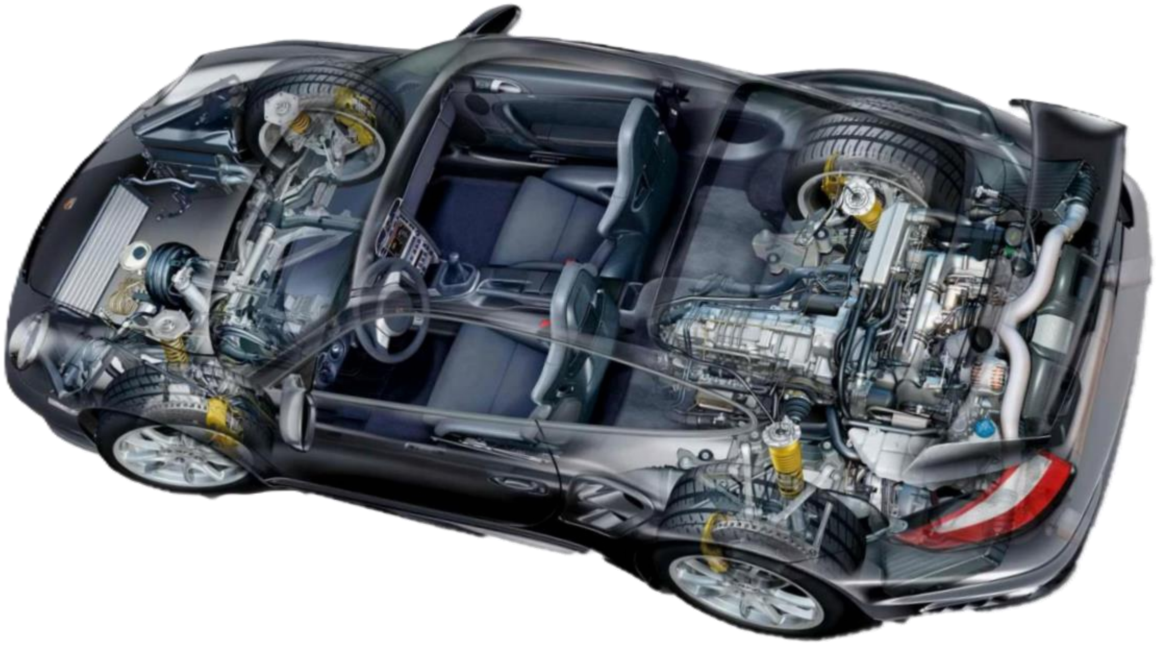


Mastering Embedded System Online Diploma

www.learn-in-depth.com

First Term (Final Project 1)

Eng. Habeba Ahmed Elbaghdady

Table of Contents:

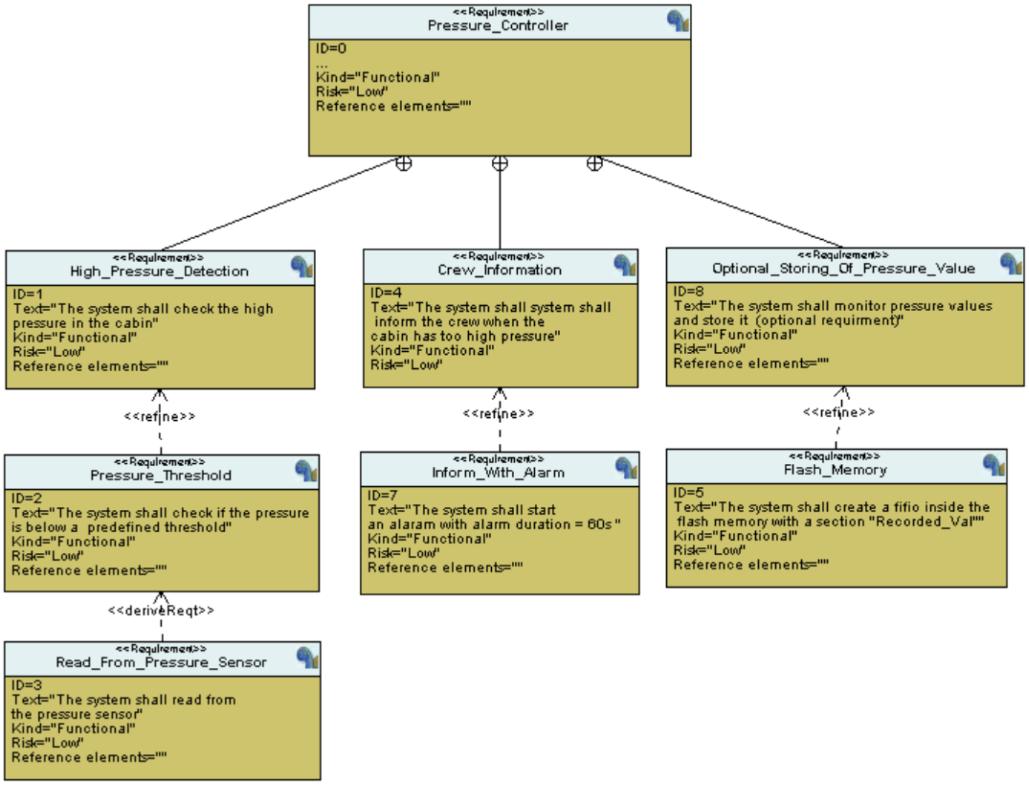
- Section Table.
- Map file.
- Hardware.

Case Study:

We have a pressure detection system in an airplane in which we have a sensor that measures the pressure and in case the pressure is more than 20 bar, an alarm is activated for a duration of 60 seconds to inform the crew of the airplane.

System Requirements:

We have some requirements in which the system shall do. There is a part of the requirements which is storing the pressure values in a flash memory.
This requirement is not implemented in the first release of the system.



<<Requirement>>
Pressure_Controller
ID=0
...
Kind="Functional"
Risk="Low"
Reference elements=""

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

<<Requirement>>
Crew_Information
ID=4
Text="The system shall system shall inform the crew when the cabin has too high pressure"
Kind="Functional"
Risk="Low"

<<Requirement>>
Optional_Storing_Of_Pressure_Value
ID=8
Text="The system shall monitor pressure values and store it (optional requirment)"
Kind="Functional"
Risk="Low"
Reference elements=""

<<Requirement>>
Pressure_Threshold
ID=2
Text="The system shall check if the pressure is below a predefined threshold"
Kind="Functional"
Risk="Low"
Reference elements=""

<<Requirement>>
Inform_With_Alarm
ID=7
Text="The system shall start an alaram with alarm duration = 60s "
Kind="Functional"
Risk="Low"
Reference elements=""

<<Requirement>>
Flash_Memory
ID=5
Text="The system shall create a fifio inside the flash memory with a section "Recorded_Val""
Kind="Functional"
Risk="Low"
Reference elements=""

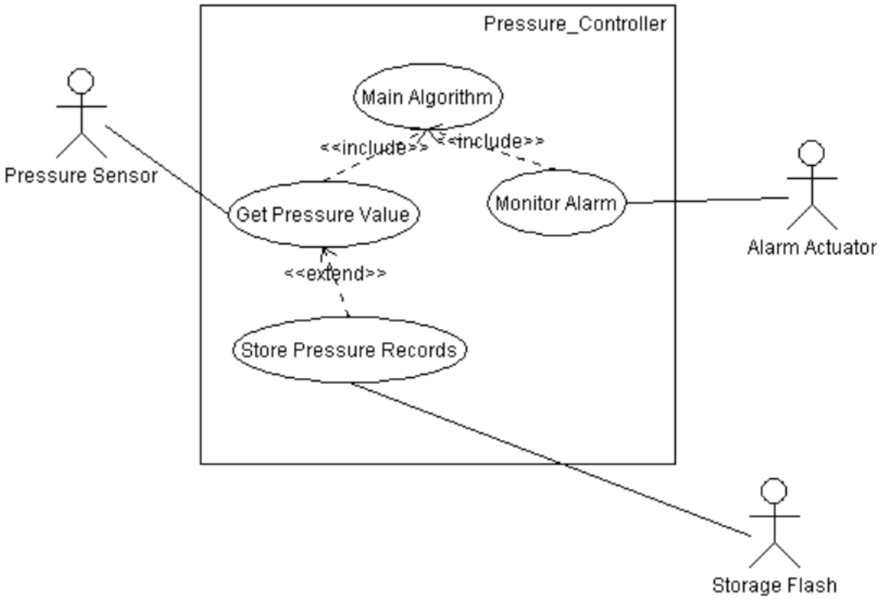
<<Requirement>>
Read_From_Pressure_Sensor
ID=3
Text="The system shall read from the pressure sensor"
Kind="Functional"
Risk="Low"
Reference elements=""

