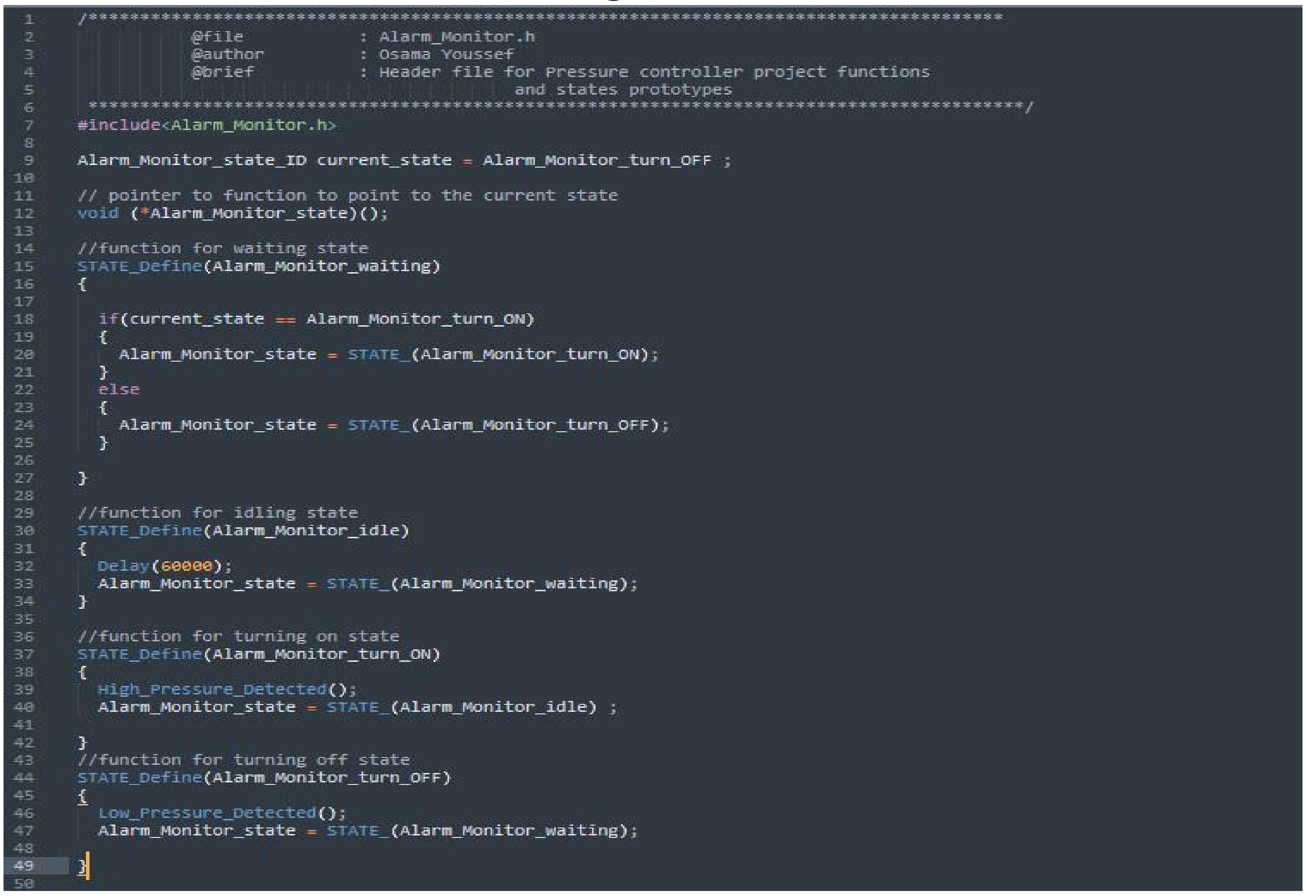


- **Alarm Monitor Header File :**



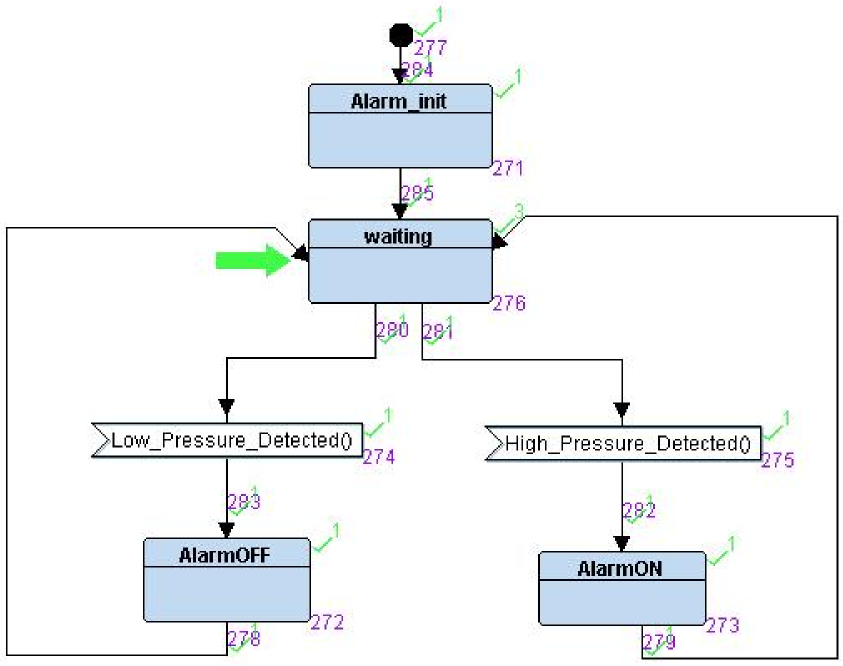
```c
/*****************************************************************************************
            @file           : Alarm_Monitor.h
            @author         : Osama Youssef
            @brief          : Header file for Pressure controller project functions
                                          and states prototypes
 ******************************************************************************************/

#ifndef ALARM_MONITOR_H_
#define ALARM_MONITOR_H_
#include<driver.h>


// Define the states of the Alarm Monitor
// there are two states : Alarm Monitor waiting  or Alarm Monitor idling , Alarm Monitor turning ON and Alarm Monitor turning OFF
typedef enum
{
    Alarm_Monitor_waiting ,
    Alarm_Monitor_idle ,
    Alarm_Monitor_turn_ON ,
    Alarm_Monitor_turn_OFF

}Alarm_Monitor_state_ID;

// prototypes for states functions
STATE_Define(Alarm_Monitor_waiting);  //function for waiting state
STATE_Define(Alarm_Monitor_idle);     //function for idling state
STATE_Define(Alarm_Monitor_turn_ON);  //function for turning on state
STATE_Define(Alarm_Monitor_turn_OFF); //function for turning off state



// extern the pointer to function to be viewed at main since we include Alarm_Monitor.h  at main.c
// to avoid a linking error
extern void (*Alarm_Monitor_state)();

extern  Alarm_Monitor_state_ID current_state ;

#endif /* ALARM_MONITOR_H_ */

```

● Alarm Monitor Program File :

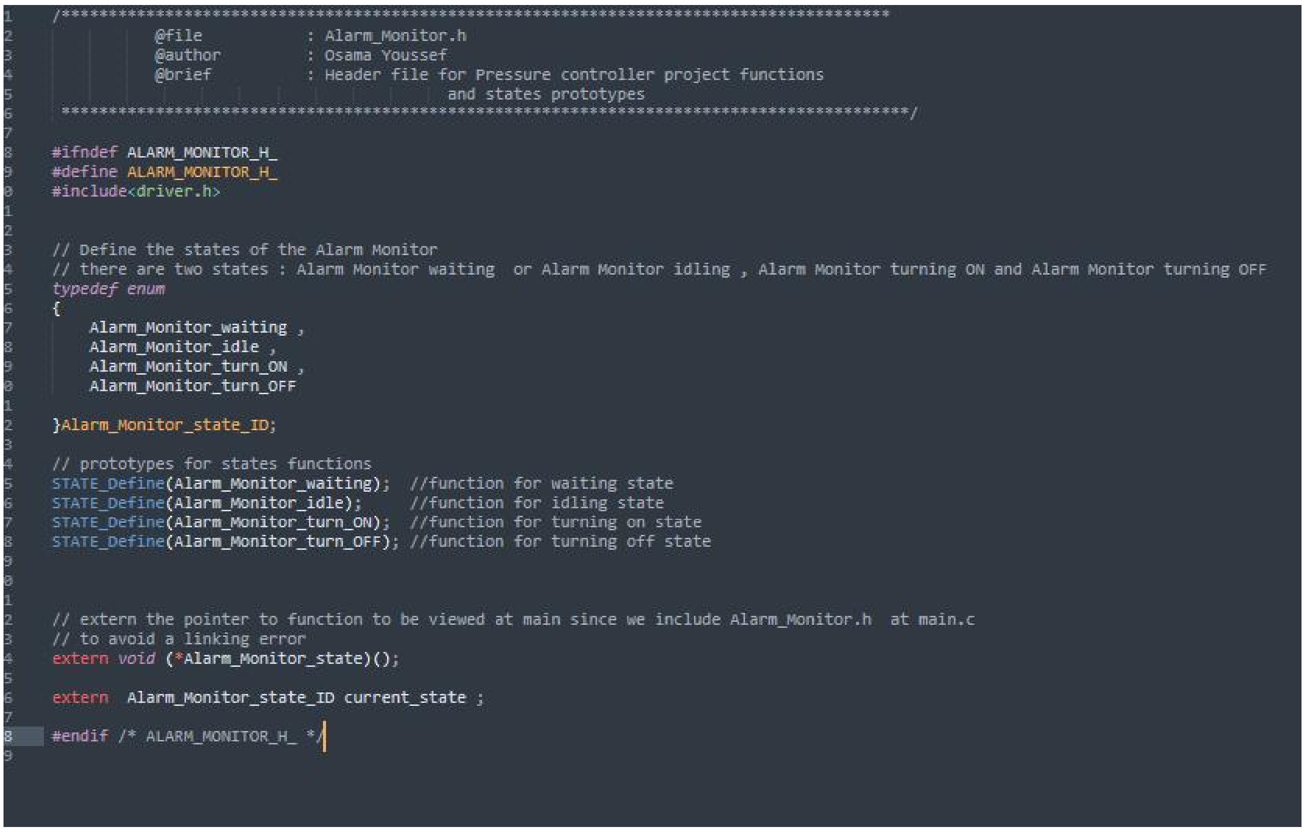


```c
/*****************************************************************************************
             @file           : Alarm_Monitor.h
             @author         : Osama Youssef
             @brief          : Header file for Pressure controller project functions
                                          and states prototypes
 ******************************************************************************************/
#include<Alarm_Monitor.h>

Alarm_Monitor_state_ID current_state = Alarm_Monitor_turn_OFF ;

// pointer to function to point to the current state
void (*Alarm_Monitor_state)();

//function for waiting state
STATE_Define(Alarm_Monitor_waiting)
{

  if(current_state == Alarm_Monitor_turn_ON)
  {
    Alarm_Monitor_state = STATE_(Alarm_Monitor_turn_ON);
  }
  else
  {
    Alarm_Monitor_state = STATE_(Alarm_Monitor_turn_OFF);
  }

}

//function for idling state
STATE_Define(Alarm_Monitor_idle)
{
  Delay(60000);
  Alarm_Monitor_state = STATE_(Alarm_Monitor_waiting);
}

//function for turning on state
STATE_Define(Alarm_Monitor_turn_ON)
{
  High_Pressure_Detected();
  Alarm_Monitor_state = STATE_(Alarm_Monitor_idle) ;

}
//function for turning off state
STATE_Define(Alarm_Monitor_turn_OFF)
{
  Low_Pressure_Detected();
  Alarm_Monitor_state = STATE_(Alarm_Monitor_waiting);

}

```

● Alarm Monitor Driver Diagram :

