Mastering Embedded System Online Diploma

www.learn-in-depth.com

First Term (Final Project 1)

Eng. Ebram Edward Fouad Habib

My Profile:

https://www.learn-in-depth.com/online-diploma/ebramedward7@gmail.com

List of Contents:

Pressure Controller

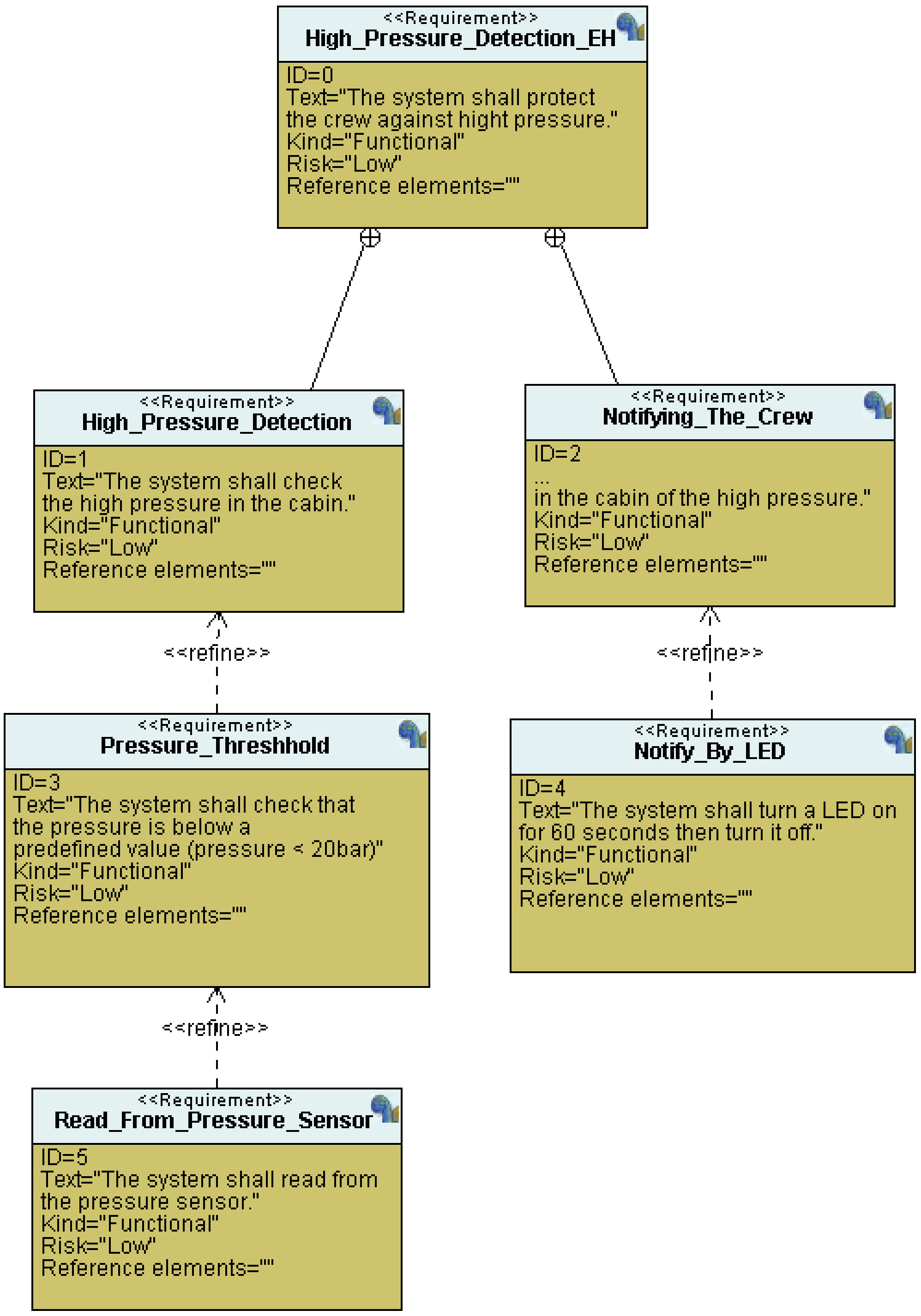
Project Description:

- **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.**

Assumptions:

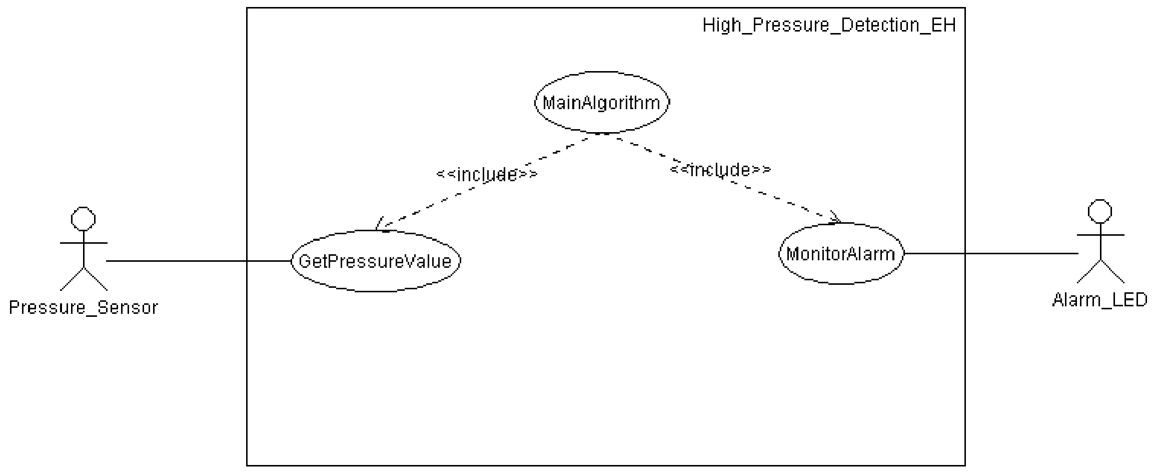
- The controller set up and shutdown procedures are not modeled
- The controller maintenance is not modeled
- The pressure sensor never fails
- The alarm never fails
- The controller never faces power cut

Requirements Diagram:

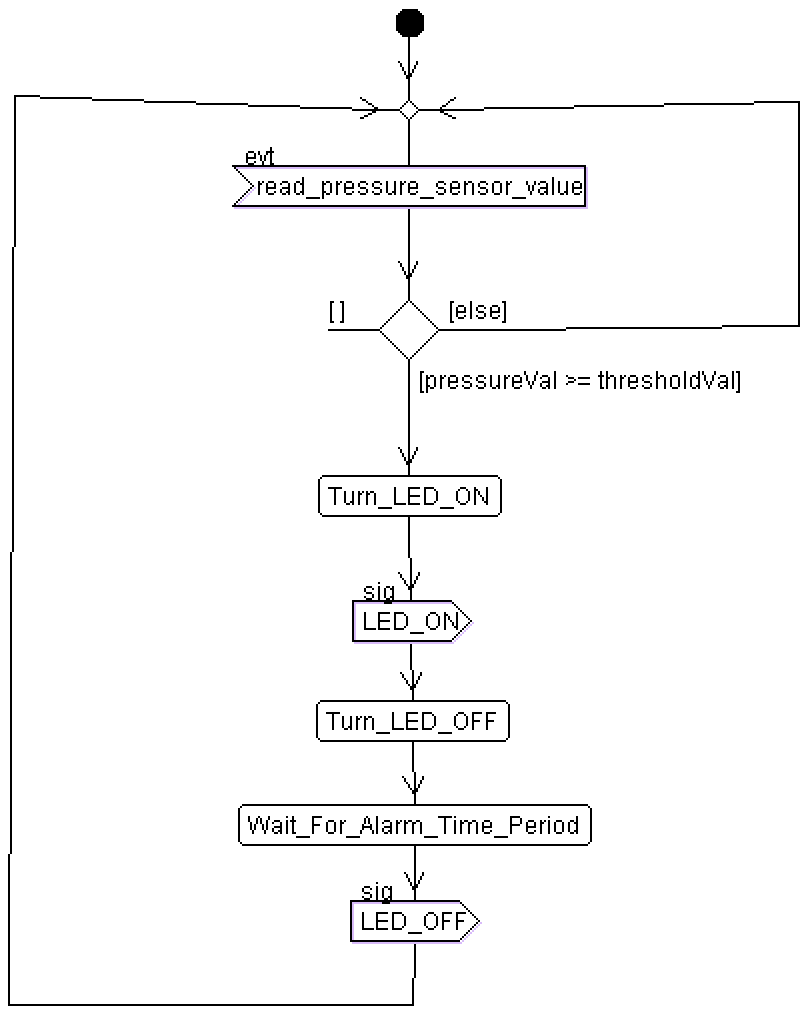


System Analysis:

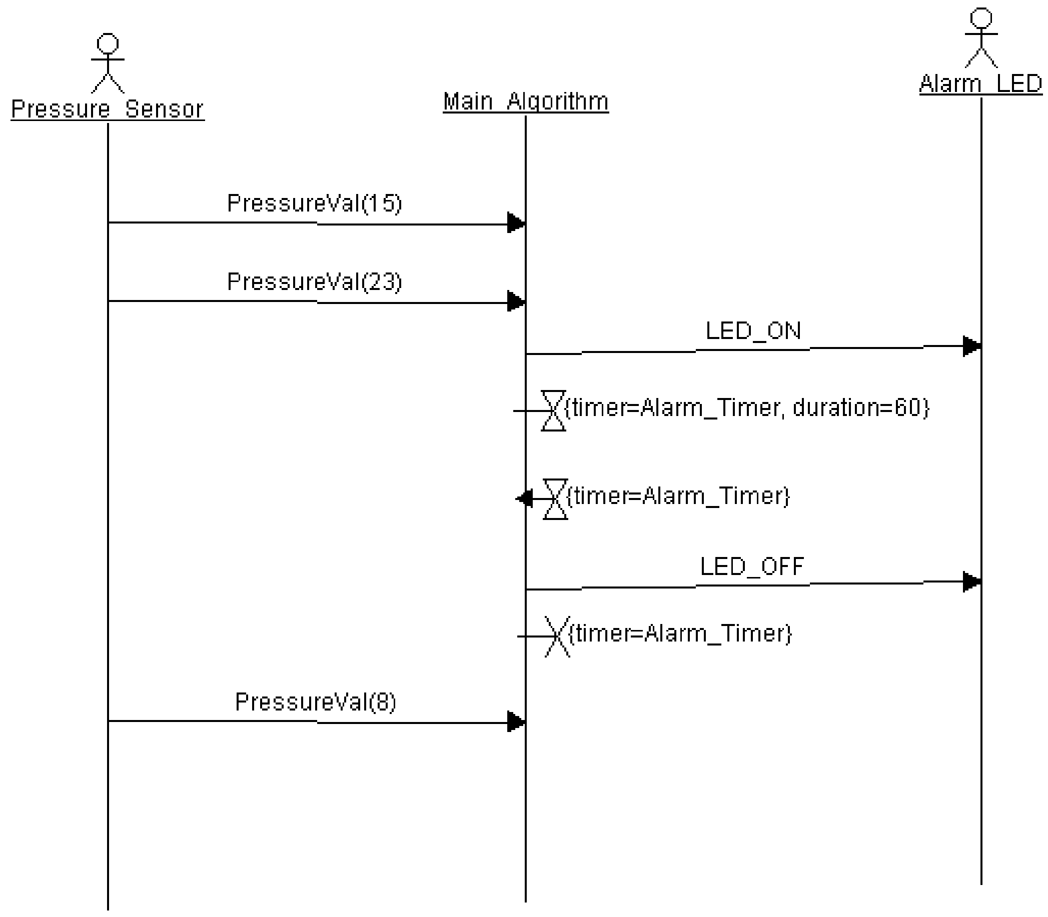
1. Use Case Diagram



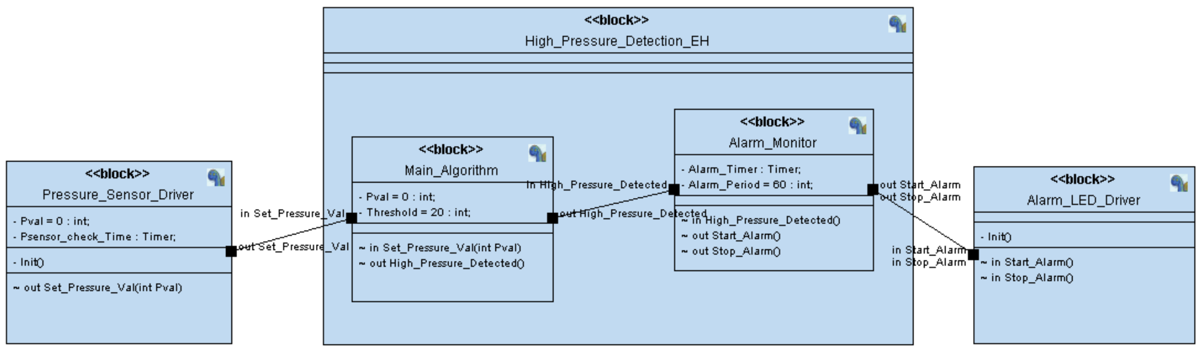
2. Activity Diagram


evt
read_pressure_sensor_value
[ ]
[else]
[pressureVal >= thresholdVal]
Turn_LED_ON
sig
LED_ON
Turn_LED_OFF
Wait_For_Alarm_Time_Period
sig
LED_OFF