System Analysis:

The system analysis includes the use-case diagram , activity diagram, and sequence diagram to show the interactions between actors and our system.

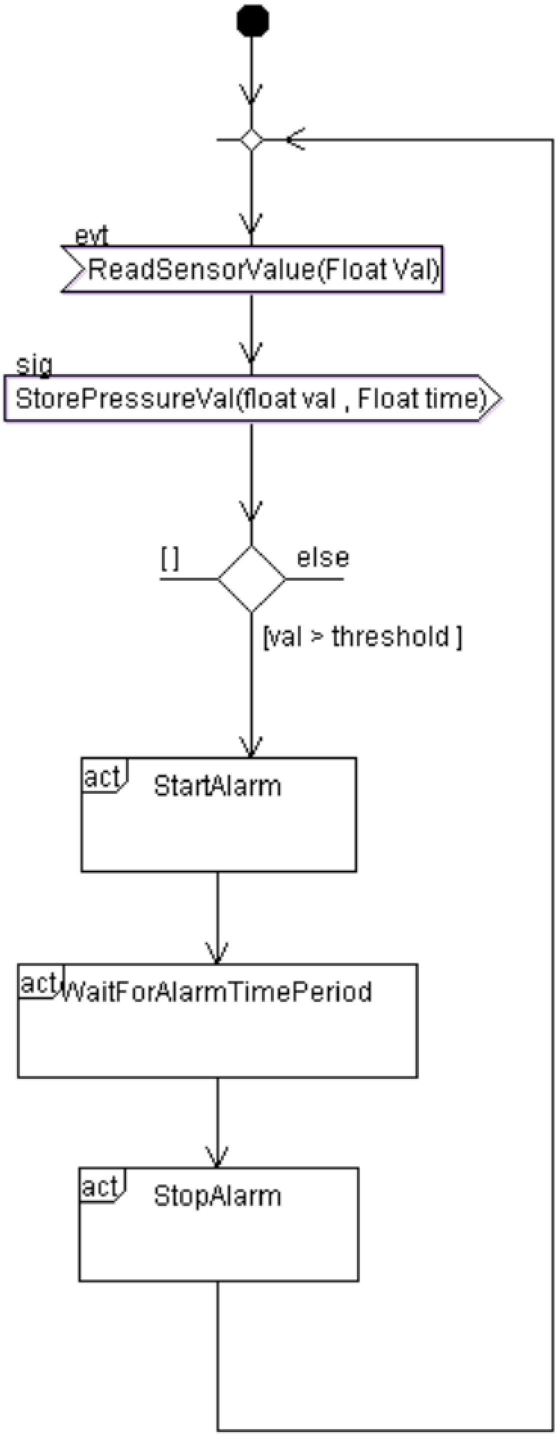
1. Use-Case Diagram

   It shows our actors which are pressure sensor, alarm actuator and storage flash and how they act with our system.
   The system has a main algorithm that takes the pressure sensor values and checks if it exceeds our threshold or not, if it does, the main algorithm makes the alarm work to inform the crew.
   The storing of the pressure sensors values will be in the extended version of our system.