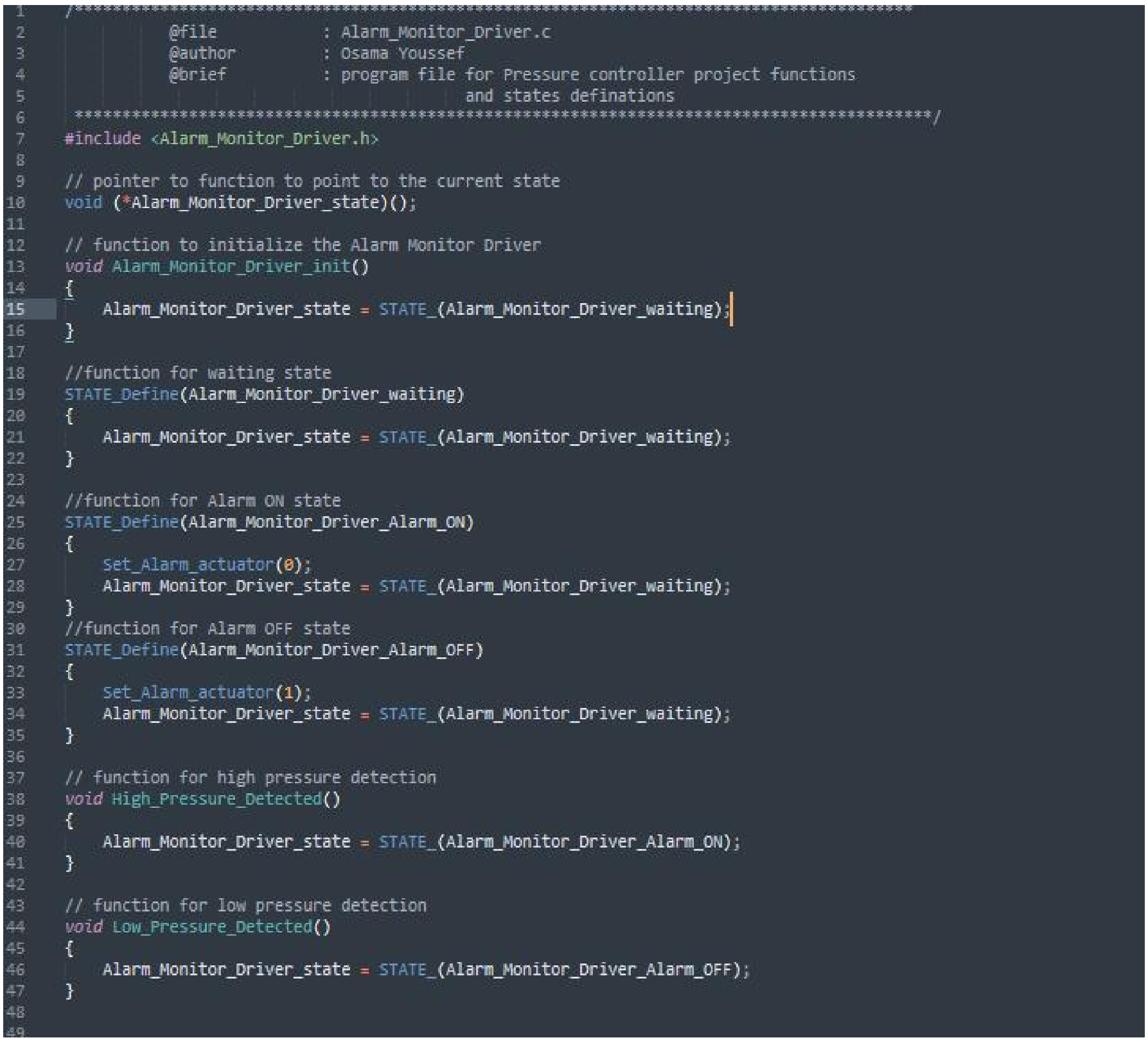


● Alarm Monitor Driver Header File :



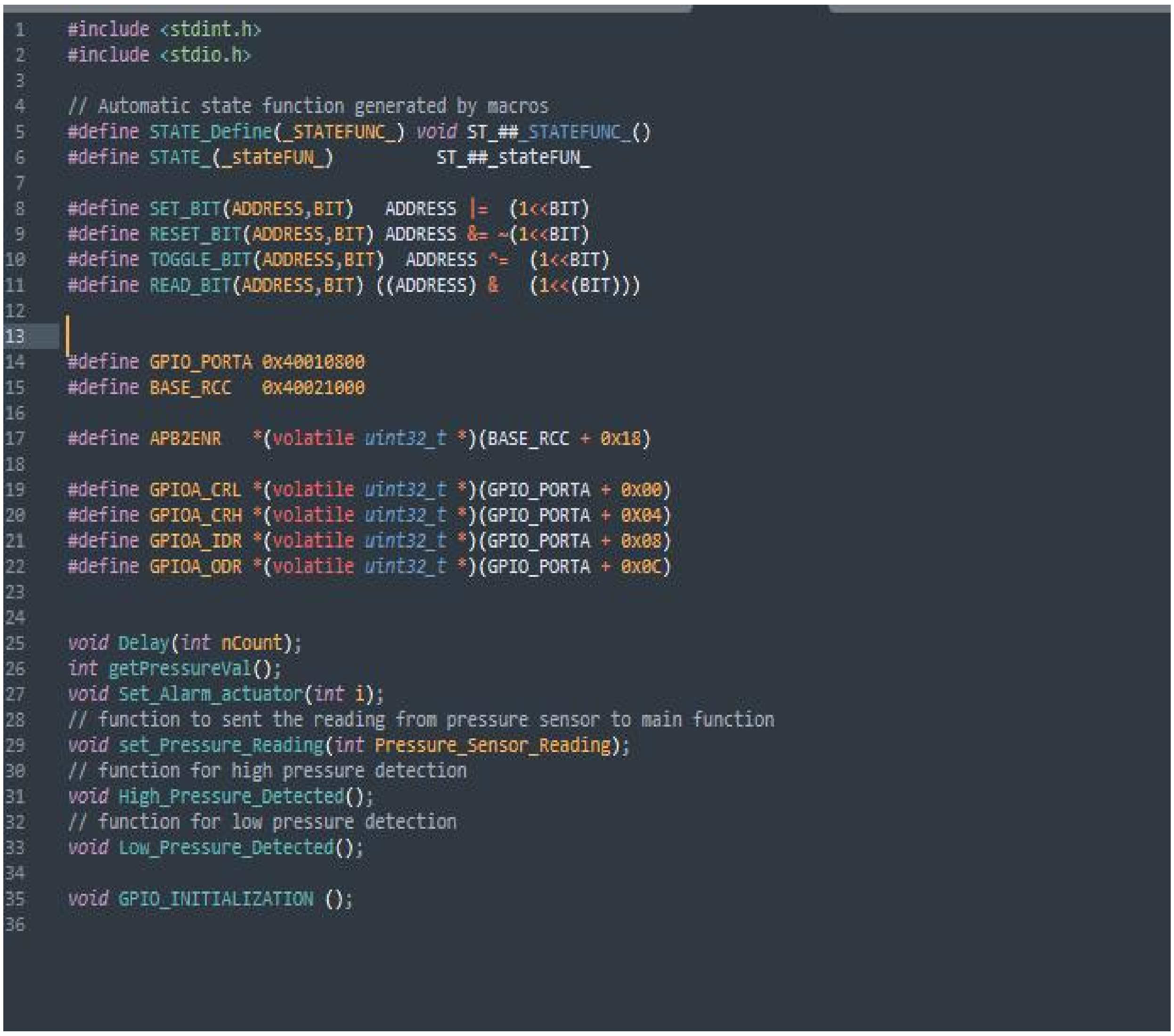
```c
/*******************************************************************************************
			@file		: Alarm_Monitor.h
			@author		: Osama Youssef
			@brief		: Header file for Pressure controller project functions
								  and states prototypes
 *******************************************************************************************/

#ifndef ALARM_MONITOR_H_
#define ALARM_MONITOR_H_
#include<driver.h>


// Define the states of the Alarm Monitor
// there are two states : Alarm Monitor waiting  or Alarm Monitor idling , Alarm Monitor turning ON and Alarm Monitor turning OFF
typedef enum
{
	Alarm_Monitor_waiting ,
	Alarm_Monitor_idle ,
	Alarm_Monitor_turn_ON ,
	Alarm_Monitor_turn_OFF

}Alarm_Monitor_state_ID;

// prototypes for states functions
STATE_Define(Alarm_Monitor_waiting);  //function for waiting state
STATE_Define(Alarm_Monitor_idle);     //function for idling state
STATE_Define(Alarm_Monitor_turn_ON);  //function for turning on state
STATE_Define(Alarm_Monitor_turn_OFF); //function for turning off state



// extern the pointer to function to be viewed at main since we include Alarm_Monitor.h  at main.c
// to avoid a linking error
extern void (*Alarm_Monitor_state)();

extern  Alarm_Monitor_state_ID current_state ;

#endif /* ALARM_MONITOR_H_ */
```

- **Alarm Monitor Driver Program File :**



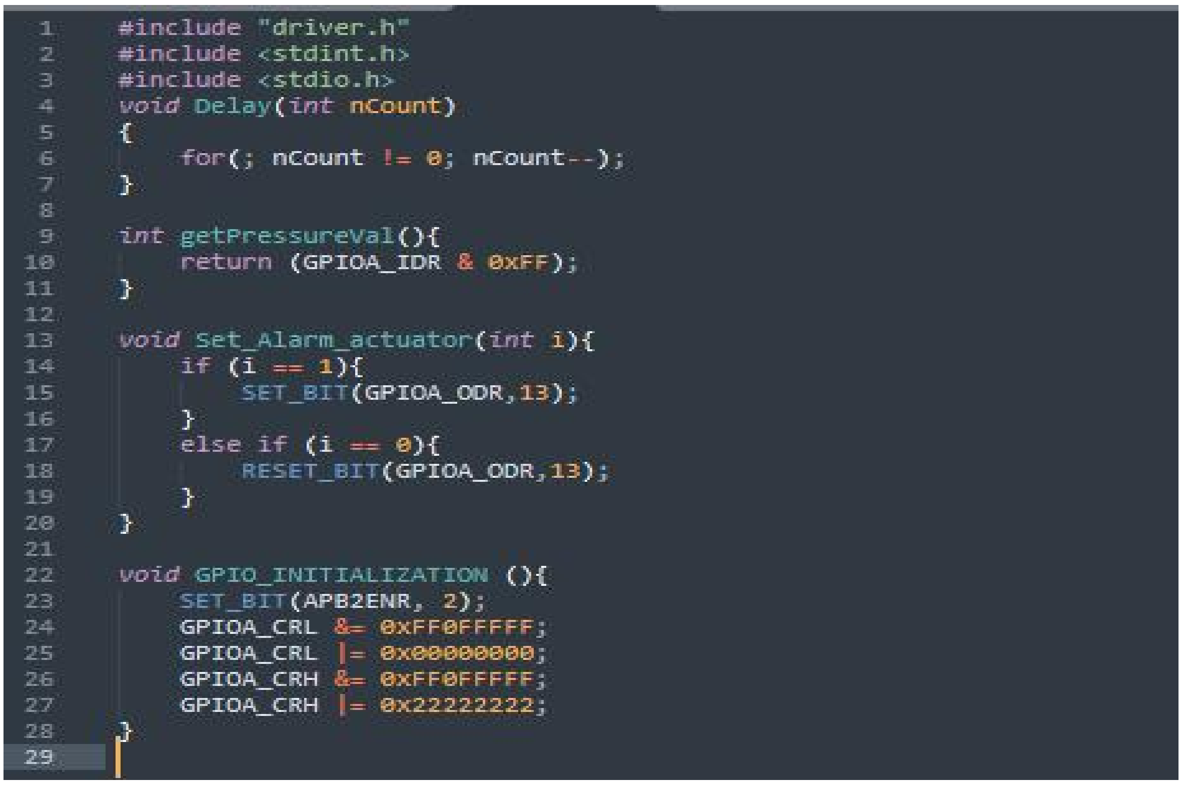
```c
/**********************************************************************************************
            @file           : Alarm_Monitor_Driver.c
            @author         : Osama Youssef
            @brief          : program file for Pressure controller project functions
                                          and states definations
 *********************************************************************************************/
#include <Alarm_Monitor_Driver.h>

// pointer to function to point to the current state
void (*Alarm_Monitor_Driver_state)();

// function to initialize the Alarm Monitor Driver
void Alarm_Monitor_Driver_init()
{
    Alarm_Monitor_Driver_state = STATE_(Alarm_Monitor_Driver_waiting);
}

//function for waiting state
STATE_Define(Alarm_Monitor_Driver_waiting)
{
    Alarm_Monitor_Driver_state = STATE_(Alarm_Monitor_Driver_waiting);
}

//function for Alarm ON state
STATE_Define(Alarm_Monitor_Driver_Alarm_ON)
{
    Set_Alarm_actuator(0);
    Alarm_Monitor_Driver_state = STATE_(Alarm_Monitor_Driver_waiting);
}
//function for Alarm OFF state
STATE_Define(Alarm_Monitor_Driver_Alarm_OFF)
{
    Set_Alarm_actuator(1);
    Alarm_Monitor_Driver_state = STATE_(Alarm_Monitor_Driver_waiting);
}

// function for high pressure detection
void High_Pressure_Detected()
{
    Alarm_Monitor_Driver_state = STATE_(Alarm_Monitor_Driver_Alarm_ON);
}

// function for low pressure detection
void Low_Pressure_Detected()
{
    Alarm_Monitor_Driver_state = STATE_(Alarm_Monitor_Driver_Alarm_OFF);
}
```

● Driver Header File :