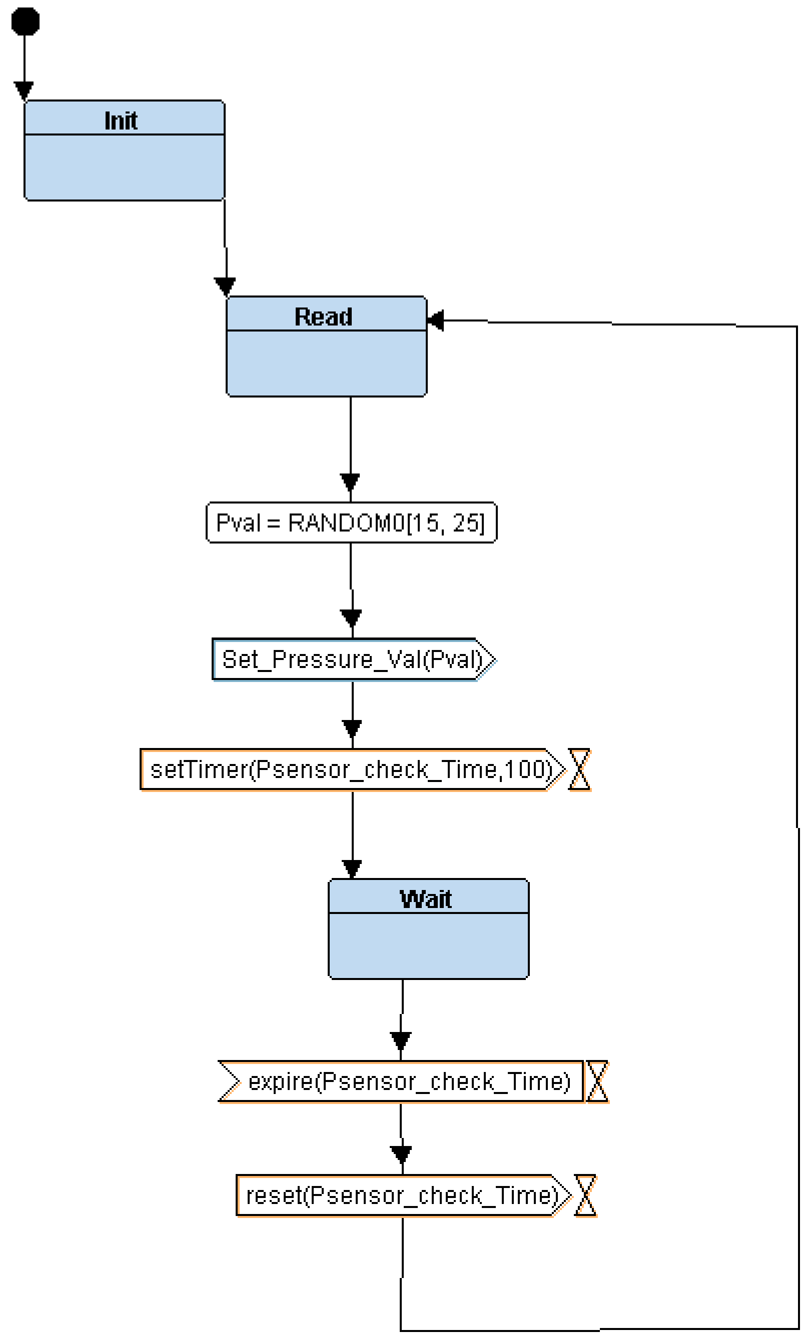
3. Sequence Diagram

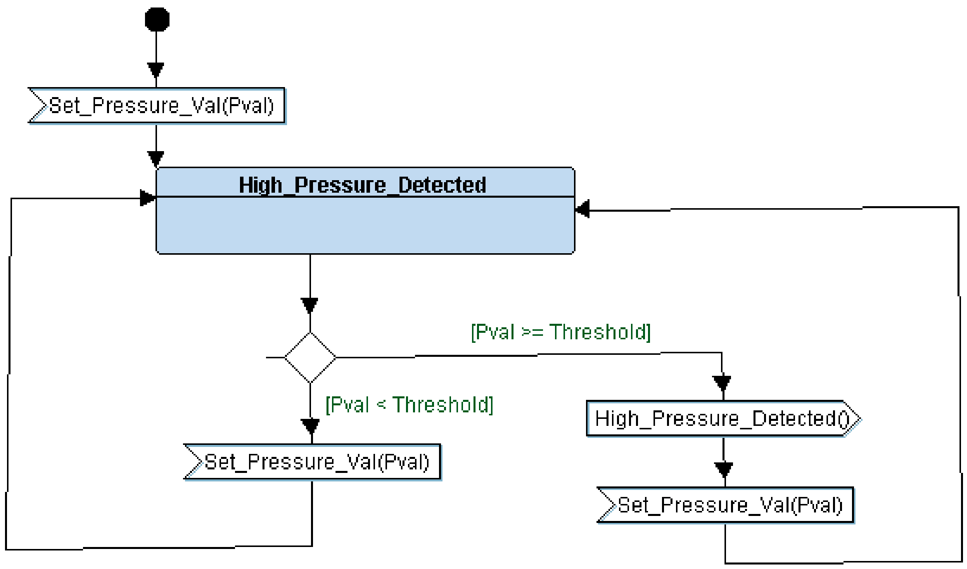


System Design (Modules with its own state machines)

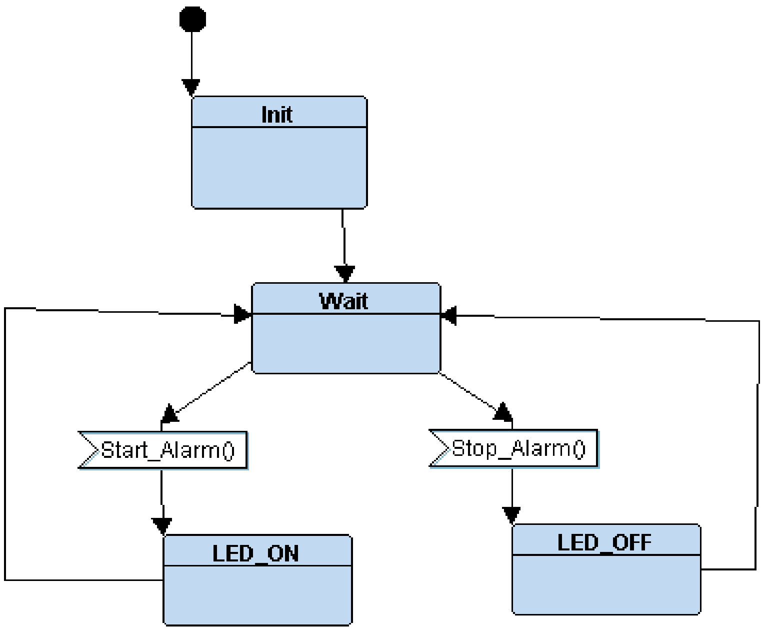


1-Pressure Sensor State Diagram:

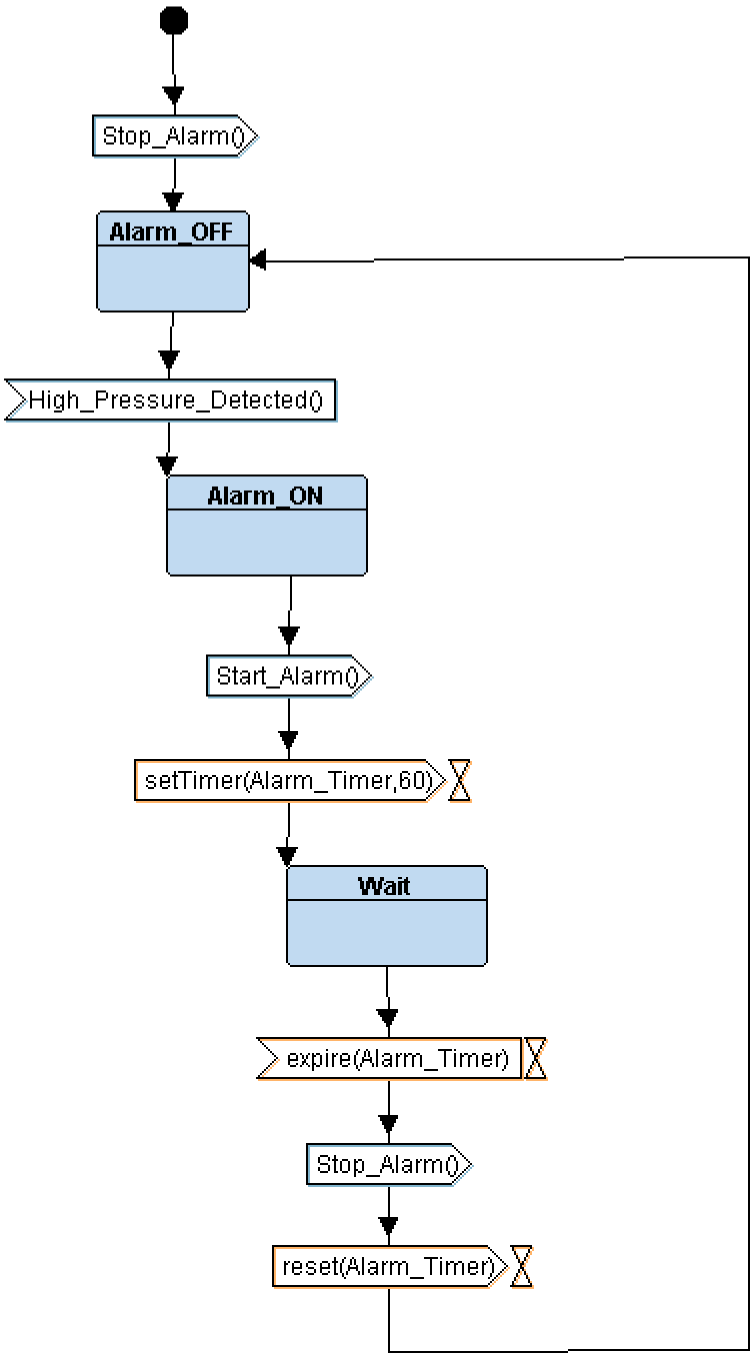

Init
Read
Pval = RANDOM0[15, 25]
Set_Pressure_Val(Pval)
setTimer(Psensor_check_Time,100)
Wait
expire(Psensor_check_Time)
reset(Psensor_check_Time)



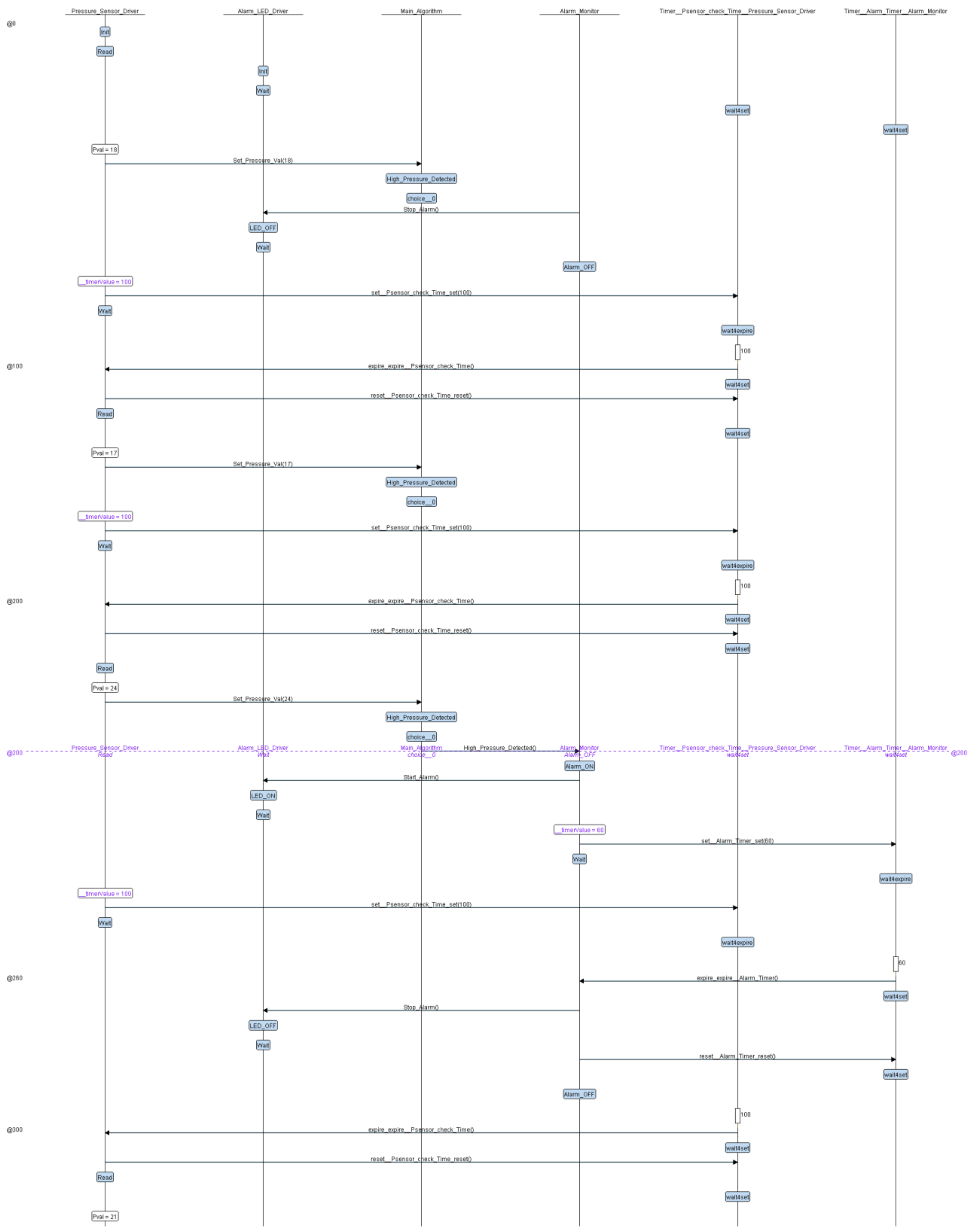
3-Alarm LED State Diagram:



4- Alarm Monitor State Diagram:

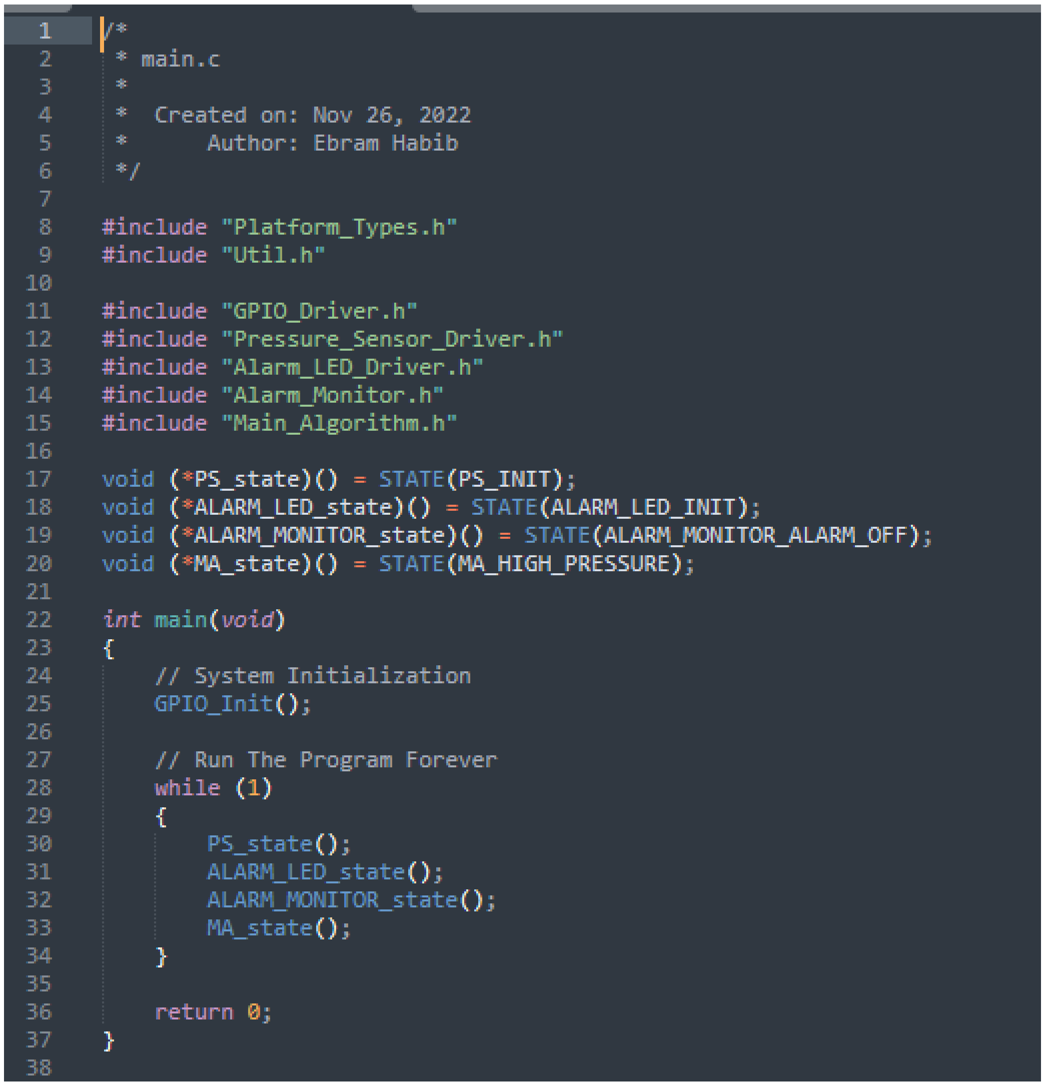


- System Design Simulation:



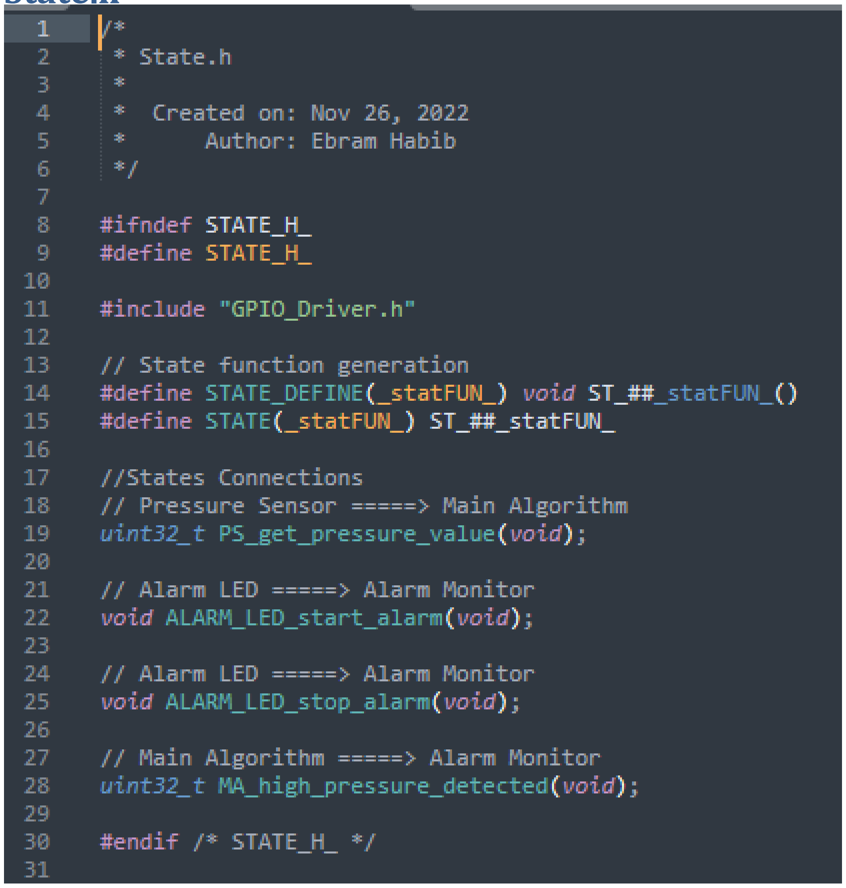
Implementation of each module in C:
(An Image for each file.c & file.h with the Corresponding state machine)

main.c



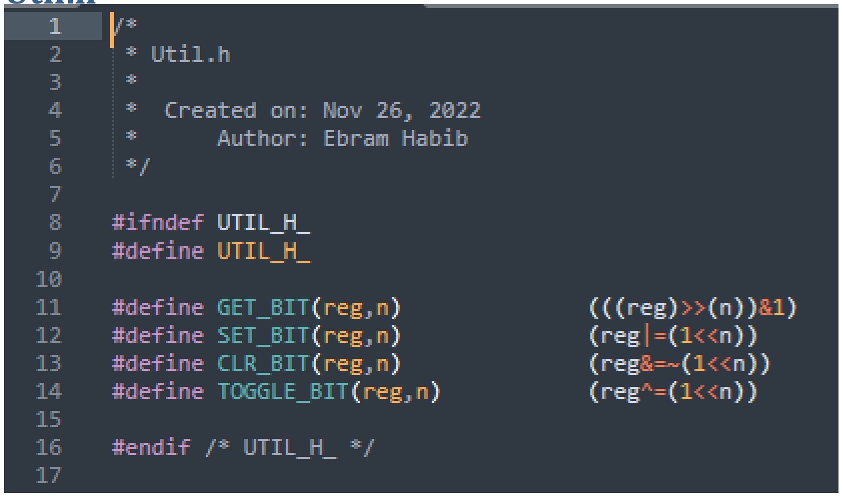
```c
/*
 * main.c
 *
 *  Created on: Nov 26, 2022
 *      Author: Ebram Habib
 */

#include "Platform_Types.h"
#include "Util.h"

#include "GPIO_Driver.h"
#include "Pressure_Sensor_Driver.h"
#include "Alarm_LED_Driver.h"
#include "Alarm_Monitor.h"
#include "Main_Algorithm.h"

void (*PS_state)() = STATE(PS_INIT);
void (*ALARM_LED_state)() = STATE(ALARM_LED_INIT);
void (*ALARM_MONITOR_state)() = STATE(ALARM_MONITOR_ALARM_OFF);
void (*MA_state)() = STATE(MA_HIGH_PRESSURE);

int main(void)
{
    // System Initialization
    GPIO_Init();

    // Run The Program Forever
    while (1)
    {
        PS_state();
        ALARM_LED_state();
        ALARM_MONITOR_state();
        MA_state();
    }

    return 0;
}
```

State.h



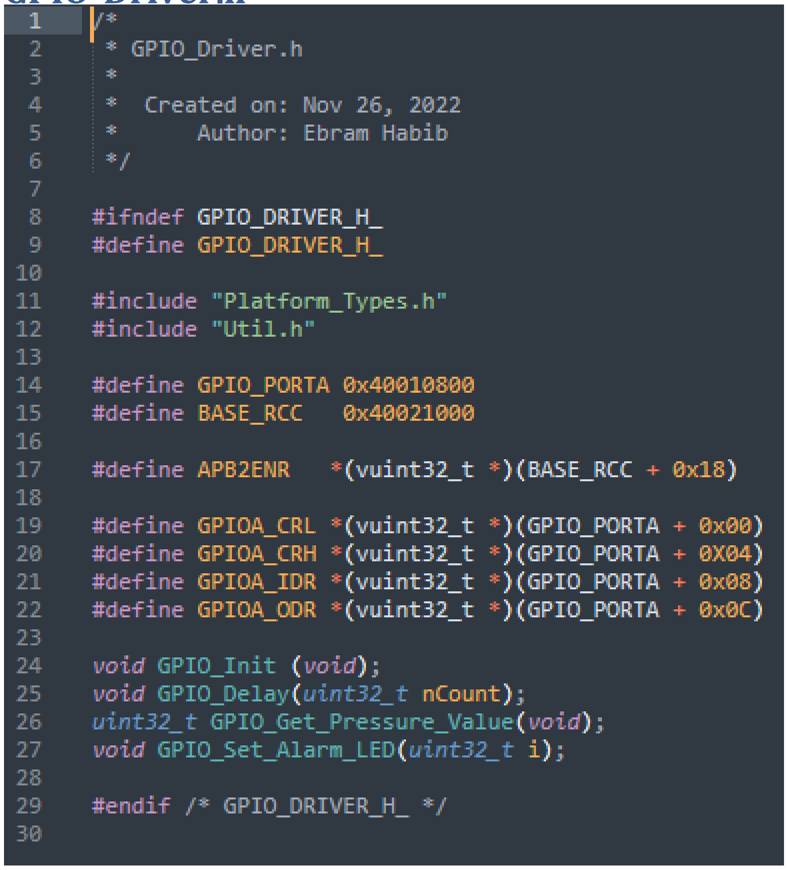
```c
/*
 * State.h
 *
 *  Created on: Nov 26, 2022
 *      Author: Ebram Habib
 */

#ifndef STATE_H_
#define STATE_H_

#include "GPIO_Driver.h"

// State function generation
#define STATE_DEFINE(_statFUN_) void ST_##_statFUN_()
#define STATE(_statFUN_) ST_##_statFUN_

//States Connections
// Pressure Sensor =====> Main Algorithm
uint32_t PS_get_pressure_value(void);

// Alarm LED =====> Alarm Monitor
void ALARM_LED_start_alarm(void);

// Alarm LED =====> Alarm Monitor
void ALARM_LED_stop_alarm(void);

// Main Algorithm =====> Alarm Monitor
uint32_t MA_high_pressure_detected(void);

#endif /* STATE_H_ */
```

Util.h



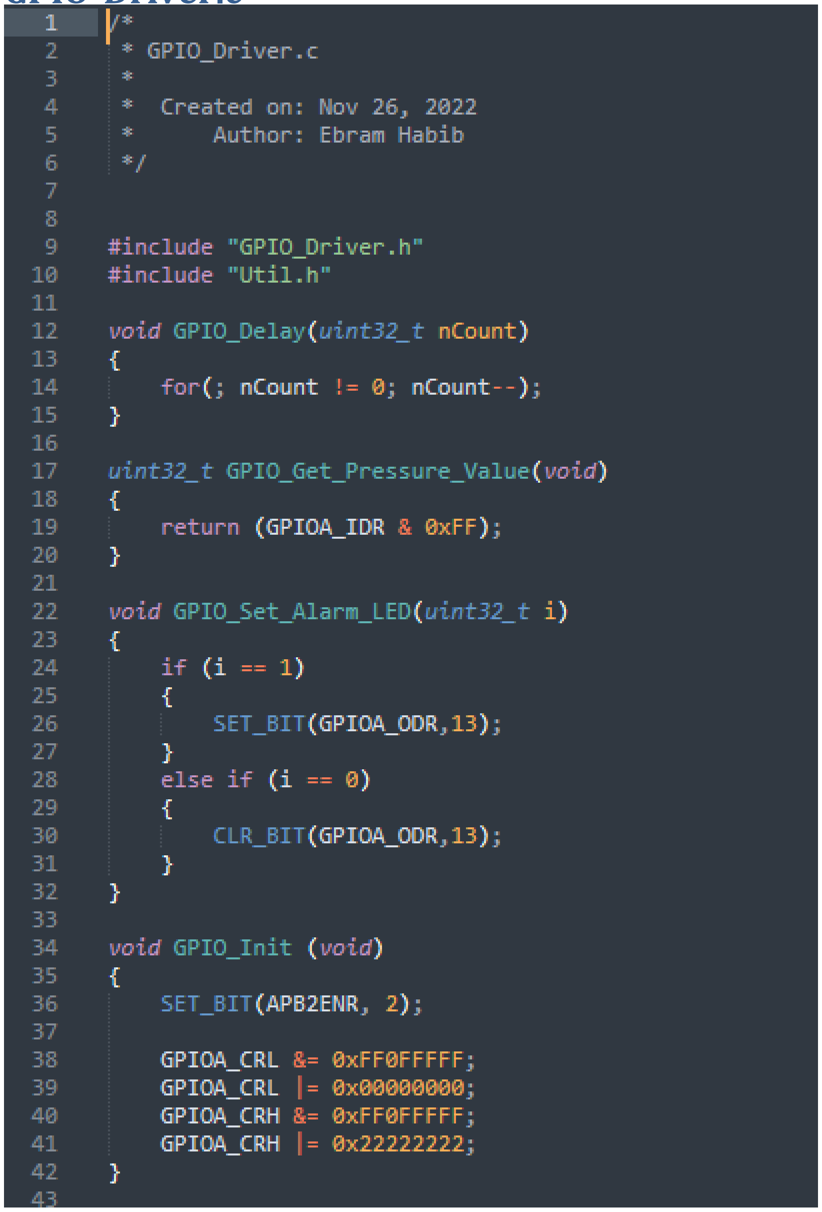
```c
/*
 * Util.h
 *
 *  Created on: Nov 26, 2022
 *      Author: Ebram Habib
 */

#ifndef UTIL_H_
#define UTIL_H_

#define GET_BIT(reg,n)              (((reg)>>(n))&1)
#define SET_BIT(reg,n)              (reg|=(1<<n))
#define CLR_BIT(reg,n)              (reg&=~(1<<n))
#define TOGGLE_BIT(reg,n)           (reg^=(1<<n))

#endif /* UTIL_H_ */
```