2. Activity Diagram:

The system waits for the event of sensor reading that gets the pressure value form the sensor, it stores the value of the sensor and checks if the value of the pressure sensor is more than 20, if it does, it starts an alarm with a duration of 60 seconds to inform the crew and stops the alarm.

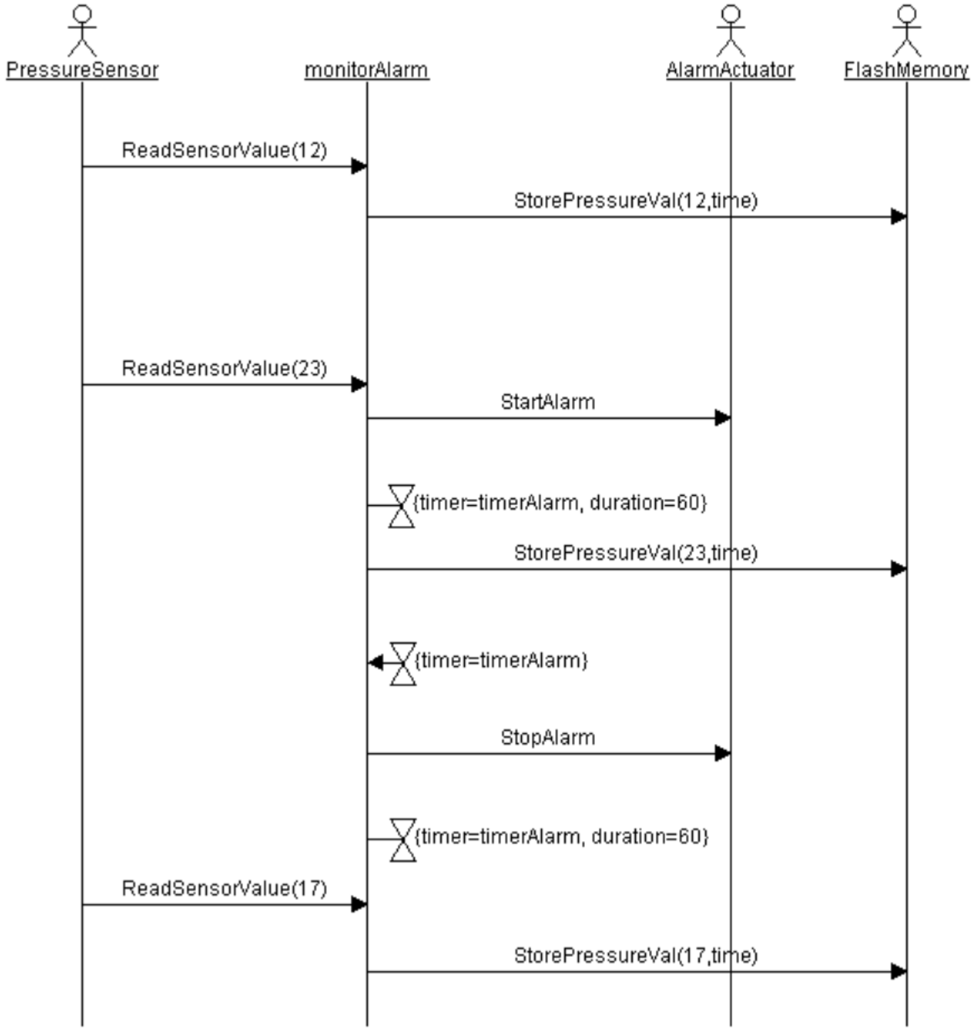

evt
ReadSensorValue(Float Val)
sig
StorePressureVal(float val , Float time)
[]
else
[val > threshold]
act StartAlarm
act WaitForAlarmTimePeriod
act StopAlarm

3. Sequence Diagram:

In this diagram, we have different interactions between the actors of our system.

Pressure sensor reads the first value which is 12, this value is less than the threshold, so the main algorithm only stores the value and return to read from the sensor.