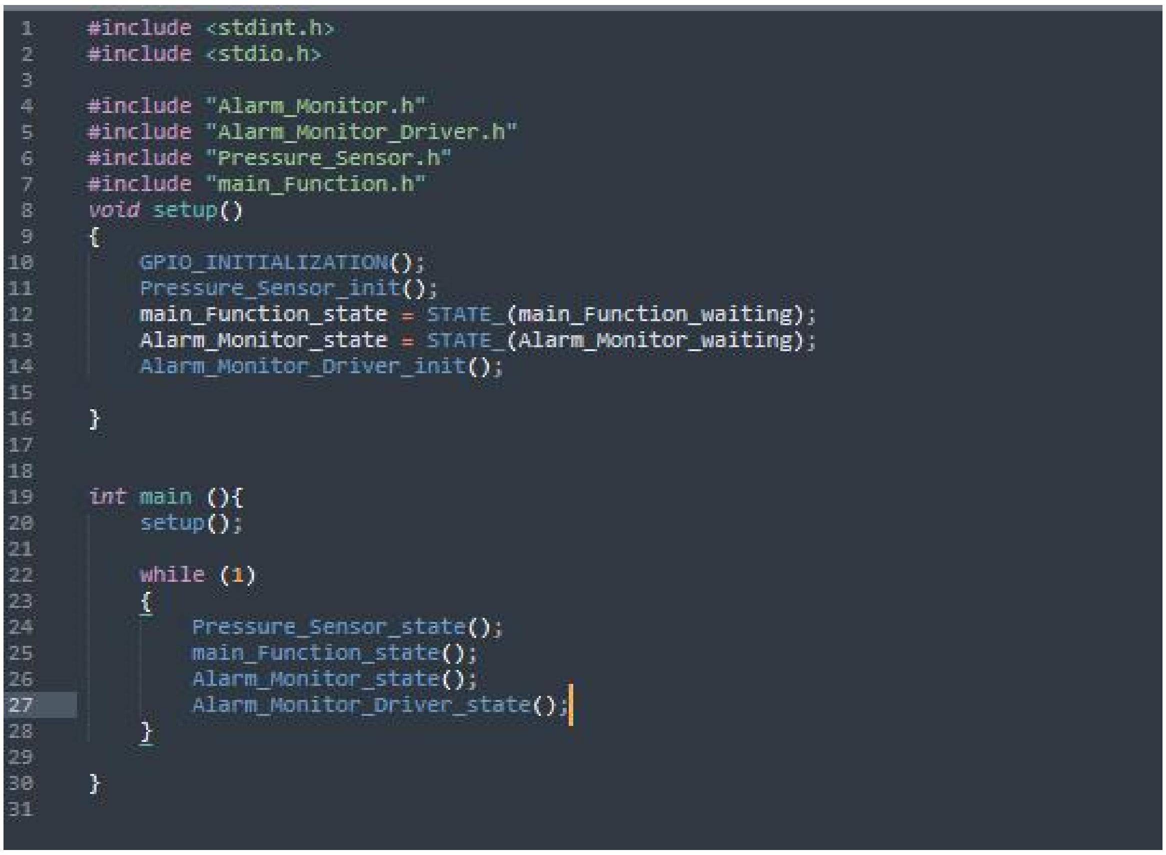
```c
#include <stdint.h>
#include <stdio.h>

// Automatic state function generated by macros
#define STATE_Define(_STATEFUNC_) void ST_##_STATEFUNC_()
#define STATE_(_stateFUN_)          ST_##_stateFUN_

#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);
// function to sent the reading from pressure sensor to main function
void set_Pressure_Reading(int Pressure_Sensor_Reading);
// function for high pressure detection
void High_Pressure_Detected();
// function for low pressure detection
void Low_Pressure_Detected();

void GPIO_INITIALIZATION ();
```

● Driver Program File :



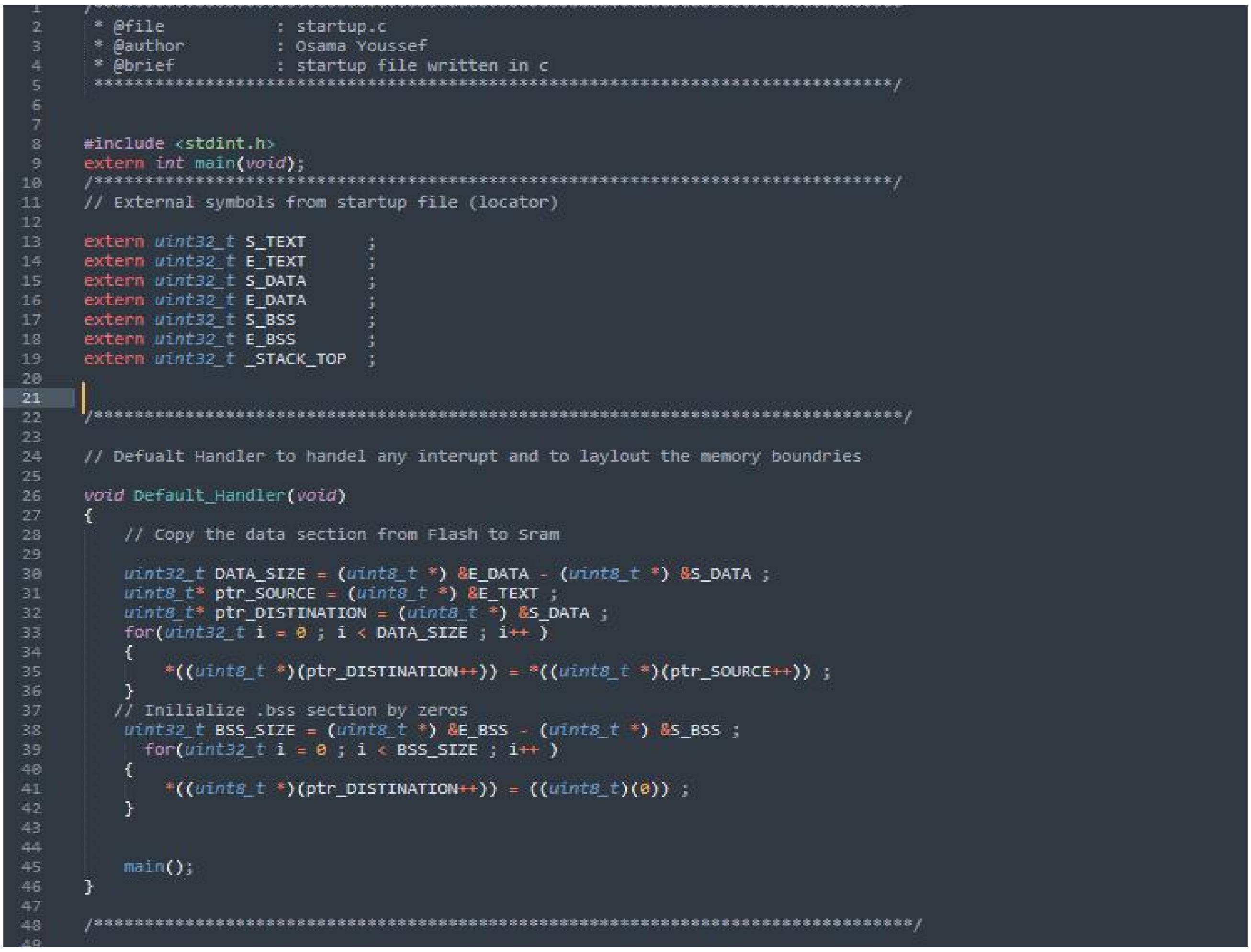
```c
#include "driver.h"
#include <stdint.h>
#include <stdio.h>
void Delay(int nCount)
{
    for(; nCount != 0; nCount--);
}

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

void Set_Alarm_actuator(int i){
    if (i == 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;
}

```

● Main Program File :

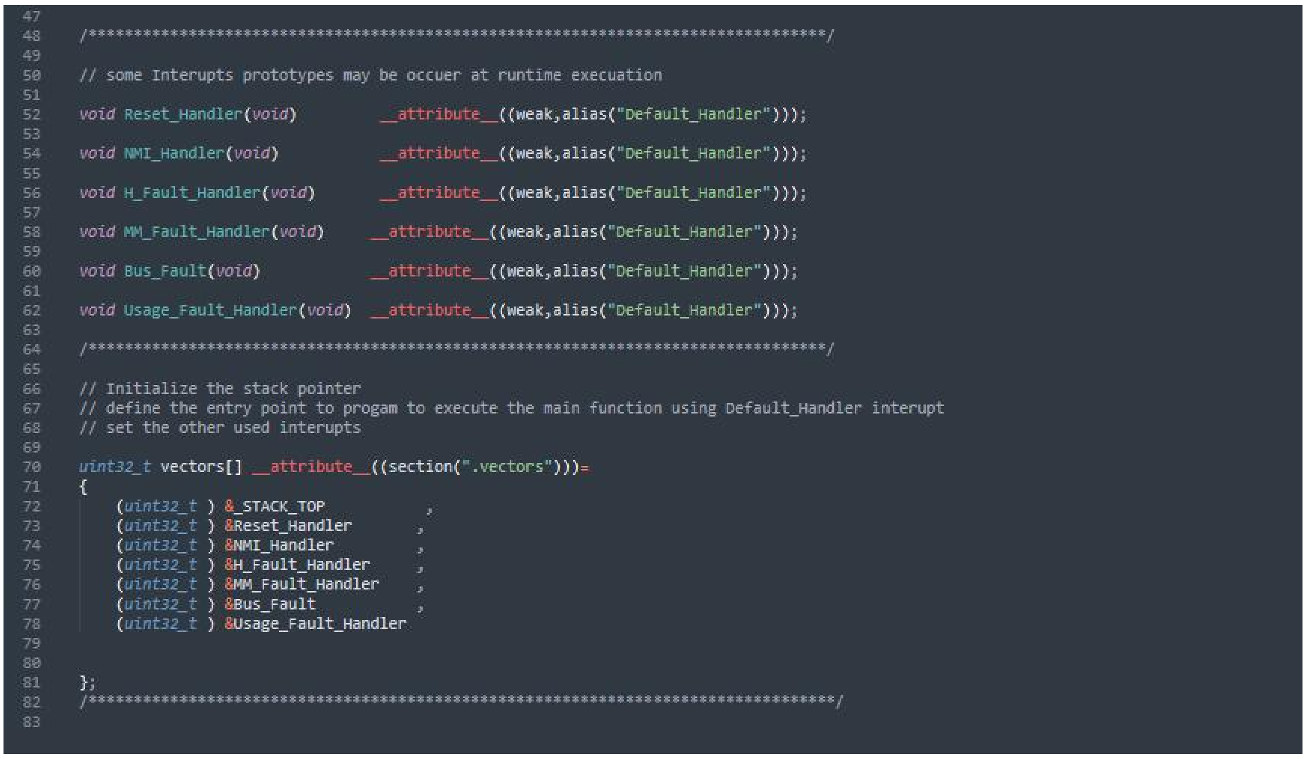


```c
#include <stdint.h>
#include <stdio.h>

#include "Alarm_Monitor.h"
#include "Alarm_Monitor_Driver.h"
#include "Pressure_Sensor.h"
#include "main_Function.h"
void setup()
{
    GPIO_INITIALIZATION();
    Pressure_Sensor_init();
    main_Function_state = STATE_(main_Function_waiting);
    Alarm_Monitor_state = STATE_(Alarm_Monitor_waiting);
    Alarm_Monitor_Driver_init();

}


int main (){
    setup();

    while (1)
    {
        Pressure_Sensor_state();
        main_Function_state();
        Alarm_Monitor_state();
        Alarm_Monitor_Driver_state();
    }

}

```

● Startup Program File :