GPIO_Driver.h



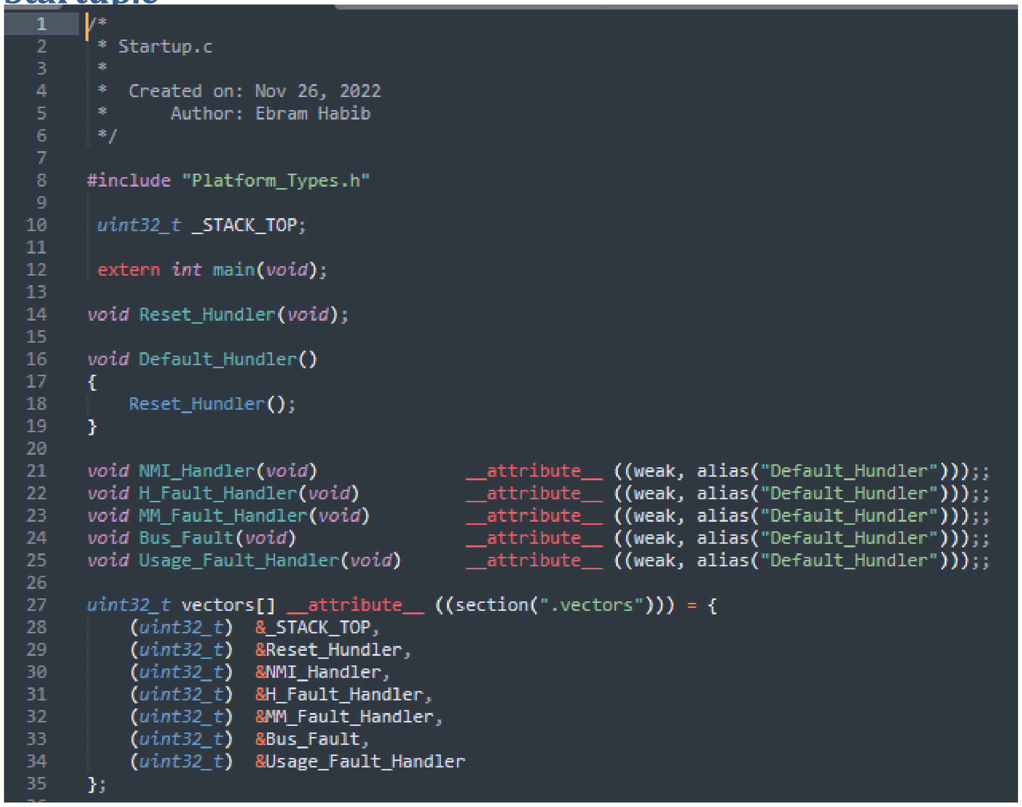
```c
/*
 * GPIO_Driver.h
 *
 *  Created on: Nov 26, 2022
 *      Author: Ebram Habib
 */

#ifndef GPIO_DRIVER_H_
#define GPIO_DRIVER_H_

#include "Platform_Types.h"
#include "Util.h"

#define GPIO_PORTA 0x40010800
#define BASE_RCC   0x40021000

#define APB2ENR   *(vuint32_t *)(BASE_RCC + 0x18)

#define GPIOA_CRL *(vuint32_t *)(GPIO_PORTA + 0x00)
#define GPIOA_CRH *(vuint32_t *)(GPIO_PORTA + 0X04)
#define GPIOA_IDR *(vuint32_t *)(GPIO_PORTA + 0x08)
#define GPIOA_ODR *(vuint32_t *)(GPIO_PORTA + 0x0C)

void GPIO_Init (void);
void GPIO_Delay(uint32_t nCount);
uint32_t GPIO_Get_Pressure_Value(void);
void GPIO_Set_Alarm_LED(uint32_t i);

#endif /* GPIO_DRIVER_H_ */

```

GPIO_Driver.c

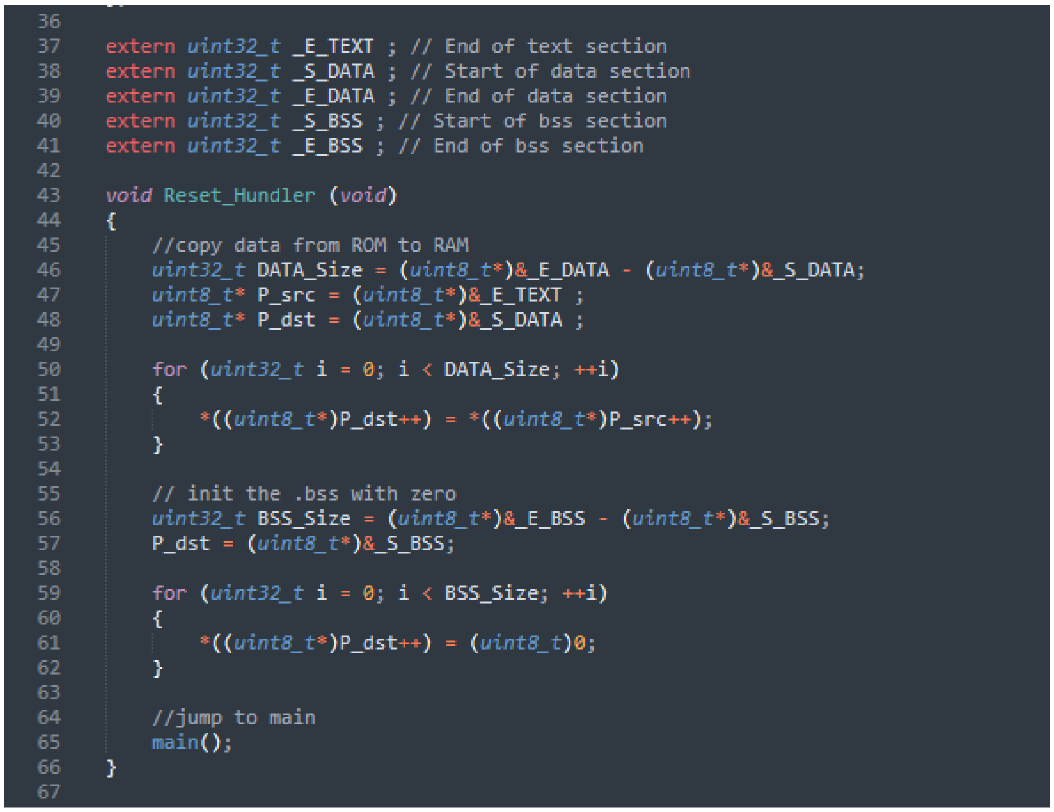


```c
/*
 * GPIO_Driver.c
 *
 *  Created on: Nov 26, 2022
 *      Author: Ebram Habib
 */


#include "GPIO_Driver.h"
#include "Util.h"

void GPIO_Delay(uint32_t nCount)
{
    for(; nCount != 0; nCount--);
}

uint32_t GPIO_Get_Pressure_Value(void)
{
    return (GPIOA_IDR & 0xFF);
}

void GPIO_Set_Alarm_LED(uint32_t i)
{
    if (i == 1)
    {
        SET_BIT(GPIOA_ODR,13);
    }
    else if (i == 0)
    {
        CLR_BIT(GPIOA_ODR,13);
    }
}

void GPIO_Init (void)
{
    SET_BIT(APB2ENR, 2);

    GPIOA_CRL &= 0xFF0FFFFF;
    GPIOA_CRL |= 0x00000000;
    GPIOA_CRH &= 0xFF0FFFFF;
    GPIOA_CRH |= 0x22222222;
}

```

Startup.c

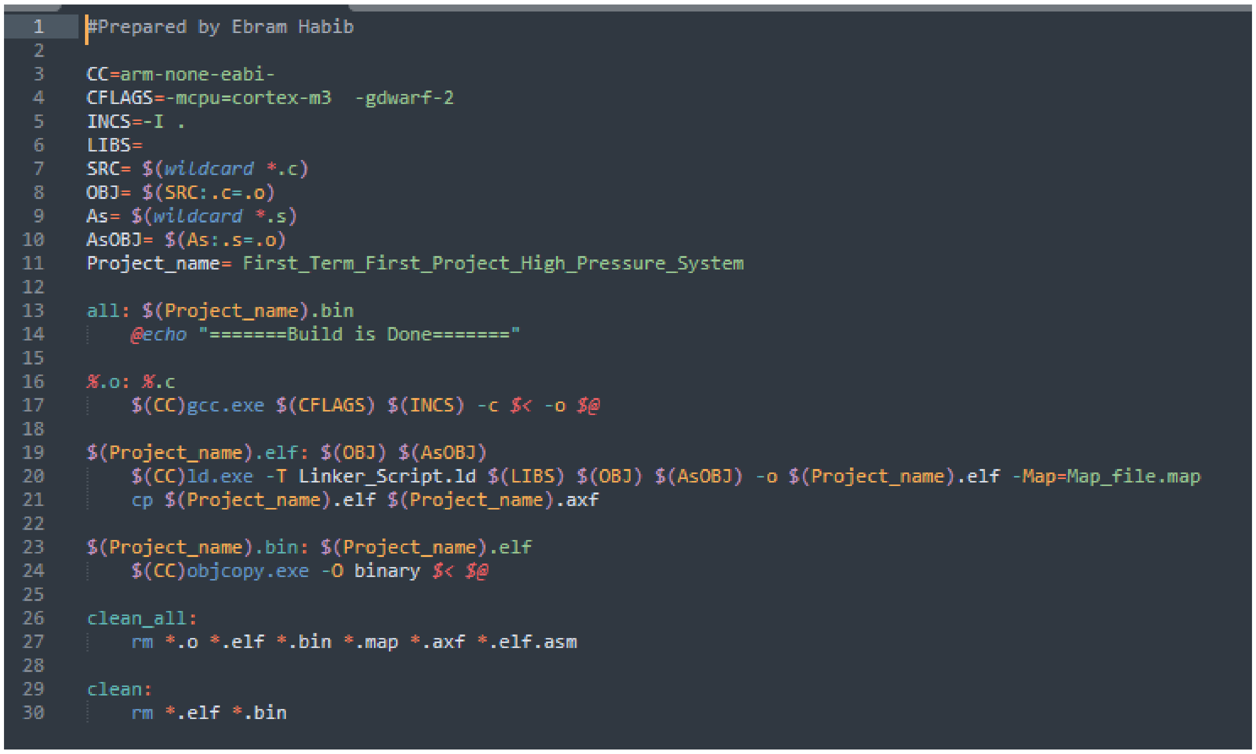


```c
/*
 * Startup.c
 *
 *  Created on: Nov 26, 2022
 *      Author: Ebram Habib
 */

#include "Platform_Types.h"

 uint32_t _STACK_TOP;

 extern int main(void);

void Reset_Hundler(void);

void Default_Hundler()
{
    Reset_Hundler();
}

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

uint32_t vectors[] __attribute__ ((section(".vectors"))) = {
    (uint32_t)  &_STACK_TOP,
    (uint32_t)  &Reset_Hundler,
    (uint32_t)  &NMI_Handler,
    (uint32_t)  &H_Fault_Handler,
    (uint32_t)  &MM_Fault_Handler,
    (uint32_t)  &Bus_Fault,
    (uint32_t)  &Usage_Fault_Handler
};
```



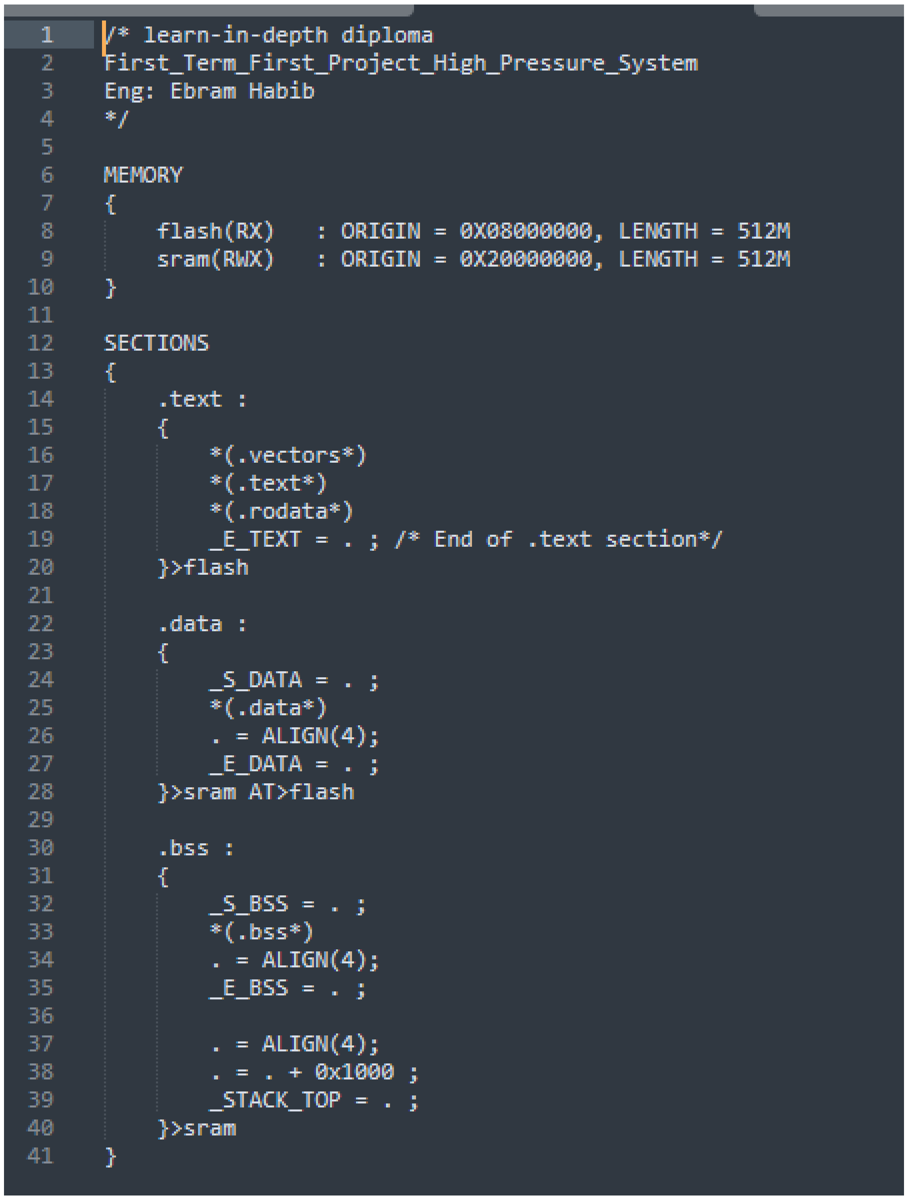
```c

extern uint32_t _E_TEXT ; // End of text section
extern uint32_t _S_DATA ; // Start of data section
extern uint32_t _E_DATA ; // End of data section
extern uint32_t _S_BSS ; // Start of bss section
extern uint32_t _E_BSS ; // End of bss section

void Reset_Hundler (void)
{
    //copy data from ROM to RAM
    uint32_t DATA_Size = (uint8_t*)&_E_DATA - (uint8_t*)&_S_DATA;
    uint8_t* P_src = (uint8_t*)&_E_TEXT ;
    uint8_t* P_dst = (uint8_t*)&_S_DATA ;

    for (uint32_t i = 0; i < DATA_Size; ++i)
    {
        *((uint8_t*)P_dst++) = *((uint8_t*)P_src++);
    }

    // init the .bss with zero
    uint32_t BSS_Size = (uint8_t*)&_E_BSS - (uint8_t*)&_S_BSS;
    P_dst = (uint8_t*)&_S_BSS;

    for (uint32_t i = 0; i < BSS_Size; ++i)
    {
        *((uint8_t*)P_dst++) = (uint8_t)0;
    }

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

MakeFile



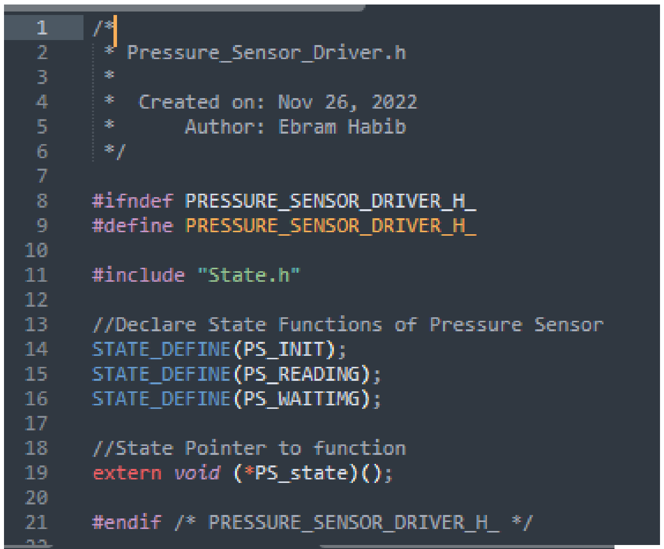
```
#Prepared by Ebram Habib