When the value read from the sensor is more than the threshold it starts the alarm for a duration of 60 seconds, and it stores the value.



System Design:

1.Block Diagram:

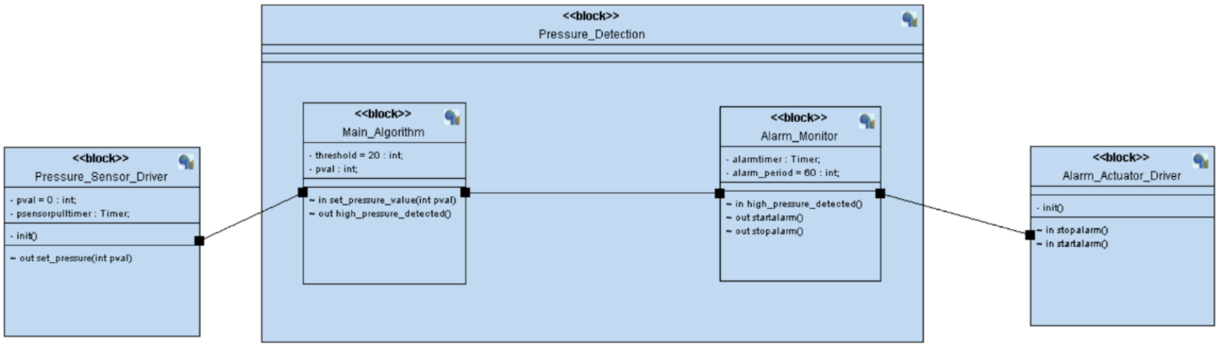
We have pressure sensor driver includes an init function and it measures the surrounding pressure and send the signal with its value to the main algorithm.

The main block is the pressure detection block which contains the main algorithm and the alarm monitor.

Main Algorithm takes the pressure value from the pressure sensor and compare it with the threshold if it is more than the threshold it sends a signal that there is high pressure detected to the alarm monitor.

Alarm monitor has the timer and its duration, it sends a signal of start alarm and stop alarm to the Alarm actuator driver.

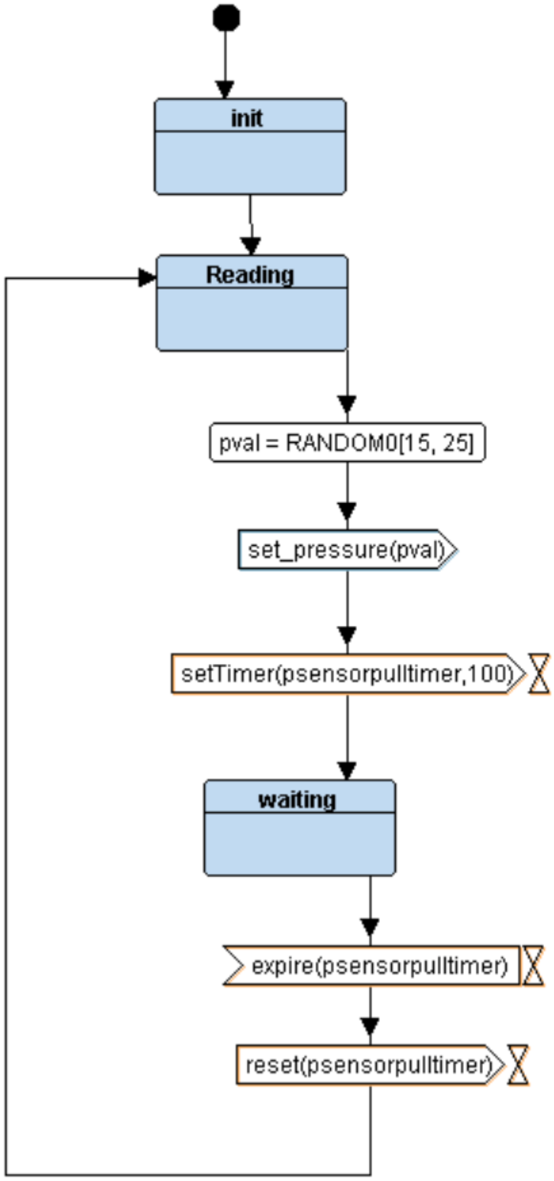
Alarm actuator driver is used to start or stop the alarm according to the signal that came from the alarm monitor.