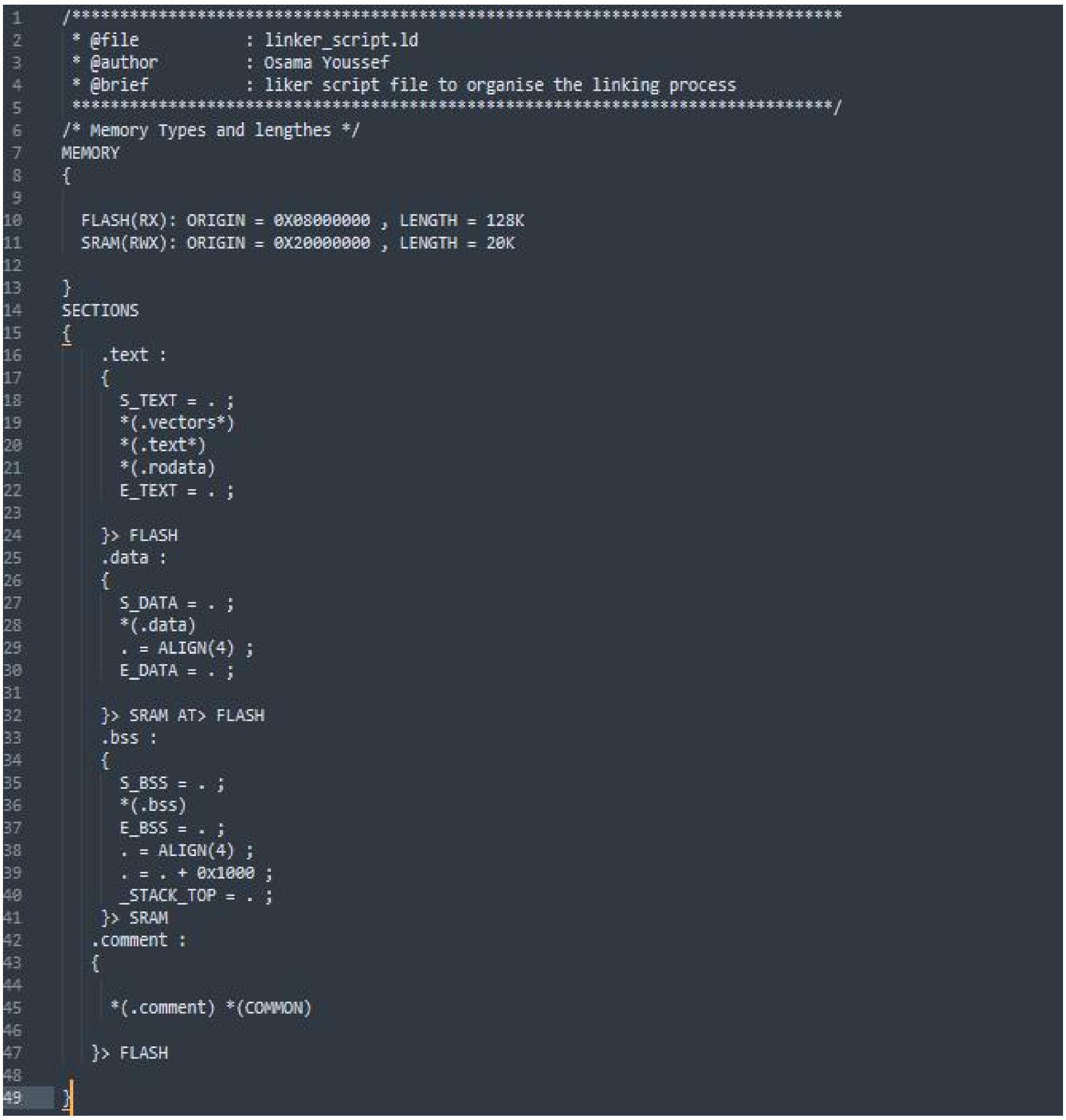


```c
/*********************************************************************************
 * @file           : startup.c
 * @author         : Osama Youssef
 * @brief          : startup file written in c
 *********************************************************************************/


#include <stdint.h>
extern int main(void);
/*********************************************************************************/
// External symbols from startup file (locator)

extern uint32_t S_TEXT      ;
extern uint32_t E_TEXT      ;
extern uint32_t S_DATA      ;
extern uint32_t E_DATA      ;
extern uint32_t S_BSS       ;
extern uint32_t E_BSS       ;
extern uint32_t _STACK_TOP  ;


/**********************************************************************************/

// Defualt Handler to handel any interupt and to laylout the memory boundries

void Default_Handler(void)
{
    // Copy the data section from Flash to Sram

    uint32_t DATA_SIZE = (uint8_t *) &E_DATA - (uint8_t *) &S_DATA ;
    uint8_t* ptr_SOURCE = (uint8_t *) &E_TEXT ;
    uint8_t* ptr_DISTINATION = (uint8_t *) &S_DATA ;
    for(uint32_t i = 0 ; i < DATA_SIZE ; i++ )
    {
        *((uint8_t *)(ptr_DISTINATION++)) = *((uint8_t *)(ptr_SOURCE++)) ;
    }
   // Inilialize .bss section by zeros
    uint32_t BSS_SIZE = (uint8_t *) &E_BSS - (uint8_t *) &S_BSS ;
      for(uint32_t i = 0 ; i < BSS_SIZE ; i++ )
    {
        *((uint8_t *)(ptr_DISTINATION++)) = ((uint8_t)(0)) ;
    }


    main();
}

/**********************************************************************************/
```



```c

/*********************************************************************************/

// some Interupts prototypes may be occuer at runtime execuation

void Reset_Handler(void)         __attribute__((weak,alias("Default_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")));

/*********************************************************************************/

// Initialize the stack pointer
// define the entry point to progam to execute the main function using Default_Handler interupt
// set the other used interupts

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


};
/**********************************************************************************/

```

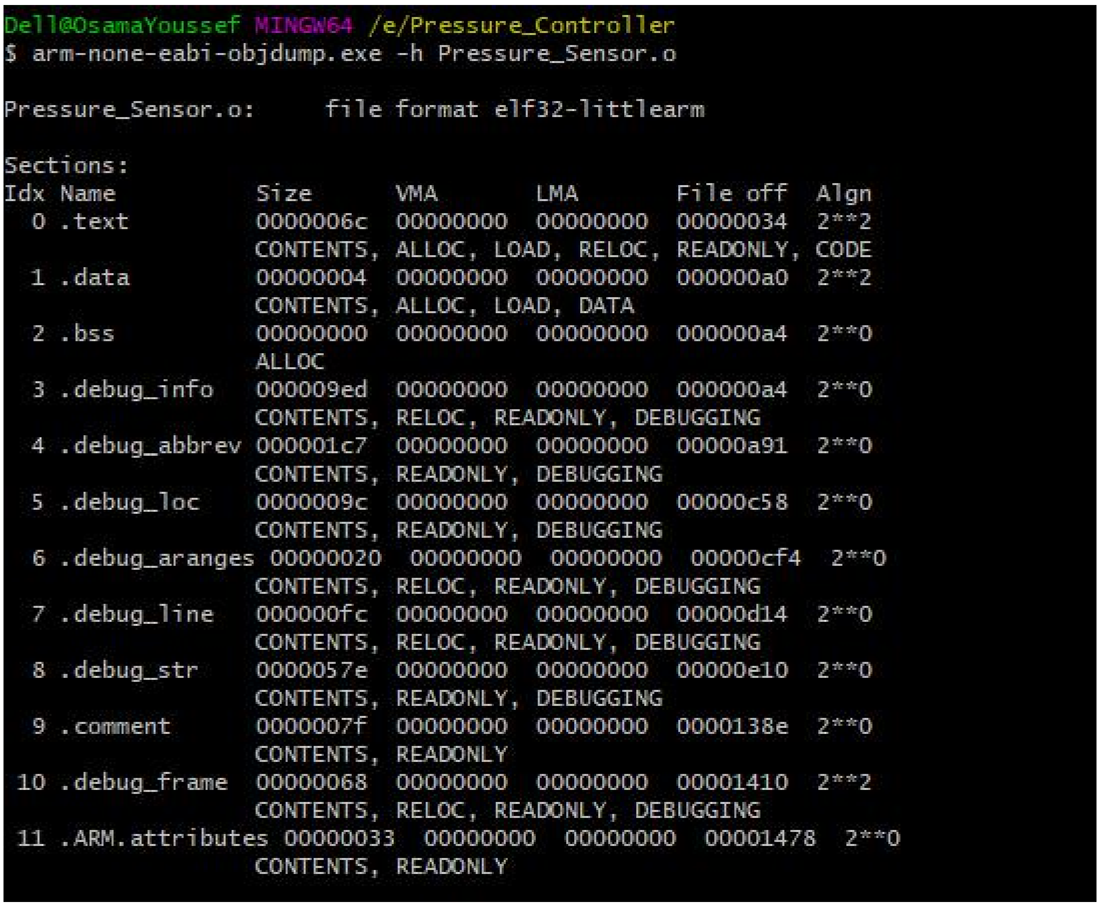
● Linker script.ld :



```
/*******************************************************************************
 * @file           : linker_script.ld
 * @author         : Osama Youssef
 * @brief          : liker script file to organise the linking process
 ******************************************************************************/
/* Memory Types and lengthes */
MEMORY
{

  FLASH(RX): ORIGIN = 0X08000000 , LENGTH = 128K
  SRAM(RWX): ORIGIN = 0X20000000 , LENGTH = 20K

}
SECTIONS
{
    .text :
    {
      S_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
   .comment :
   {

     *(.comment) *(COMMON)

   }> FLASH

}
```

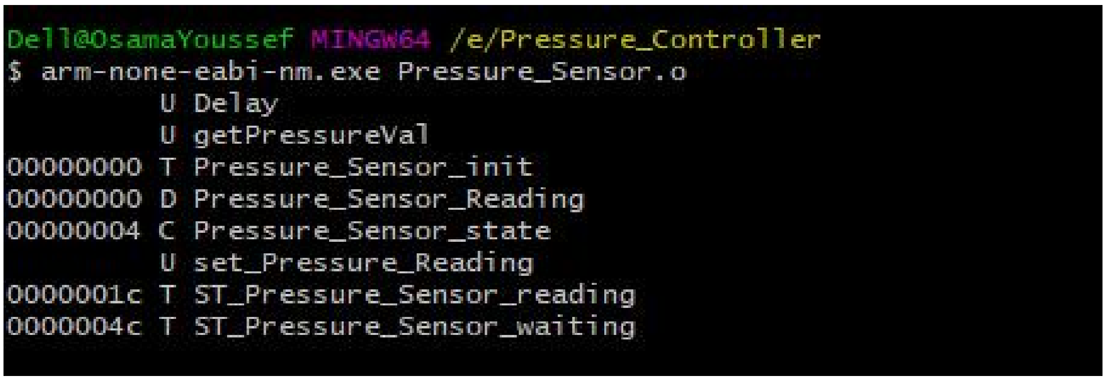
➢Sections and symbols for object files :

- **Pressure Sensor object file sections :**



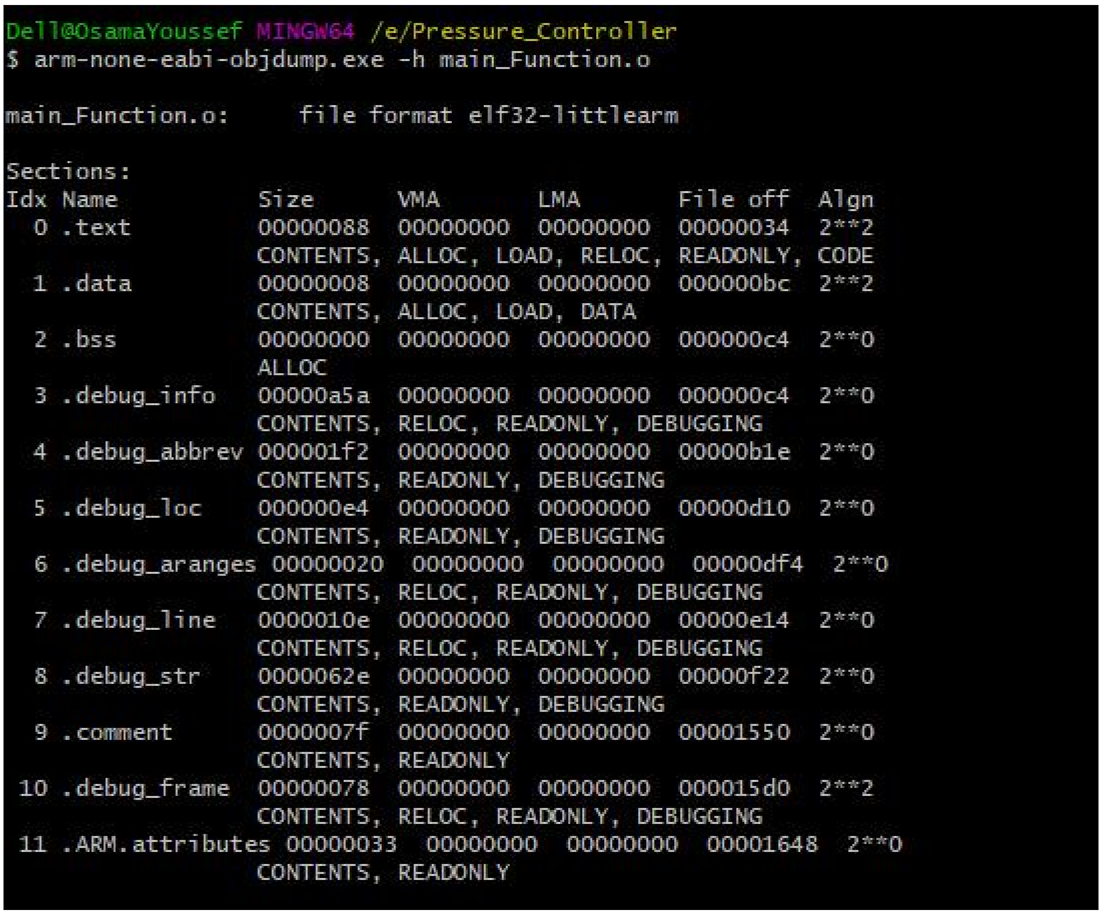
```
Dell@OsamaYoussef MINGW64 /e/Pressure_Controller
$ arm-none-eabi-objdump.exe -h Pressure_Sensor.o

Pressure_Sensor.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000006c  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000004  00000000  00000000  000000a0  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000a4  2**0
                  ALLOC
  3 .debug_info   000009ed  00000000  00000000  000000a4  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 000001c7  00000000  00000000  00000a91  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    0000009c  00000000  00000000  00000c58  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  00000cf4  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   000000fc  00000000  00000000  00000d14  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str    0000057e  00000000  00000000  00000e10  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      0000007f  00000000  00000000  0000138e  2**0
                  CONTENTS, READONLY
 10 .debug_frame  00000068  00000000  00000000  00001410  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
 11 .ARM.attributes 00000033  00000000  00000000  00001478  2**0
                  CONTENTS, READONLY
```