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_First_Project_High_Pressure_System

all: $(Project_name).bin
	@echo "=======Build is Done======="

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

$(Project_name).elf: $(OBJ) $(AsOBJ)
	$(CC)ld.exe -T Linker_Script.ld $(LIBS) $(OBJ) $(AsOBJ) -o $(Project_name).elf -Map=Map_file.map
	cp $(Project_name).elf $(Project_name).axf

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

clean_all:
	rm *.o *.elf *.bin *.map *.axf *.elf.asm

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

Linker_Script.ld

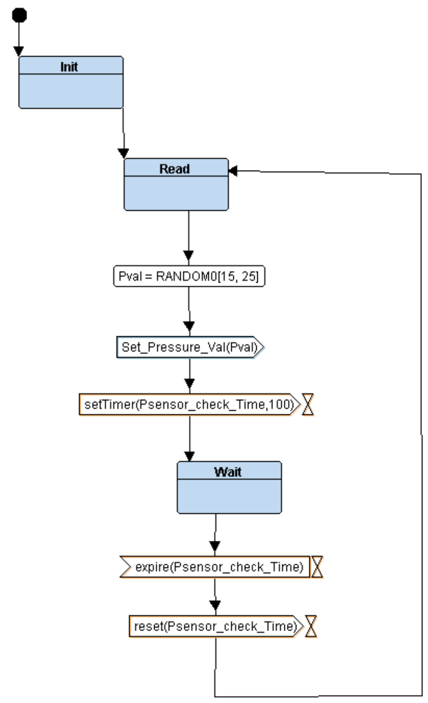


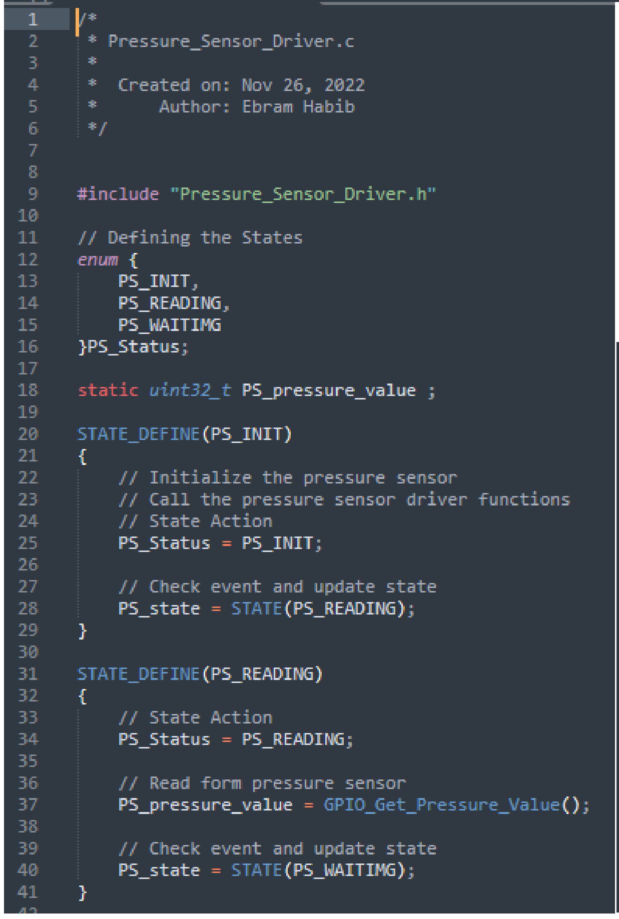
```
/* learn-in-depth diploma
First_Term_First_Project_High_Pressure_System
Eng: Ebram Habib
*/

MEMORY
{
    flash(RX)   : ORIGIN = 0X08000000, LENGTH = 512M
    sram(RWX)   : ORIGIN = 0X20000000, LENGTH = 512M
}

SECTIONS
{
    .text :
    {
        *(.vectors*)
        *(.text*)
        *(.rodata*)
        _E_TEXT = . ; /* End of .text section*/
    }>flash

    .data :
    {
        _S_DATA = . ;
        *(.data*)
        . = ALIGN(4);
        _E_DATA = . ;
    }>sram AT>flash

    .bss :
    {
        _S_BSS = . ;
        *(.bss*)
        . = ALIGN(4);
        _E_BSS = . ;

        . = ALIGN(4);
        . = . + 0x1000 ;
        _STACK_TOP = . ;
    }>sram
}
```

Pressure_Sensor_Driver

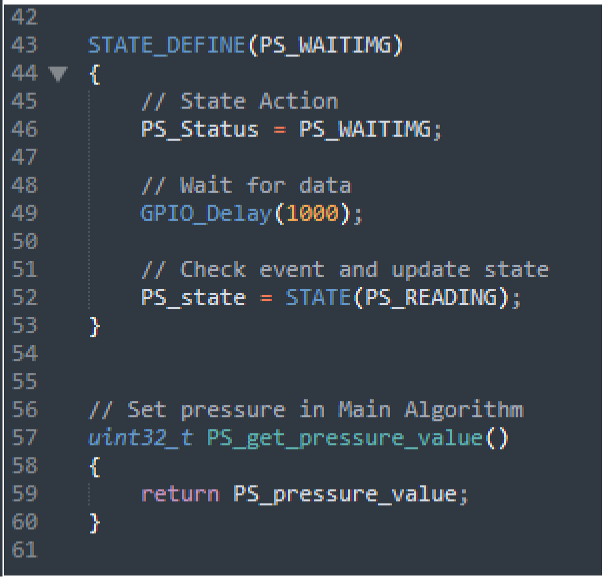


```c
/*
 * Pressure_Sensor_Driver.h
 *
 *  Created on: Nov 26, 2022
 *      Author: Ebram Habib
 */

#ifndef PRESSURE_SENSOR_DRIVER_H_
#define PRESSURE_SENSOR_DRIVER_H_

#include "State.h"

//Declare State Functions of Pressure Sensor
STATE_DEFINE(PS_INIT);
STATE_DEFINE(PS_READING);
STATE_DEFINE(PS_WAITIMG);

//State Pointer to function
extern void (*PS_state)();

#endif /* PRESSURE_SENSOR_DRIVER_H_ */
```


Init
Read
Pval = RANDOM0[15, 25]
Set_Pressure_Val(Pval)
setTimer(Psensor_check_Time,100)
Wait
expire(Psensor_check_Time)
reset(Psensor_check_Time)

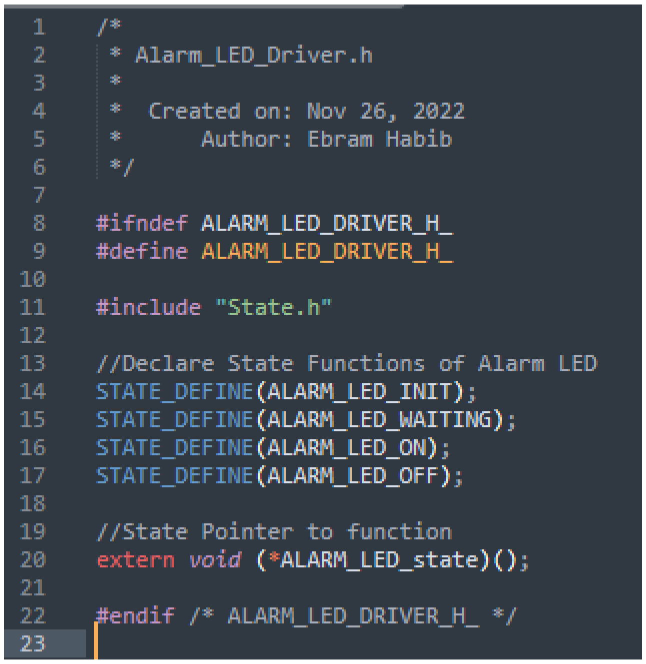


```c
/*
 * Pressure_Sensor_Driver.c
 *
 *  Created on: Nov 26, 2022
 *      Author: Ebram Habib
 */


#include "Pressure_Sensor_Driver.h"

// Defining the States
enum {
    PS_INIT,
    PS_READING,
    PS_WAITIMG
}PS_Status;

static uint32_t PS_pressure_value ;

STATE_DEFINE(PS_INIT)
{
    // Initialize the pressure sensor
    // Call the pressure sensor driver functions
    // State Action
    PS_Status = PS_INIT;

    // Check event and update state
    PS_state = STATE(PS_READING);
}

STATE_DEFINE(PS_READING)
{
    // State Action
    PS_Status = PS_READING;

    // Read form pressure sensor
    PS_pressure_value = GPIO_Get_Pressure_Value();

    // Check event and update state
    PS_state = STATE(PS_WAITIMG);
}
```

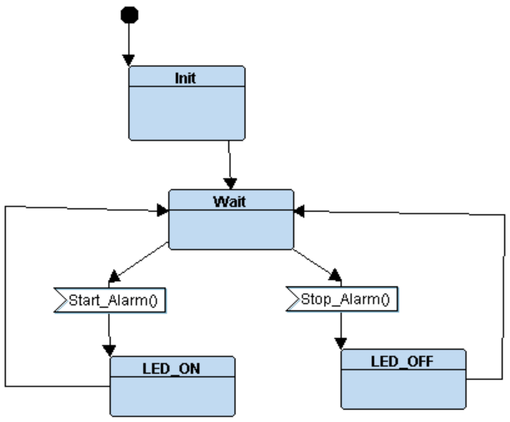


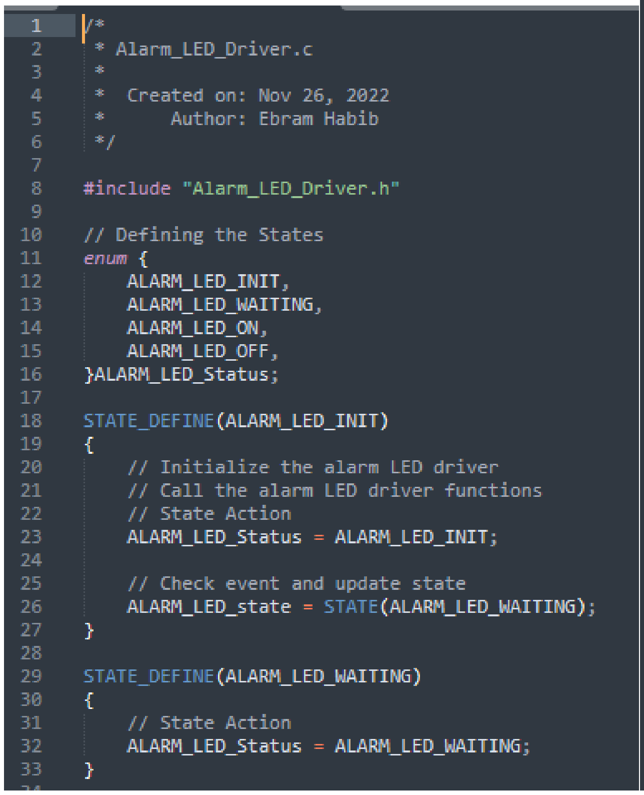
```c

STATE_DEFINE(PS_WAITIMG)
{
    // State Action
    PS_Status = PS_WAITIMG;

    // Wait for data
    GPIO_Delay(1000);

    // Check event and update state
    PS_state = STATE(PS_READING);
}


// Set pressure in Main Algorithm
uint32_t PS_get_pressure_value()
{
    return PS_pressure_value;
}

```

Alarm_LED_Driver

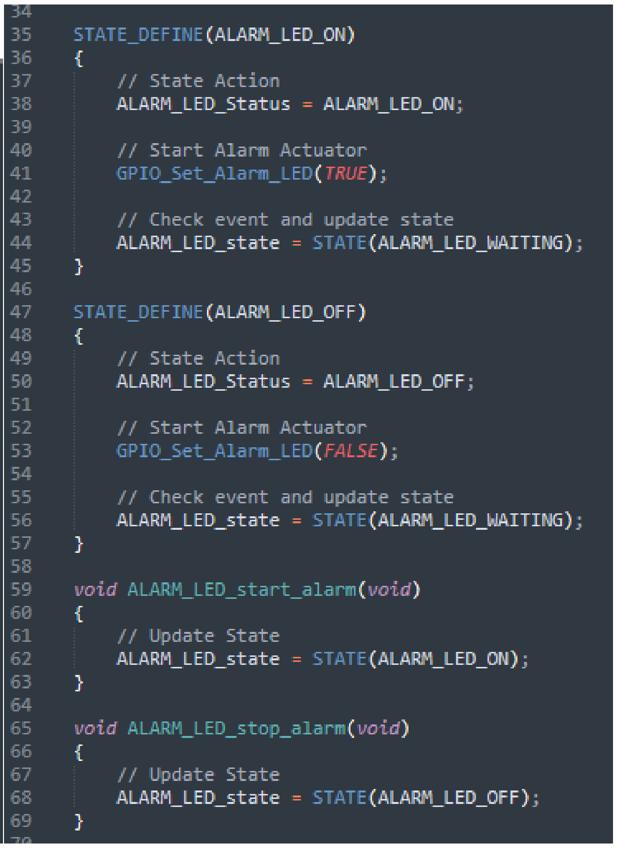


```c
/*
 * Alarm_LED_Driver.h
 *
 *  Created on: Nov 26, 2022
 *      Author: Ebram Habib
 */

#ifndef ALARM_LED_DRIVER_H_
#define ALARM_LED_DRIVER_H_

#include "State.h"

//Declare State Functions of Alarm LED
STATE_DEFINE(ALARM_LED_INIT);
STATE_DEFINE(ALARM_LED_WAITING);
STATE_DEFINE(ALARM_LED_ON);
STATE_DEFINE(ALARM_LED_OFF);

//State Pointer to function
extern void (*ALARM_LED_state)();

#endif /* ALARM_LED_DRIVER_H_ */
```