- **Pressure Sensor object file symbols :**



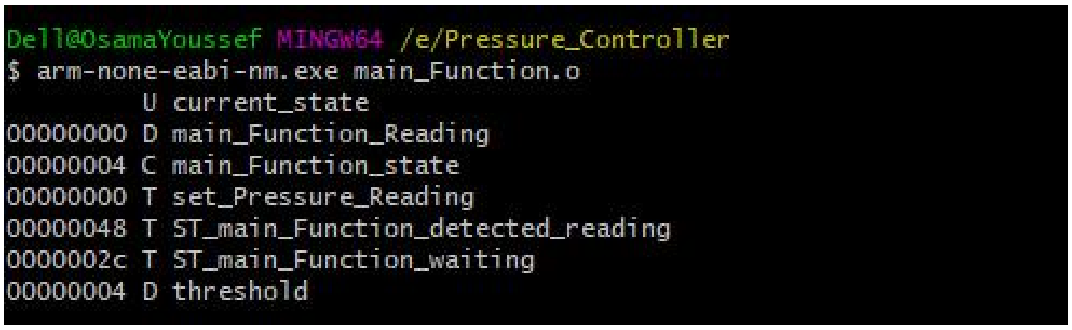
```
Dell@OsamaYoussef MINGW64 /e/Pressure_Controller
$ arm-none-eabi-nm.exe Pressure_Sensor.o
         U Delay
         U getPressureVal
00000000 T Pressure_Sensor_init
00000000 D Pressure_Sensor_Reading
00000004 C Pressure_Sensor_state
         U set_Pressure_Reading
0000001c T ST_Pressure_Sensor_reading
0000004c T ST_Pressure_Sensor_waiting
```

● Main Function object file sections :



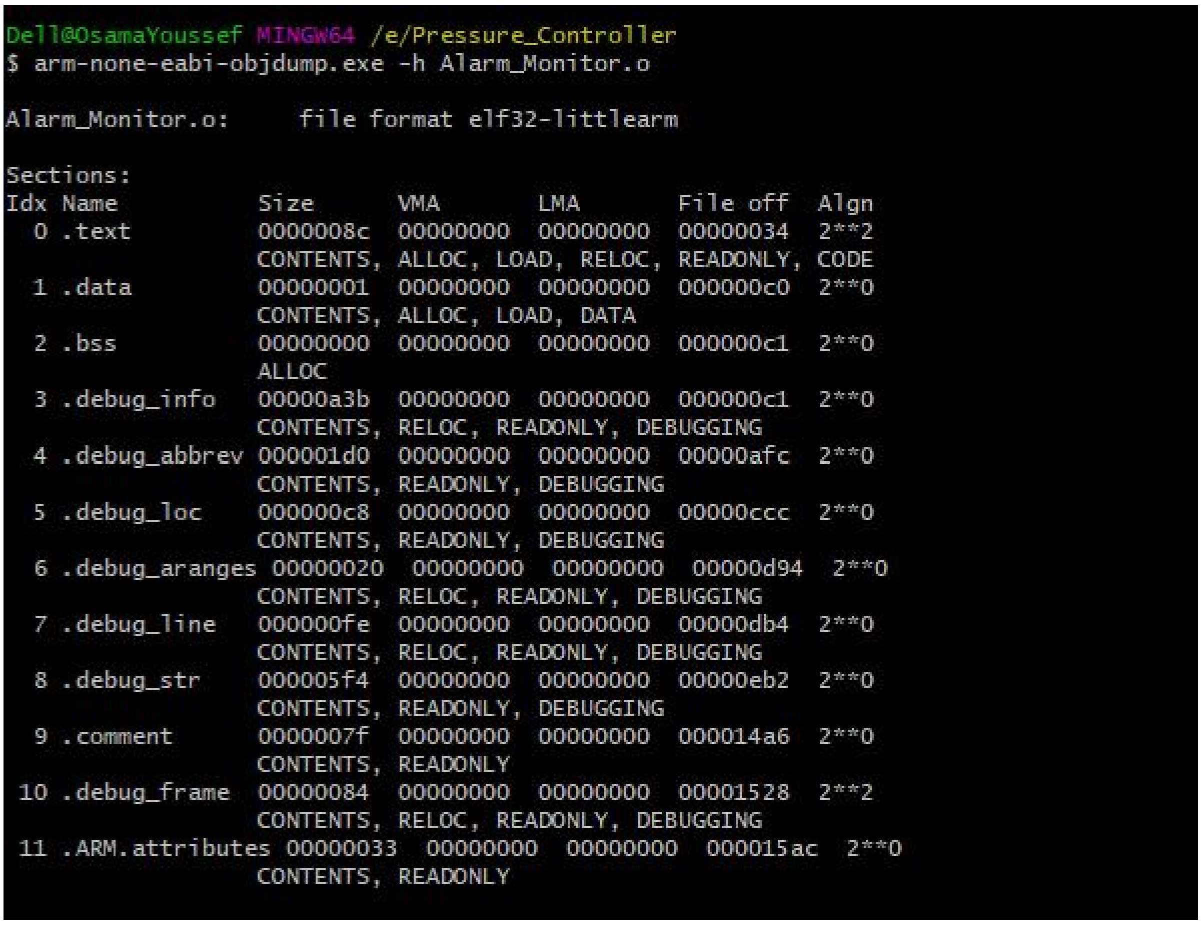
```
Dell@OsamaYoussef MINGW64 /e/Pressure_Controller
$ arm-none-eabi-objdump.exe -h main_Function.o

main_Function.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000088  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000008  00000000  00000000  000000bc  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000c4  2**0
                  ALLOC
  3 .debug_info   00000a5a  00000000  00000000  000000c4  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 000001f2  00000000  00000000  00000b1e  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    000000e4  00000000  00000000  00000d10  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  00000df4  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   0000010e  00000000  00000000  00000e14  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str    0000062e  00000000  00000000  00000f22  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      0000007f  00000000  00000000  00001550  2**0
                  CONTENTS, READONLY
 10 .debug_frame  00000078  00000000  00000000  000015d0  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
 11 .ARM.attributes 00000033  00000000  00000000  00001648  2**0
                  CONTENTS, READONLY
```

● Main Function object file symbols :



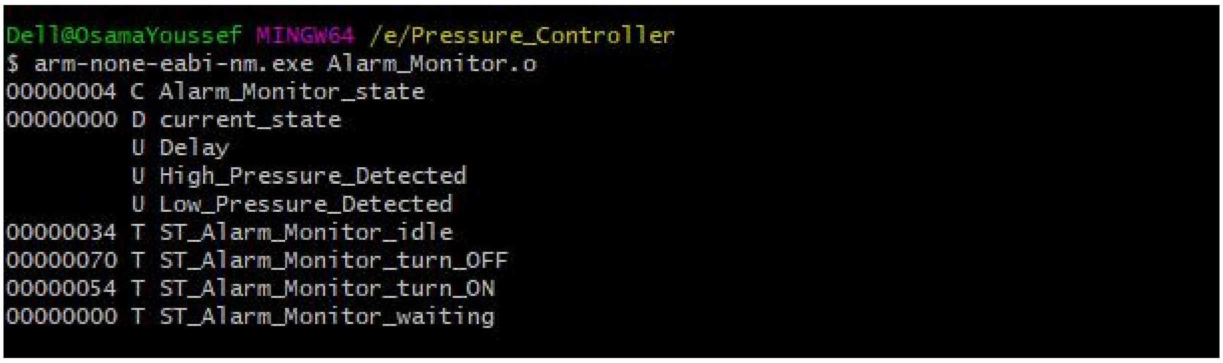
```
Dell@OsamaYoussef MINGW64 /e/Pressure_Controller
$ arm-none-eabi-nm.exe main_Function.o
         U current_state
00000000 D main_Function_Reading
00000004 C main_Function_state
00000000 T set_Pressure_Reading
00000048 T ST_main_Function_detected_reading
0000002c T ST_main_Function_waiting
00000004 D threshold
```

● Alarm Monitor object file sections :



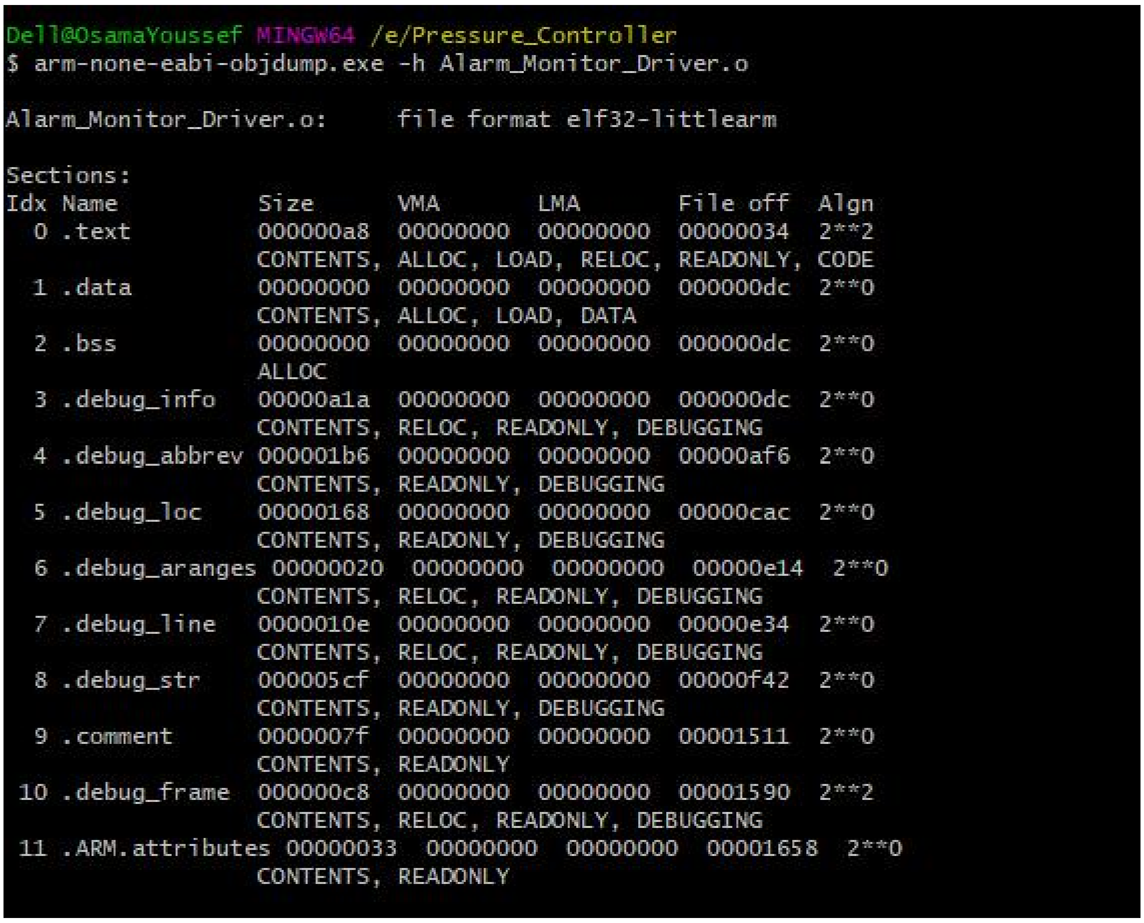
```
Dell@OsamaYoussef MINGW64 /e/Pressure_Controller
$ arm-none-eabi-objdump.exe -h Alarm_Monitor.o

Alarm_Monitor.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000008c  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000001  00000000  00000000  000000c0  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000c1  2**0
                  ALLOC
  3 .debug_info   00000a3b  00000000  00000000  000000c1  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 000001d0  00000000  00000000  00000afc  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    000000c8  00000000  00000000  00000ccc  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  00000d94  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   000000fe  00000000  00000000  00000db4  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str    000005f4  00000000  00000000  00000eb2  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      0000007f  00000000  00000000  000014a6  2**0
                  CONTENTS, READONLY
 10 .debug_frame  00000084  00000000  00000000  00001528  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
 11 .ARM.attributes 00000033  00000000  00000000  000015ac  2**0
                  CONTENTS, READONLY
```

● Alarm Monitor object file symbols :



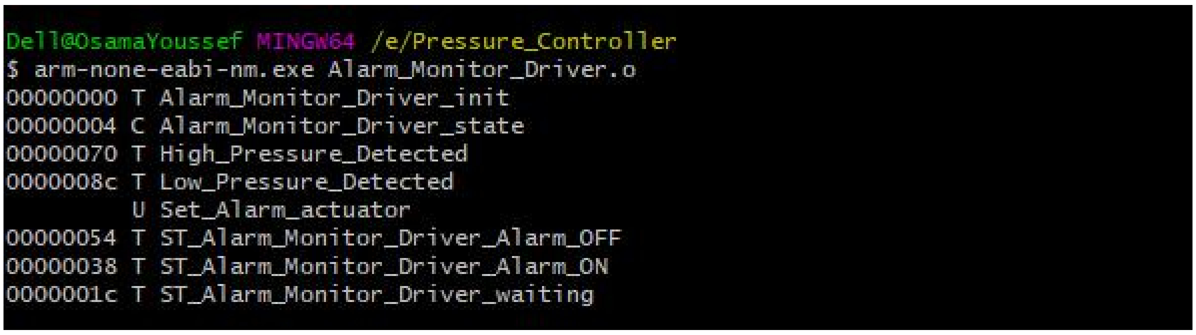
```
Dell@OsamaYoussef MINGW64 /e/Pressure_Controller
$ arm-none-eabi-nm.exe Alarm_Monitor.o
00000004 C Alarm_Monitor_state
00000000 D current_state
         U Delay
         U High_Pressure_Detected
         U Low_Pressure_Detected
00000034 T ST_Alarm_Monitor_idle
00000070 T ST_Alarm_Monitor_turn_OFF
00000054 T ST_Alarm_Monitor_turn_ON
00000000 T ST_Alarm_Monitor_waiting
```

- **Alarm Monitor Driver object file sections :**



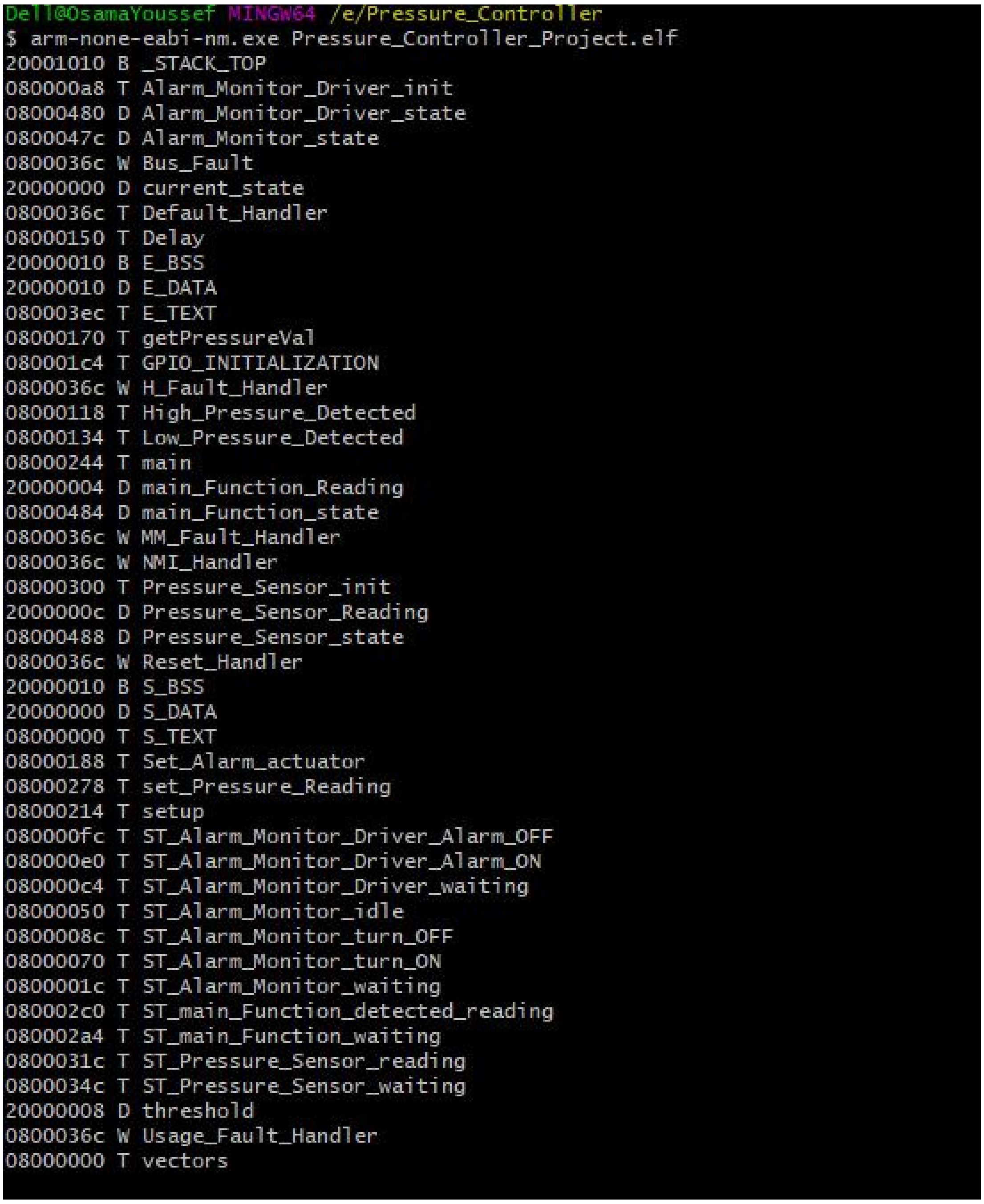
```
Dell@OsamaYoussef MINGW64 /e/Pressure_Controller
$ arm-none-eabi-objdump.exe -h Alarm_Monitor_Driver.o

Alarm_Monitor_Driver.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000000a8  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  000000dc  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000dc  2**0
                  ALLOC
  3 .debug_info   00000a1a  00000000  00000000  000000dc  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 000001b6  00000000  00000000  00000af6  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    00000168  00000000  00000000  00000cac  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  00000e14  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   0000010e  00000000  00000000  00000e34  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str    000005cf  00000000  00000000  00000f42  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      0000007f  00000000  00000000  00001511  2**0
                  CONTENTS, READONLY
 10 .debug_frame  000000c8  00000000  00000000  00001590  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
 11 .ARM.attributes 00000033  00000000  00000000  00001658  2**0
                  CONTENTS, READONLY
```

- **Alarm Monitor Driver object file symbols :**



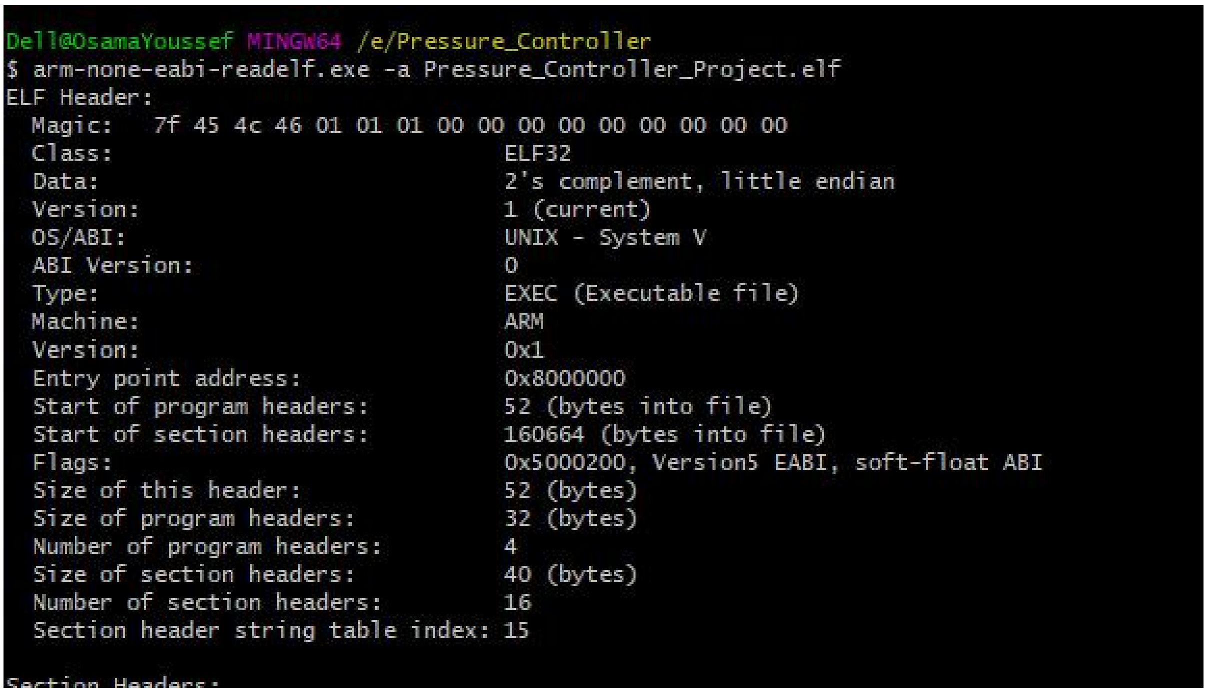
```
Dell@OsamaYoussef MINGW64 /e/Pressure_Controller
$ arm-none-eabi-nm.exe Alarm_Monitor_Driver.o
00000000 T Alarm_Monitor_Driver_init
00000004 C Alarm_Monitor_Driver_state
00000070 T High_Pressure_Detected
0000008c T Low_Pressure_Detected
         U Set_Alarm_actuator
00000054 T ST_Alarm_Monitor_Driver_Alarm_OFF
00000038 T ST_Alarm_Monitor_Driver_Alarm_ON
0000001c T ST_Alarm_Monitor_Driver_waiting
```

➢ Symbols for final executable file:



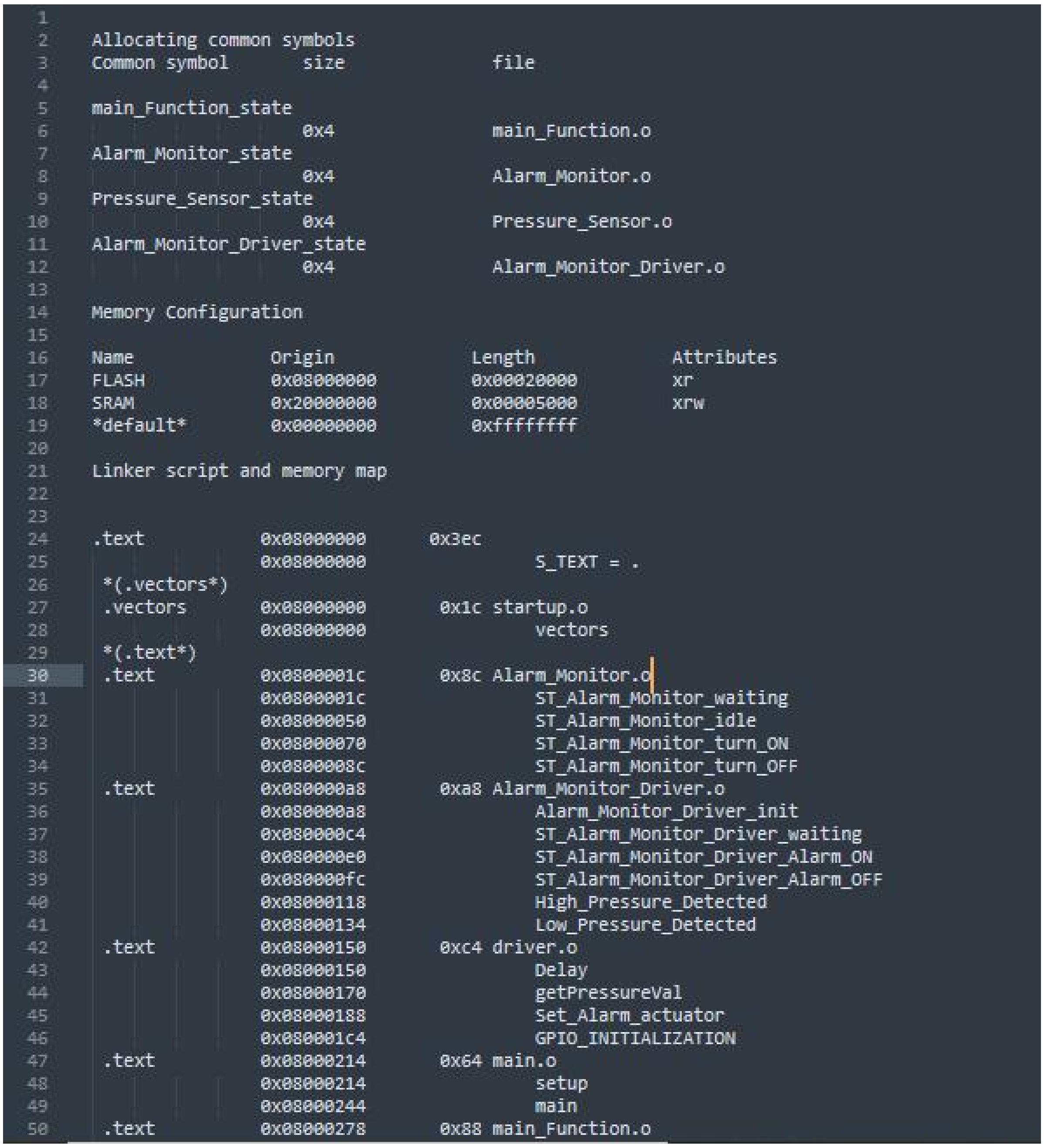
```
Dell@OsamaYoussef MINGW64 /e/Pressure_Controller
$ arm-none-eabi-nm.exe Pressure_Controller_Project.elf
20001010 B _STACK_TOP
080000a8 T Alarm_Monitor_Driver_init
08000480 D Alarm_Monitor_Driver_state
0800047c D Alarm_Monitor_state
0800036c W Bus_Fault
20000000 D current_state
0800036c T Default_Handler
08000150 T Delay
20000010 B E_BSS
20000010 D E_DATA
080003ec T E_TEXT
08000170 T getPressureVal
080001c4 T GPIO_INITIALIZATION
0800036c W H_Fault_Handler
08000118 T High_Pressure_Detected
08000134 T Low_Pressure_Detected
08000244 T main
20000004 D main_Function_Reading
08000484 D main_Function_state
0800036c W MM_Fault_Handler
0800036c W NMI_Handler
08000300 T Pressure_Sensor_init
2000000c D Pressure_Sensor_Reading
08000488 D Pressure_Sensor_state
0800036c W Reset_Handler
20000010 B S_BSS
20000000 D S_DATA
08000000 T S_TEXT
08000188 T Set_Alarm_actuator
08000278 T set_Pressure_Reading
08000214 T setup
080000fc T ST_Alarm_Monitor_Driver_Alarm_OFF
080000e0 T ST_Alarm_Monitor_Driver_Alarm_ON
080000c4 T ST_Alarm_Monitor_Driver_waiting
08000050 T ST_Alarm_Monitor_idle
0800008c T ST_Alarm_Monitor_turn_OFF
08000070 T ST_Alarm_Monitor_turn_ON
0800001c T ST_Alarm_Monitor_waiting
080002c0 T ST_main_Function_detected_reading
080002a4 T ST_main_Function_waiting
0800031c T ST_Pressure_Sensor_reading
0800034c T ST_Pressure_Sensor_waiting
20000008 D threshold
0800036c W Usage_Fault_Handler
08000000 T vectors
```