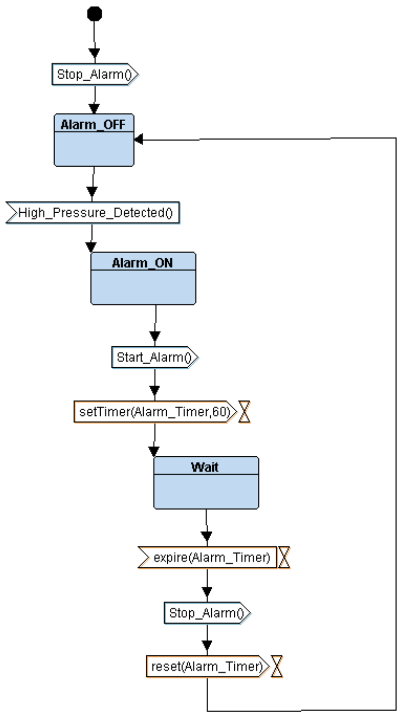


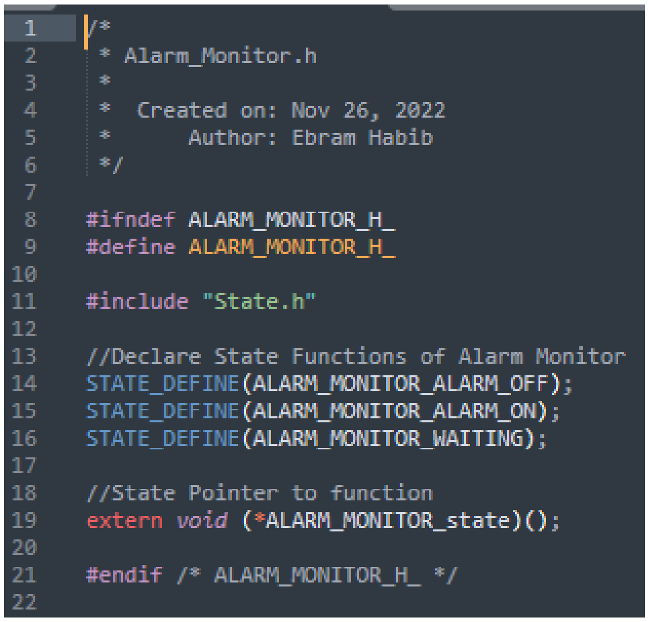
```c
/*
 * Alarm_LED_Driver.c
 *
 *  Created on: Nov 26, 2022
 *      Author: Ebram Habib
 */

#include "Alarm_LED_Driver.h"

// Defining the States
enum {
    ALARM_LED_INIT,
    ALARM_LED_WAITING,
    ALARM_LED_ON,
    ALARM_LED_OFF,
}ALARM_LED_Status;

STATE_DEFINE(ALARM_LED_INIT)
{
    // Initialize the alarm LED driver
    // Call the alarm LED driver functions
    // State Action
    ALARM_LED_Status = ALARM_LED_INIT;

    // Check event and update state
    ALARM_LED_state = STATE(ALARM_LED_WAITING);
}

STATE_DEFINE(ALARM_LED_WAITING)
{
    // State Action
    ALARM_LED_Status = ALARM_LED_WAITING;
}
```



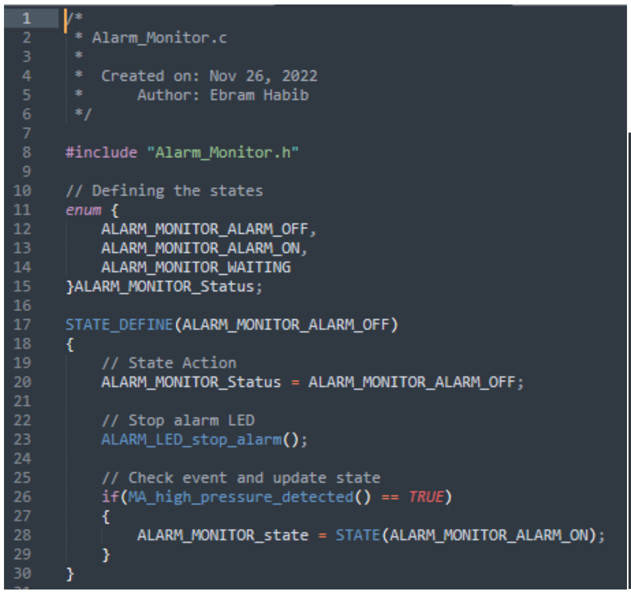
```c
STATE_DEFINE(ALARM_LED_ON)
{
    // State Action
    ALARM_LED_Status = ALARM_LED_ON;

    // Start Alarm Actuator
    GPIO_Set_Alarm_LED(TRUE);

    // Check event and update state
    ALARM_LED_state = STATE(ALARM_LED_WAITING);
}

STATE_DEFINE(ALARM_LED_OFF)
{
    // State Action
    ALARM_LED_Status = ALARM_LED_OFF;

    // Start Alarm Actuator
    GPIO_Set_Alarm_LED(FALSE);

    // Check event and update state
    ALARM_LED_state = STATE(ALARM_LED_WAITING);
}

void ALARM_LED_start_alarm(void)
{
    // Update State
    ALARM_LED_state = STATE(ALARM_LED_ON);
}

void ALARM_LED_stop_alarm(void)
{
    // Update State
    ALARM_LED_state = STATE(ALARM_LED_OFF);
}
```

Alarm_Monitor

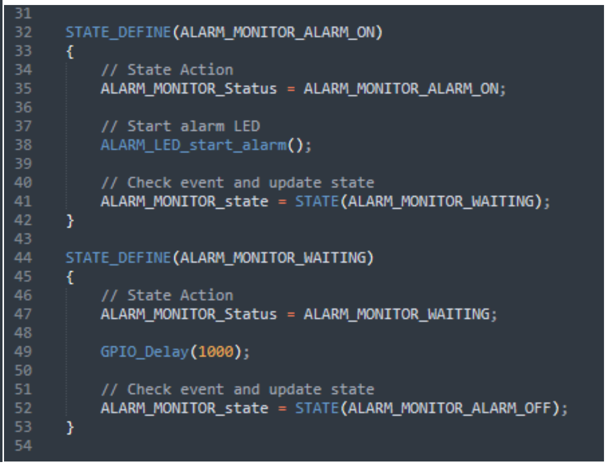




```c
/*
 * Alarm_Monitor.h
 *
 *  Created on: Nov 26, 2022
 *      Author: Ebram Habib
 */

#ifndef ALARM_MONITOR_H_
#define ALARM_MONITOR_H_

#include "State.h"

//Declare State Functions of Alarm Monitor
STATE_DEFINE(ALARM_MONITOR_ALARM_OFF);
STATE_DEFINE(ALARM_MONITOR_ALARM_ON);
STATE_DEFINE(ALARM_MONITOR_WAITING);

//State Pointer to function
extern void (*ALARM_MONITOR_state)();

#endif /* ALARM_MONITOR_H_ */
```

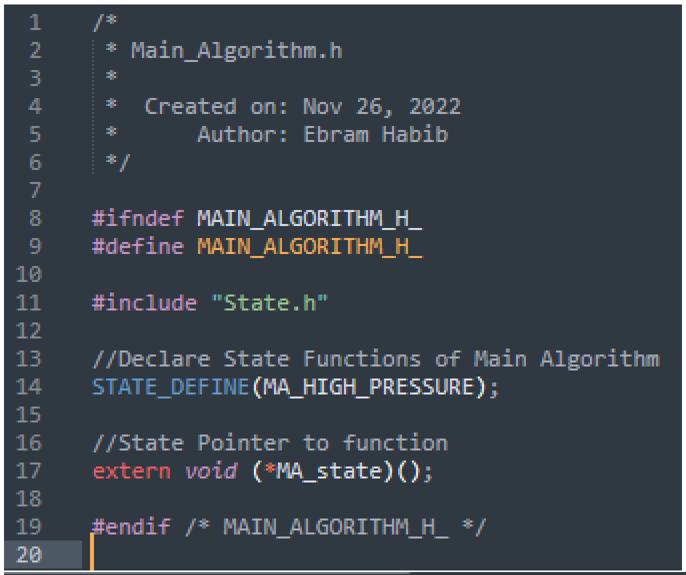


```c
/*
 * Alarm_Monitor.c
 *
 *  Created on: Nov 26, 2022
 *      Author: Ebram Habib
 */

#include "Alarm_Monitor.h"

// Defining the states
enum {
    ALARM_MONITOR_ALARM_OFF,
    ALARM_MONITOR_ALARM_ON,
    ALARM_MONITOR_WAITING
}ALARM_MONITOR_Status;

STATE_DEFINE(ALARM_MONITOR_ALARM_OFF)
{
    // State Action
    ALARM_MONITOR_Status = ALARM_MONITOR_ALARM_OFF;

    // Stop alarm LED
    ALARM_LED_stop_alarm();

    // Check event and update state
    if(MA_high_pressure_detected() == TRUE)
    {
        ALARM_MONITOR_state = STATE(ALARM_MONITOR_ALARM_ON);
    }
}
```

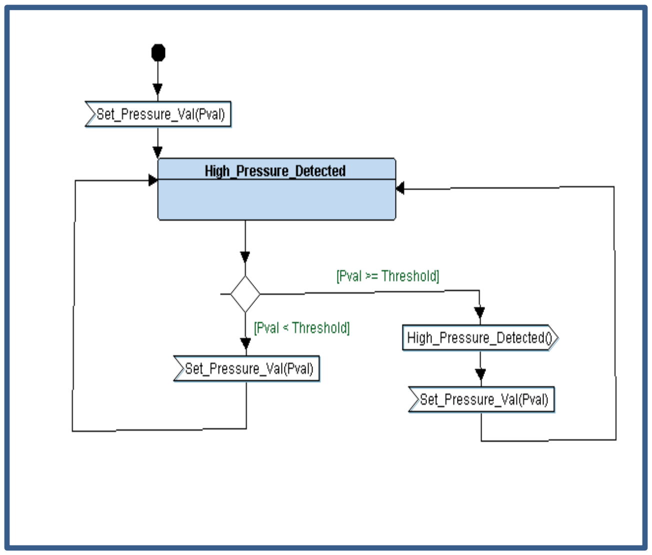


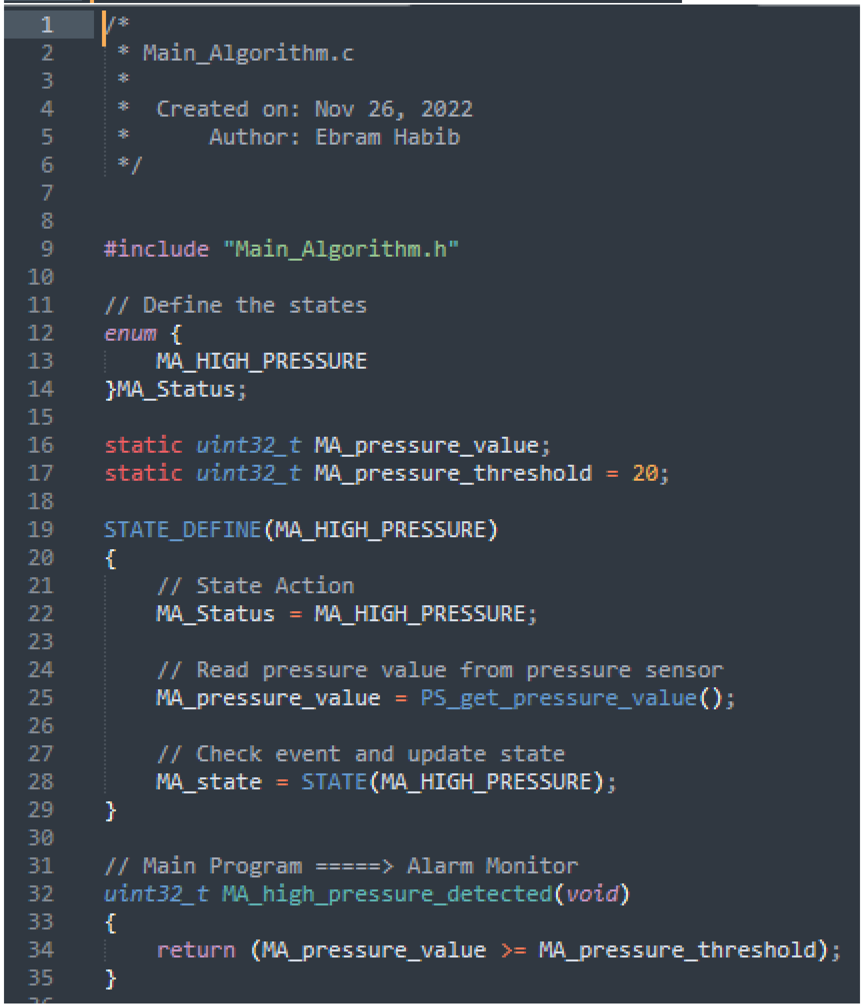
```c

STATE_DEFINE(ALARM_MONITOR_ALARM_ON)
{
    // State Action
    ALARM_MONITOR_Status = ALARM_MONITOR_ALARM_ON;

    // Start alarm LED
    ALARM_LED_start_alarm();

    // Check event and update state
    ALARM_MONITOR_state = STATE(ALARM_MONITOR_WAITING);
}

STATE_DEFINE(ALARM_MONITOR_WAITING)
{
    // State Action
    ALARM_MONITOR_Status = ALARM_MONITOR_WAITING;

    GPIO_Delay(1000);

    // Check event and update state
    ALARM_MONITOR_state = STATE(ALARM_MONITOR_ALARM_OFF);
}