➢ **Readelf utility for final executable file :**

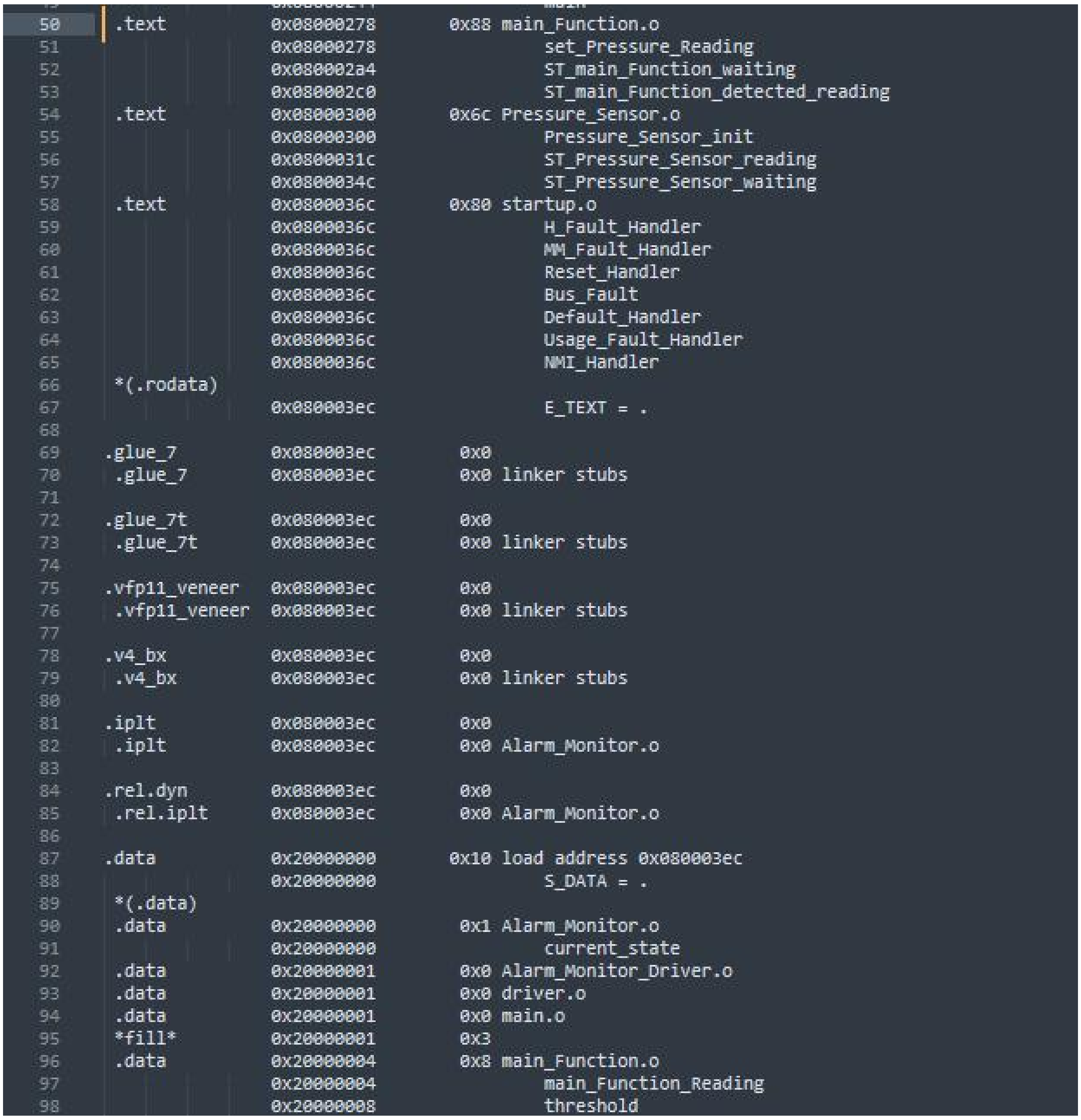


```
Dell@OsamaYoussef MINGW64 /e/Pressure_Controller
$ arm-none-eabi-readelf.exe -a Pressure_Controller_Project.elf
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           ARM
  Version:                           0x1
  Entry point address:               0x8000000
  Start of program headers:          52 (bytes into file)
  Start of section headers:          160664 (bytes into file)
  Flags:                             0x5000200, Version5 EABI, soft-float ABI
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         4
  Size of section headers:           40 (bytes)
  Number of section headers:         16
  Section header string table index: 15

Section Headers:
```

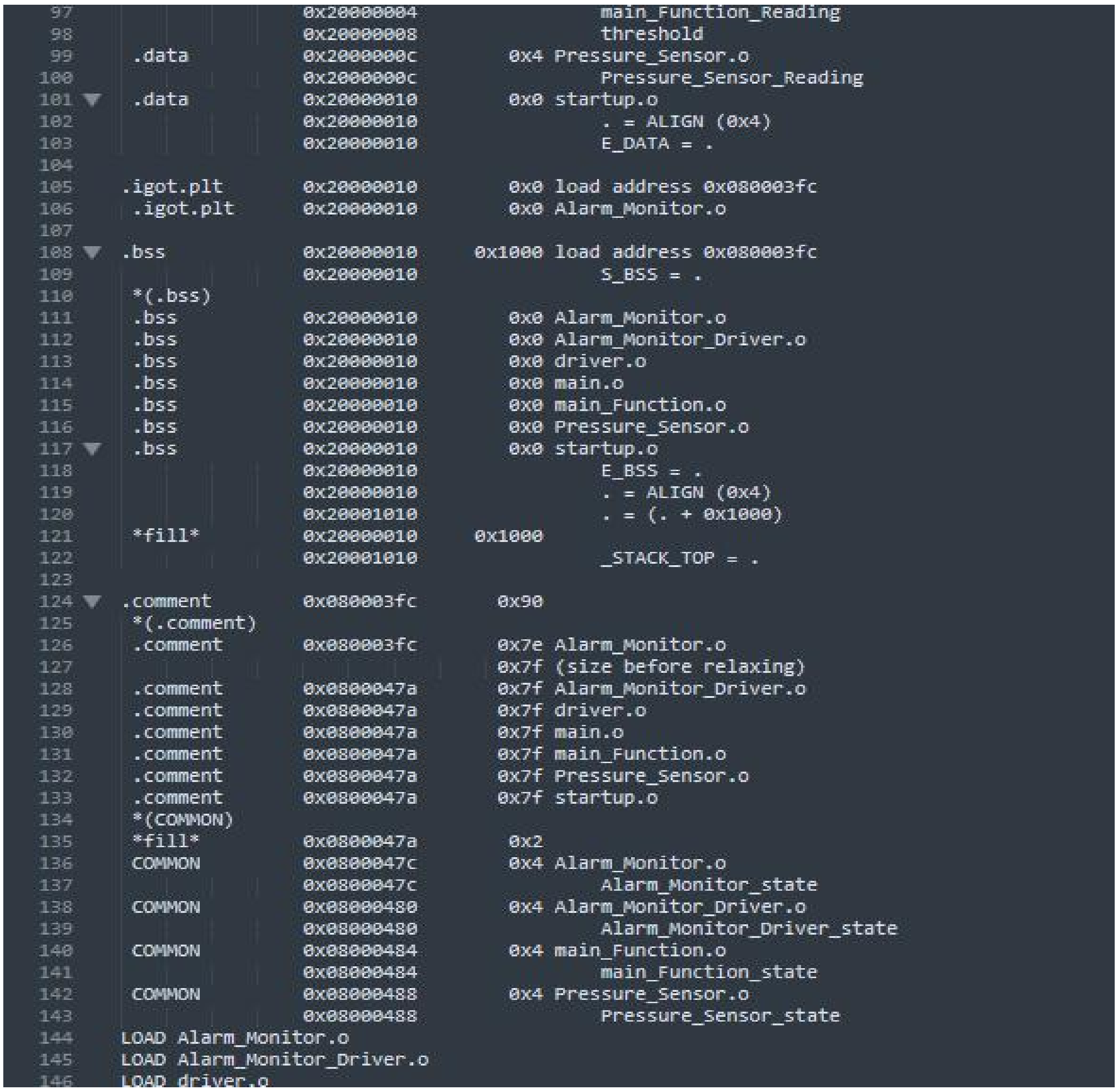
➢ The map file and symbols :



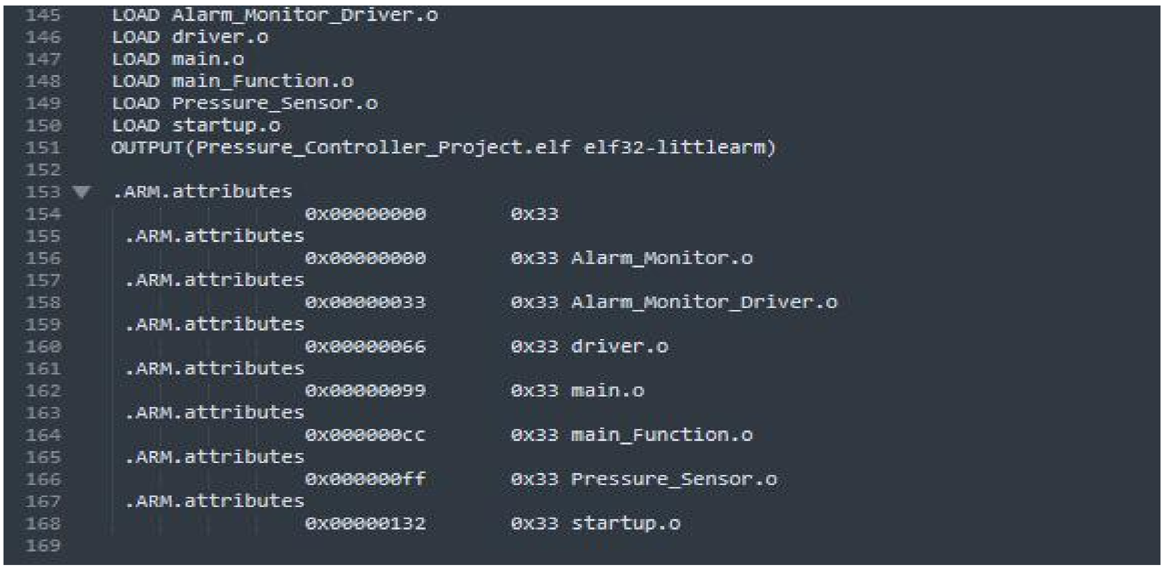
```

Allocating common symbols
Common symbol       size              file

main_Function_state
                    0x4               main_Function.o
Alarm_Monitor_state
                    0x4               Alarm_Monitor.o
Pressure_Sensor_state
                    0x4               Pressure_Sensor.o
Alarm_Monitor_Driver_state
                    0x4               Alarm_Monitor_Driver.o

Memory Configuration

Name                Origin            Length            Attributes
FLASH               0x08000000        0x00020000        xr
SRAM                0x20000000        0x00005000        xrw
*default*           0x00000000        0xffffffff

Linker script and memory map


.text              0x08000000      0x3ec
                   0x08000000              S_TEXT = .
 *(.vectors*)
 .vectors          0x08000000       0x1c startup.o
                   0x08000000              vectors
 *(.text*)
 .text             0x0800001c       0x8c Alarm_Monitor.o
                   0x0800001c              ST_Alarm_Monitor_waiting
                   0x08000050              ST_Alarm_Monitor_idle
                   0x08000070              ST_Alarm_Monitor_turn_ON
                   0x0800008c              ST_Alarm_Monitor_turn_OFF
 .text             0x080000a8       0xa8 Alarm_Monitor_Driver.o
                   0x080000a8              Alarm_Monitor_Driver_init
                   0x080000c4              ST_Alarm_Monitor_Driver_waiting
                   0x080000e0              ST_Alarm_Monitor_Driver_Alarm_ON
                   0x080000fc              ST_Alarm_Monitor_Driver_Alarm_OFF
                   0x08000118              High_Pressure_Detected
                   0x08000134              Low_Pressure_Detected
 .text             0x08000150       0xc4 driver.o
                   0x08000150              Delay
                   0x08000170              getPressureVal
                   0x08000188              Set_Alarm_actuator
                   0x080001c4              GPIO_INITIALIZATION
 .text             0x08000214       0x64 main.o
                   0x08000214              setup
                   0x08000244              main
 .text             0x08000278       0x88 main_Function.o
```



```
 .text          0x08000278      0x88 main_Function.o
                0x08000278               set_Pressure_Reading
                0x080002a4               ST_main_Function_waiting
                0x080002c0               ST_main_Function_detected_reading
 .text          0x08000300      0x6c Pressure_Sensor.o
                0x08000300               Pressure_Sensor_init
                0x0800031c               ST_Pressure_Sensor_reading
                0x0800034c               ST_Pressure_Sensor_waiting
 .text          0x0800036c      0x80 startup.o
                0x0800036c               H_Fault_Handler
                0x0800036c               MM_Fault_Handler
                0x0800036c               Reset_Handler
                0x0800036c               Bus_Fault
                0x0800036c               Default_Handler
                0x0800036c               Usage_Fault_Handler
                0x0800036c               NMI_Handler
 *(.rodata)
                0x080003ec               E_TEXT = .

.glue_7         0x080003ec       0x0
 .glue_7        0x080003ec       0x0 linker stubs

.glue_7t        0x080003ec       0x0
 .glue_7t       0x080003ec       0x0 linker stubs

.vfp11_veneer   0x080003ec       0x0
 .vfp11_veneer  0x080003ec       0x0 linker stubs

.v4_bx          0x080003ec       0x0
 .v4_bx         0x080003ec       0x0 linker stubs

.iplt           0x080003ec       0x0
 .iplt          0x080003ec       0x0 Alarm_Monitor.o

.rel.dyn        0x080003ec       0x0
 .rel.iplt      0x080003ec       0x0 Alarm_Monitor.o

.data           0x20000000      0x10 load address 0x080003ec
                0x20000000               S_DATA = .
 *(.data)
 .data          0x20000000       0x1 Alarm_Monitor.o
                0x20000000               current_state
 .data          0x20000001       0x0 Alarm_Monitor_Driver.o
 .data          0x20000001       0x0 driver.o
 .data          0x20000001       0x0 main.o
 *fill*         0x20000001       0x3
 .data          0x20000004       0x8 main_Function.o
                0x20000004               main_Function_Reading
                0x20000008               threshold
```

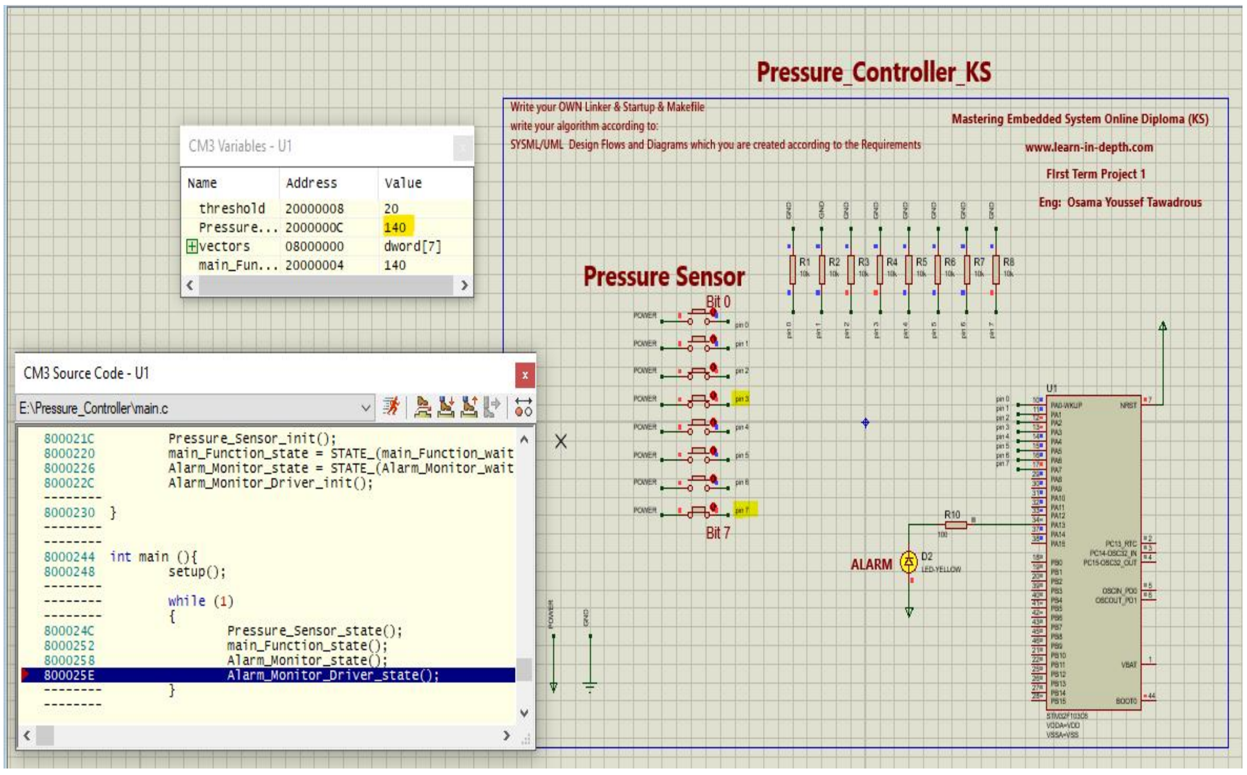


```
                     0x20000004                main_Function_Reading
                     0x20000008                threshold
 .data               0x2000000c        0x4 Pressure_Sensor.o
                     0x2000000c                Pressure_Sensor_Reading
 .data               0x20000010        0x0 startup.o
                     0x20000010                . = ALIGN (0x4)
                     0x20000010                E_DATA = .

.igot.plt            0x20000010        0x0 load address 0x080003fc
 .igot.plt           0x20000010        0x0 Alarm_Monitor.o

.bss                 0x20000010     0x1000 load address 0x080003fc
                     0x20000010                S_BSS = .
 *(.bss)
 .bss                0x20000010        0x0 Alarm_Monitor.o
 .bss                0x20000010        0x0 Alarm_Monitor_Driver.o
 .bss                0x20000010        0x0 driver.o
 .bss                0x20000010        0x0 main.o
 .bss                0x20000010        0x0 main_Function.o
 .bss                0x20000010        0x0 Pressure_Sensor.o
 .bss                0x20000010        0x0 startup.o
                     0x20000010                E_BSS = .
                     0x20000010                . = ALIGN (0x4)
                     0x20001010                . = (. + 0x1000)
 *fill*              0x20000010     0x1000
                     0x20001010                _STACK_TOP = .

.comment             0x080003fc       0x90
 *(.comment)
 .comment            0x080003fc       0x7e Alarm_Monitor.o
                                      0x7f (size before relaxing)
 .comment            0x0800047a       0x7f Alarm_Monitor_Driver.o
 .comment            0x0800047a       0x7f driver.o
 .comment            0x0800047a       0x7f main.o
 .comment            0x0800047a       0x7f main_Function.o
 .comment            0x0800047a       0x7f Pressure_Sensor.o
 .comment            0x0800047a       0x7f startup.o
 *(COMMON)
 *fill*              0x0800047a        0x2
 COMMON              0x0800047c        0x4 Alarm_Monitor.o
                     0x0800047c                Alarm_Monitor_state
 COMMON              0x08000480        0x4 Alarm_Monitor_Driver.o
                     0x08000480                Alarm_Monitor_Driver_state
 COMMON              0x08000484        0x4 main_Function.o
                     0x08000484                main_Function_state
 COMMON              0x08000488        0x4 Pressure_Sensor.o
                     0x08000488                Pressure_Sensor_state
LOAD Alarm_Monitor.o
LOAD Alarm_Monitor_Driver.o
LOAD driver.o
```



```
LOAD Alarm_Monitor_Driver.o
LOAD driver.o
LOAD main.o
LOAD main_Function.o
LOAD Pressure_Sensor.o
LOAD startup.o
OUTPUT(Pressure_Controller_Project.elf elf32-littlearm)

.ARM.attributes
                     0x00000000       0x33
 .ARM.attributes
                     0x00000000       0x33 Alarm_Monitor.o
 .ARM.attributes
                     0x00000033       0x33 Alarm_Monitor_Driver.o
 .ARM.attributes
                     0x00000066       0x33 driver.o
 .ARM.attributes
                     0x00000099       0x33 main.o
 .ARM.attributes
                     0x000000cc       0x33 main_Function.o
 .ARM.attributes
                     0x000000ff       0x33 Pressure_Sensor.o
 .ARM.attributes
                     0x00000132       0x33 startup.o
```

➢ **Simulation using Proteus :**


Pressure_Controller_KS

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: Osama Youssef Tawadrous

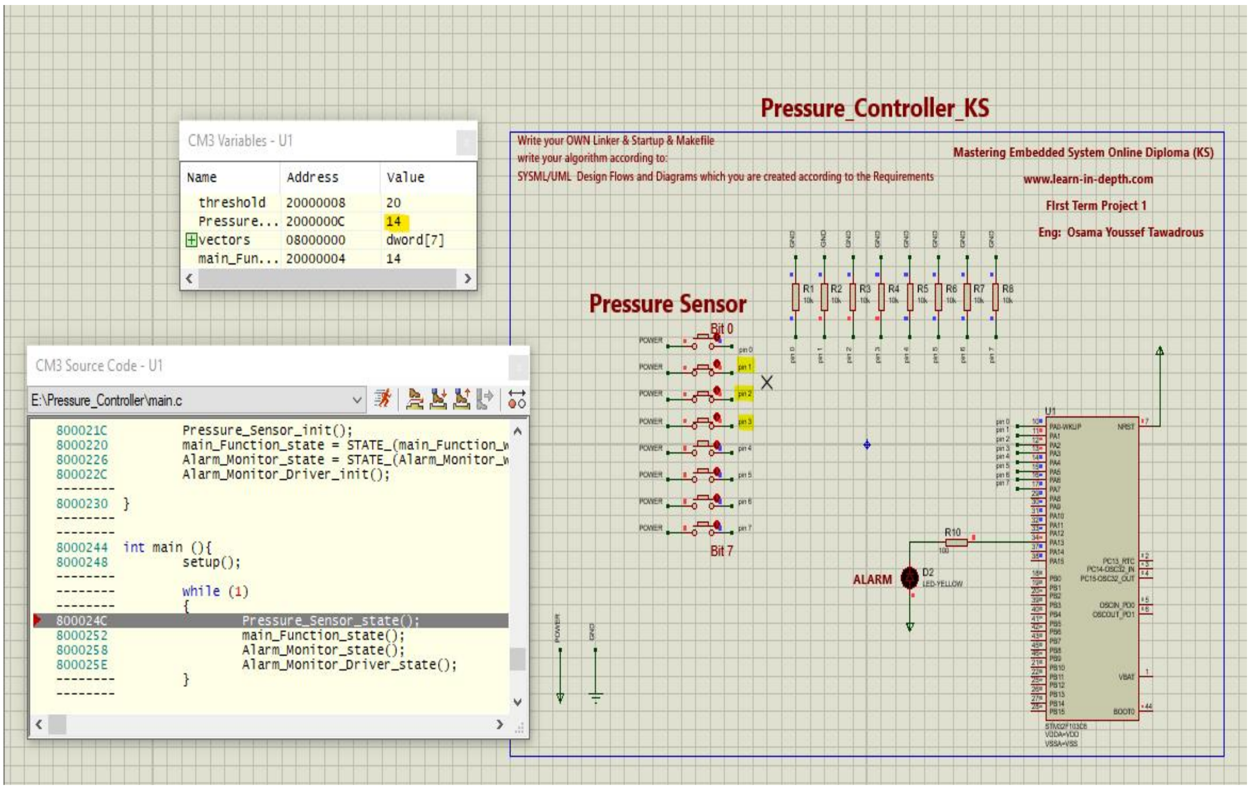
CM3 Variables - U1

Name	Address	Value
threshold	20000008	20
Pressure...	2000000C	140
⊞vectors	08000000	dword[7]
main_Fun...	20000004	140

CM3 Source Code - U1

E:\Pressure_Controller\main.c

```
800021C         Pressure_Sensor_init();
8000220         main_Function_state = STATE_(main_Function_wait
8000226         Alarm_Monitor_state = STATE_(Alarm_Monitor_wait
800022C         Alarm_Monitor_Driver_init();
--------
8000230  }
--------
--------
8000244  int main (){
8000248         setup();
--------
--------        while (1)
--------        {
800024C                 Pressure_Sensor_state();
8000252                 main_Function_state();
8000258                 Alarm_Monitor_state();
800025E                 Alarm_Monitor_Driver_state();
--------        }
--------
```


Pressure_Controller_KS

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: Osama Youssef Tawadrous

CM3 Variables - U1

Name	Address	Value
threshold	20000008	20
Pressure...	2000000C	14
⊞vectors	08000000	dword[7]
main_Fun...	20000004	14

CM3 Source Code - U1

E:\Pressure_Controller\main.c

```
800021C         Pressure_Sensor_init();
8000220         main_Function_state = STATE_(main_Function_w
8000226         Alarm_Monitor_state = STATE_(Alarm_Monitor_w
800022C         Alarm_Monitor_Driver_init();
--------
8000230  }
--------
--------
8000244  int main (){
8000248         setup();
--------
--------        while (1)
--------        {
800024C                 Pressure_Sensor_state();
8000252                 main_Function_state();
8000258                 Alarm_Monitor_state();
800025E                 Alarm_Monitor_Driver_state();
--------        }
--------
```