```

Main_Algorithm



```c
/*
 * Main_Algorithm.h
 *
 *  Created on: Nov 26, 2022
 *      Author: Ebram Habib
 */

#ifndef MAIN_ALGORITHM_H_
#define MAIN_ALGORITHM_H_

#include "State.h"

//Declare State Functions of Main Algorithm
STATE_DEFINE(MA_HIGH_PRESSURE);

//State Pointer to function
extern void (*MA_state)();

#endif /* MAIN_ALGORITHM_H_ */

```

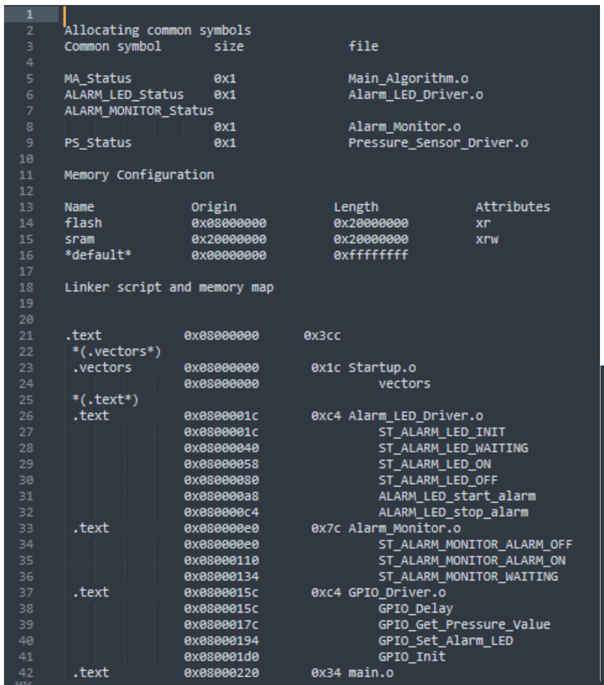




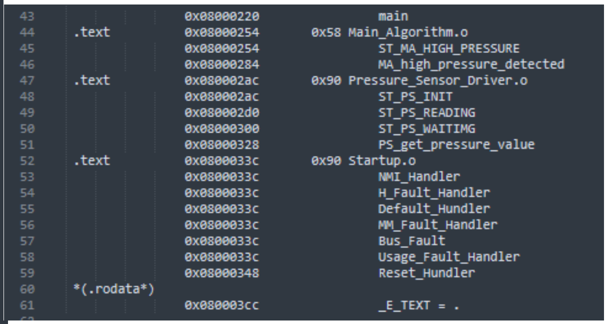
```c
/*
 * Main_Algorithm.c
 *
 *  Created on: Nov 26, 2022
 *      Author: Ebram Habib
 */


#include "Main_Algorithm.h"

// Define the states
enum {
    MA_HIGH_PRESSURE
}MA_Status;

static uint32_t MA_pressure_value;
static uint32_t MA_pressure_threshold = 20;

STATE_DEFINE(MA_HIGH_PRESSURE)
{
    // State Action
    MA_Status = MA_HIGH_PRESSURE;

    // Read pressure value from pressure sensor
    MA_pressure_value = PS_get_pressure_value();

    // Check event and update state
    MA_state = STATE(MA_HIGH_PRESSURE);
}

// Main Program =====> Alarm Monitor
uint32_t MA_high_pressure_detected(void)
{
    return (MA_pressure_value >= MA_pressure_threshold);
}
```

SW analysis:

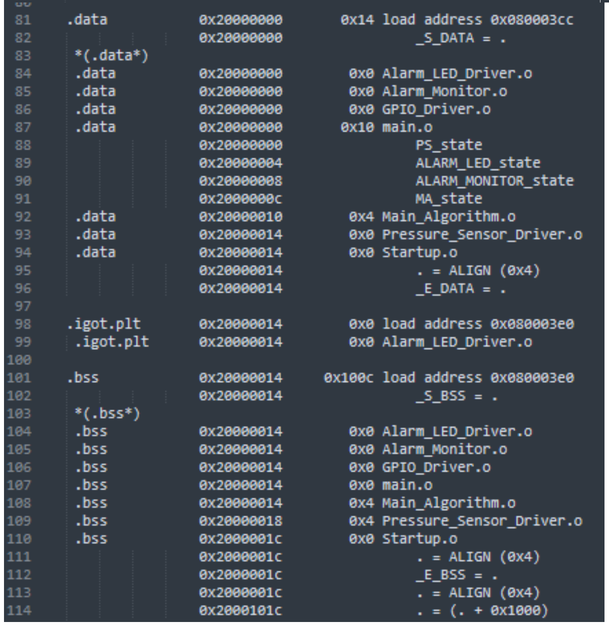
1- Map file



```

Allocating common symbols
Common symbol       size              file

MA_Status           0x1               Main_Algorithm.o
ALARM_LED_Status    0x1               Alarm_LED_Driver.o
ALARM_MONITOR_Status
                    0x1               Alarm_Monitor.o
PS_Status           0x1               Pressure_Sensor_Driver.o

Memory Configuration

Name             Origin             Length             Attributes
flash            0x08000000         0x20000000         xr
sram             0x20000000         0x20000000         xrw
*default*        0x00000000         0xffffffff

Linker script and memory map


.text           0x08000000      0x3cc
 *(.vectors*)
 .vectors       0x08000000       0x1c Startup.o
                0x08000000                vectors
 *(.text*)
 .text          0x0800001c       0xc4 Alarm_LED_Driver.o
                0x0800001c                ST_ALARM_LED_INIT
                0x08000040                ST_ALARM_LED_WAITING
                0x08000058                ST_ALARM_LED_ON
                0x08000080                ST_ALARM_LED_OFF
                0x080000a8                ALARM_LED_start_alarm
                0x080000c4                ALARM_LED_stop_alarm
 .text          0x080000e0       0x7c Alarm_Monitor.o
                0x080000e0                ST_ALARM_MONITOR_ALARM_OFF
                0x08000110                ST_ALARM_MONITOR_ALARM_ON
                0x08000134                ST_ALARM_MONITOR_WAITING
 .text          0x0800015c       0xc4 GPIO_Driver.o
                0x0800015c                GPIO_Delay
                0x0800017c                GPIO_Get_Pressure_Value
                0x08000194                GPIO_Set_Alarm_LED
                0x080001d0                GPIO_Init
 .text          0x08000220       0x34 main.o
```

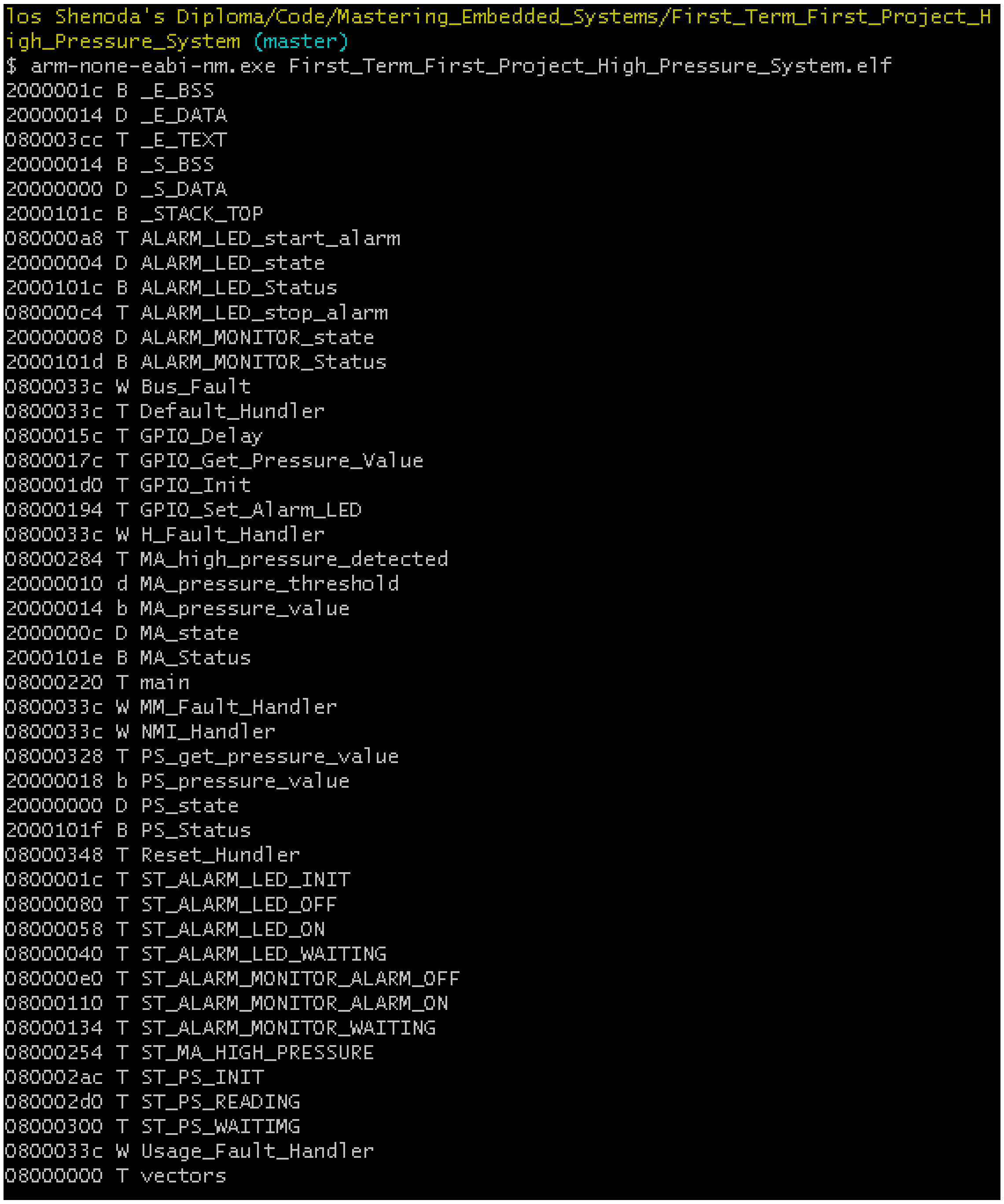


```
                0x08000220                main
 .text          0x08000254       0x58 Main_Algorithm.o
                0x08000254                ST_MA_HIGH_PRESSURE
                0x08000284                MA_high_pressure_detected
 .text          0x080002ac       0x90 Pressure_Sensor_Driver.o
                0x080002ac                ST_PS_INIT
                0x080002d0                ST_PS_READING
                0x08000300                ST_PS_WAITIMG
                0x08000328                PS_get_pressure_value
 .text          0x0800033c       0x90 Startup.o
                0x0800033c                NMI_Handler
                0x0800033c                H_Fault_Handler
                0x0800033c                Default_Hundler
                0x0800033c                MM_Fault_Handler
                0x0800033c                Bus_Fault
                0x0800033c                Usage_Fault_Handler
                0x08000348                Reset_Hundler
 *(.rodata*)
                0x080003cc                _E_TEXT = .
```



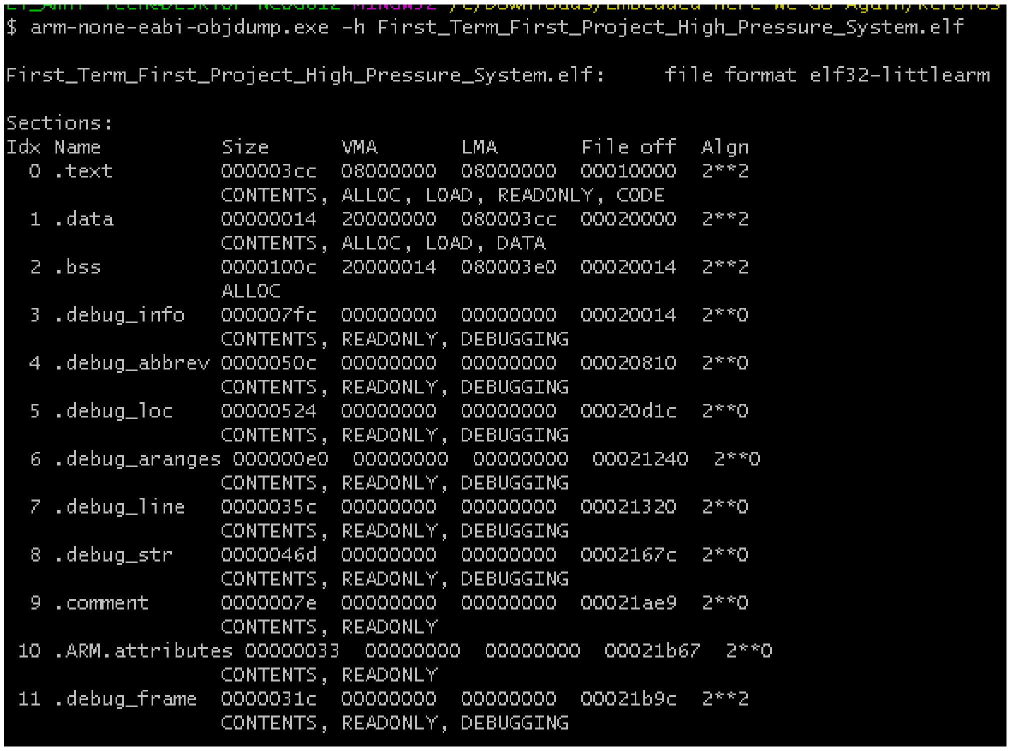
```
.data           0x20000000       0x14 load address 0x080003cc
                0x20000000                _S_DATA = .
 *(.data*)
 .data          0x20000000        0x0 Alarm_LED_Driver.o
 .data          0x20000000        0x0 Alarm_Monitor.o
 .data          0x20000000        0x0 GPIO_Driver.o
 .data          0x20000000       0x10 main.o
                0x20000000                PS_state
                0x20000004                ALARM_LED_state
                0x20000008                ALARM_MONITOR_state
                0x2000000c                MA_state
 .data          0x20000010        0x4 Main_Algorithm.o
 .data          0x20000014        0x0 Pressure_Sensor_Driver.o
 .data          0x20000014        0x0 Startup.o
                0x20000014                . = ALIGN (0x4)
                0x20000014                _E_DATA = .

.igot.plt       0x20000014        0x0 load address 0x080003e0
 .igot.plt      0x20000014        0x0 Alarm_LED_Driver.o

.bss            0x20000014      0x100c load address 0x080003e0
                0x20000014                _S_BSS = .
 *(.bss*)
 .bss           0x20000014        0x0 Alarm_LED_Driver.o
 .bss           0x20000014        0x0 Alarm_Monitor.o
 .bss           0x20000014        0x0 GPIO_Driver.o
 .bss           0x20000014        0x0 main.o
 .bss           0x20000014        0x4 Main_Algorithm.o
 .bss           0x20000018        0x4 Pressure_Sensor_Driver.o
 .bss           0x2000001c        0x0 Startup.o
                0x2000001c                . = ALIGN (0x4)
                0x2000001c                _E_BSS = .
                0x2000001c                . = ALIGN (0x4)
                0x2000101c                . = (. + 0x1000)
```

2- Symbols table



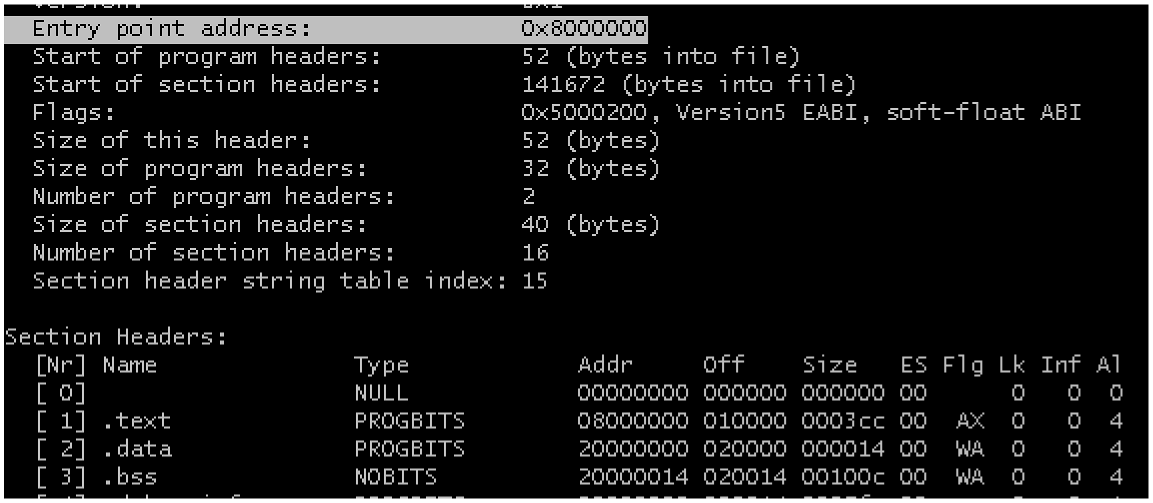
```
los Shenoda's Diploma/Code/Mastering_Embedded_Systems/First_Term_First_Project_H
igh_Pressure_System (master)
$ arm-none-eabi-nm.exe First_Term_First_Project_High_Pressure_System.elf
2000001c B _E_BSS
20000014 D _E_DATA
080003cc T _E_TEXT
20000014 B _S_BSS
20000000 D _S_DATA
2000101c B _STACK_TOP
080000a8 T ALARM_LED_start_alarm
20000004 D ALARM_LED_state
2000101c B ALARM_LED_Status
080000c4 T ALARM_LED_stop_alarm
20000008 D ALARM_MONITOR_state
2000101d B ALARM_MONITOR_Status
0800033c W Bus_Fault
0800033c T Default_Hundler
0800015c T GPIO_Delay
0800017c T GPIO_Get_Pressure_Value
080001d0 T GPIO_Init
08000194 T GPIO_Set_Alarm_LED
0800033c W H_Fault_Handler
08000284 T MA_high_pressure_detected
20000010 d MA_pressure_threshold
20000014 b MA_pressure_value
2000000c D MA_state
2000101e B MA_Status
08000220 T main
0800033c W MM_Fault_Handler
0800033c W NMI_Handler
08000328 T PS_get_pressure_value
20000018 b PS_pressure_value
20000000 D PS_state
2000101f B PS_Status
08000348 T Reset_Hundler
0800001c T ST_ALARM_LED_INIT
08000080 T ST_ALARM_LED_OFF
08000058 T ST_ALARM_LED_ON
08000040 T ST_ALARM_LED_WAITING
080000e0 T ST_ALARM_MONITOR_ALARM_OFF
08000110 T ST_ALARM_MONITOR_ALARM_ON
08000134 T ST_ALARM_MONITOR_WAITING
08000254 T ST_MA_HIGH_PRESSURE
080002ac T ST_PS_INIT
080002d0 T ST_PS_READING
08000300 T ST_PS_WAITIMG
0800033c W Usage_Fault_Handler
08000000 T vectors
```

3- Section tables



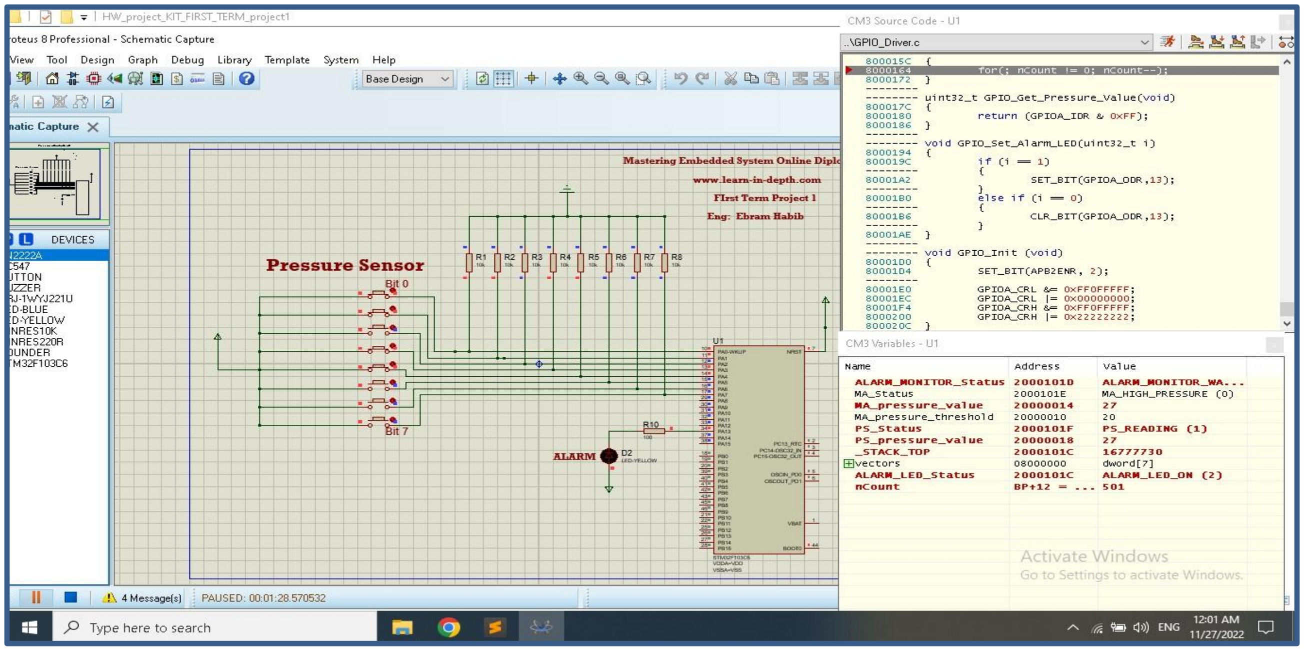
```
$ arm-none-eabi-objdump.exe -h First_Term_First_Project_High_Pressure_System.elf

First_Term_First_Project_High_Pressure_System.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000003cc  08000000  08000000  00010000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000014  20000000  080003cc  00020000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          0000100c  20000014  080003e0  00020014  2**2
                  ALLOC
  3 .debug_info   000007fc  00000000  00000000  00020014  2**0
                  CONTENTS, READONLY, DEBUGGING
  4 .debug_abbrev 0000050c  00000000  00000000  00020810  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    00000524  00000000  00000000  00020d1c  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 000000e0  00000000  00000000  00021240  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_line   0000035c  00000000  00000000  00021320  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .debug_str    0000046d  00000000  00000000  0002167c  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      0000007e  00000000  00000000  00021ae9  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  00021b67  2**0
                  CONTENTS, READONLY
 11 .debug_frame  0000031c  00000000  00000000  00021b9c  2**2
                  CONTENTS, READONLY, DEBUGGING
```

4- Entry Point Address



```
  Entry point address:               0x8000000
  Start of program headers:          52 (bytes into file)
  Start of section headers:          141672 (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:         2
  Size of section headers:           40 (bytes)
  Number of section headers:         16
  Section header string table index: 15

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        08000000 010000 0003cc 00  AX  0   0  4
  [ 2] .data             PROGBITS        20000000 020000 000014 00  WA  0   0  4
  [ 3] .bss              NOBITS          20000014 020014 00100c 00  WA  0   0  4
```

Proteus Simulation



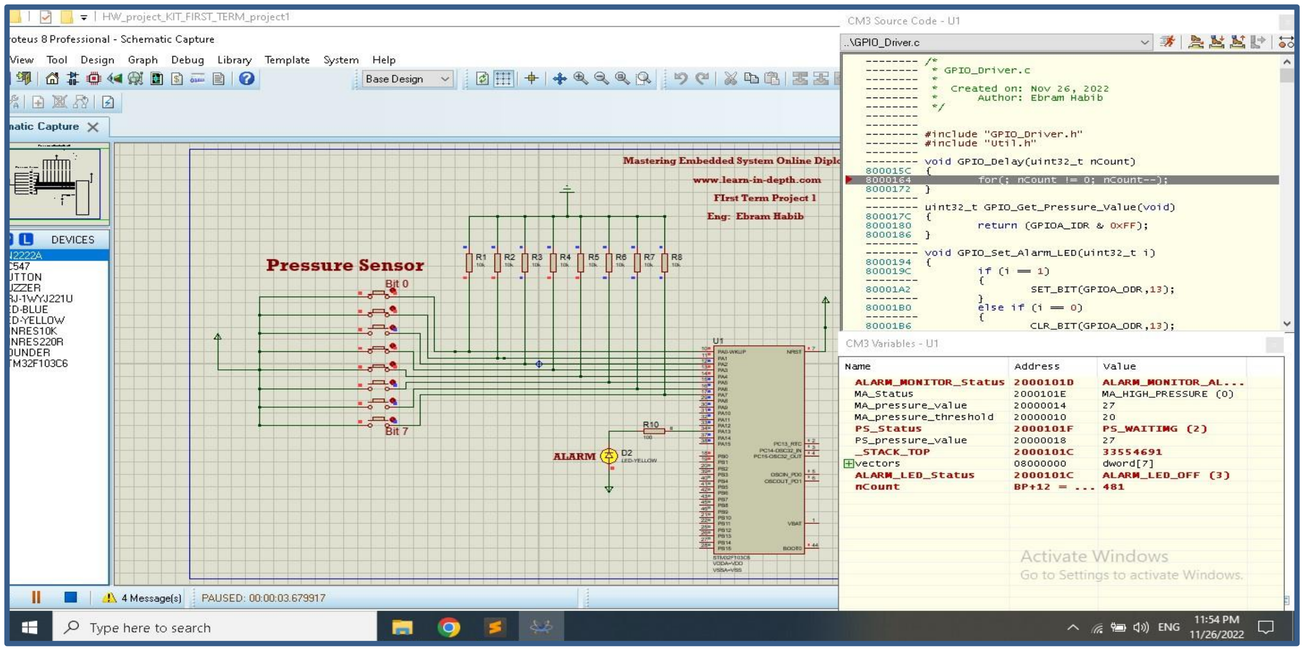
CM3 Source Code - U1

..\GPIO_Driver.c

```
800015C  {
8000164          for(; nCount != 0; nCount--);
8000172  }
--------
-------- uint32_t GPIO_Get_Pressure_Value(void)
800017C  {
8000180          return (GPIOA_IDR & 0xFF);
8000186  }
--------
-------- void GPIO_Set_Alarm_LED(uint32_t i)
8000194  {
800019C          if (i == 1)
--------         {
80001A2                  SET_BIT(GPIOA_ODR,13);
--------         }
80001B0          else if (i == 0)
--------         {
80001B6                  CLR_BIT(GPIOA_ODR,13);
--------         }
80001AE  }
--------
-------- void GPIO_Init (void)
80001D0  {
80001D4          SET_BIT(APB2ENR, 2);
--------
80001E0          GPIOA_CRL &= 0xFF0FFFFF;
80001EC          GPIOA_CRL |= 0x00000000;
80001F4          GPIOA_CRH &= 0xFF0FFFFF;
8000200          GPIOA_CRH |= 0x22222222;
800020C  }
```

CM3 Variables - U1

Name	Address	Value
ALARM_MONITOR_Status	2000101D	ALARM_MONITOR_WA...
MA_Status	2000101E	MA_HIGH_PRESSURE (0)
MA_pressure_value	20000014	27
MA_pressure_threshold	20000010	20
PS_Status	2000101F	PS_READING (1)
PS_pressure_value	20000018	27
_STACK_TOP	2000101C	16777730
⊞vectors	08000000	dword[7]
ALARM_LED_Status	2000101C	ALARM_LED_ON (2)
nCount	BP+12 = ...	501



CM3 Source Code - U1

..\GPIO_Driver.c

```
-------- /*
--------  * GPIO_Driver.c
--------  *
--------  *  Created on: Nov 26, 2022
--------  *      Author: Ebram Habib
--------  */
--------
--------
-------- #include "GPIO_Driver.h"
-------- #include "Util.h"
--------
-------- void GPIO_Delay(uint32_t nCount)
800015C  {
8000164          for(; nCount != 0; nCount--);
8000172  }
--------
-------- uint32_t GPIO_Get_Pressure_Value(void)
800017C  {
8000180          return (GPIOA_IDR & 0xFF);
8000186  }
--------
-------- void GPIO_Set_Alarm_LED(uint32_t i)
8000194  {
800019C          if (i == 1)
--------         {
80001A2                  SET_BIT(GPIOA_ODR,13);
--------         }
80001B0          else if (i == 0)
--------         {
80001B6                  CLR_BIT(GPIOA_ODR,13);
```

CM3 Variables - U1

Name	Address	Value
ALARM_MONITOR_Status	2000101D	ALARM_MONITOR_AL...
MA_Status	2000101E	MA_HIGH_PRESSURE (0)
MA_pressure_value	20000014	27
MA_pressure_threshold	20000010	20
PS_Status	2000101F	PS_WAITING (2)
PS_pressure_value	20000018	27
_STACK_TOP	2000101C	33554691
⊞vectors	08000000	dword[7]
ALARM_LED_Status	2000101C	ALARM_LED_OFF (3)
nCount	BP+12 = ...	481