2.State Machine

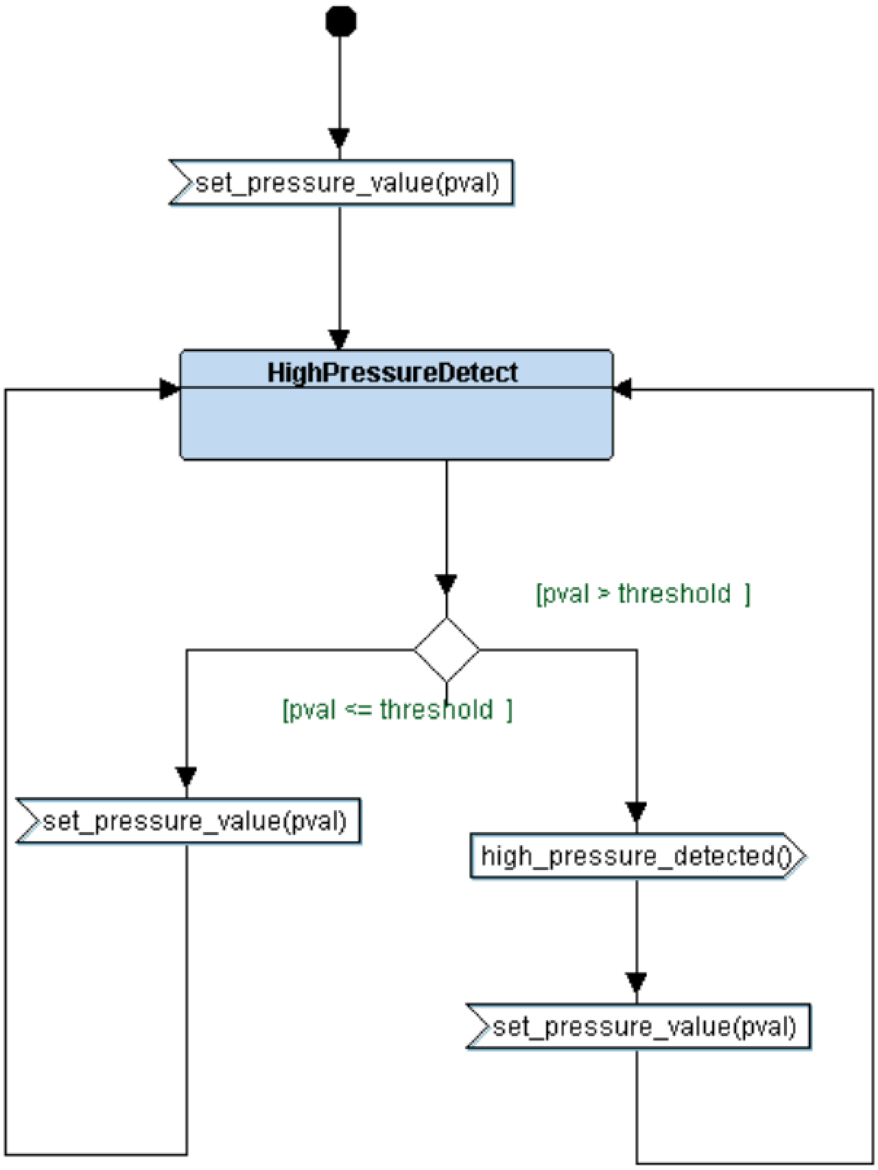
- Pressure Sensor driver:

It is first initialized, and it starts reading a value from a random generator.
It sets the value of pressure and sends the signal to the main algorithm and
It starts a timer at which at the end of it rereads the pressure value.



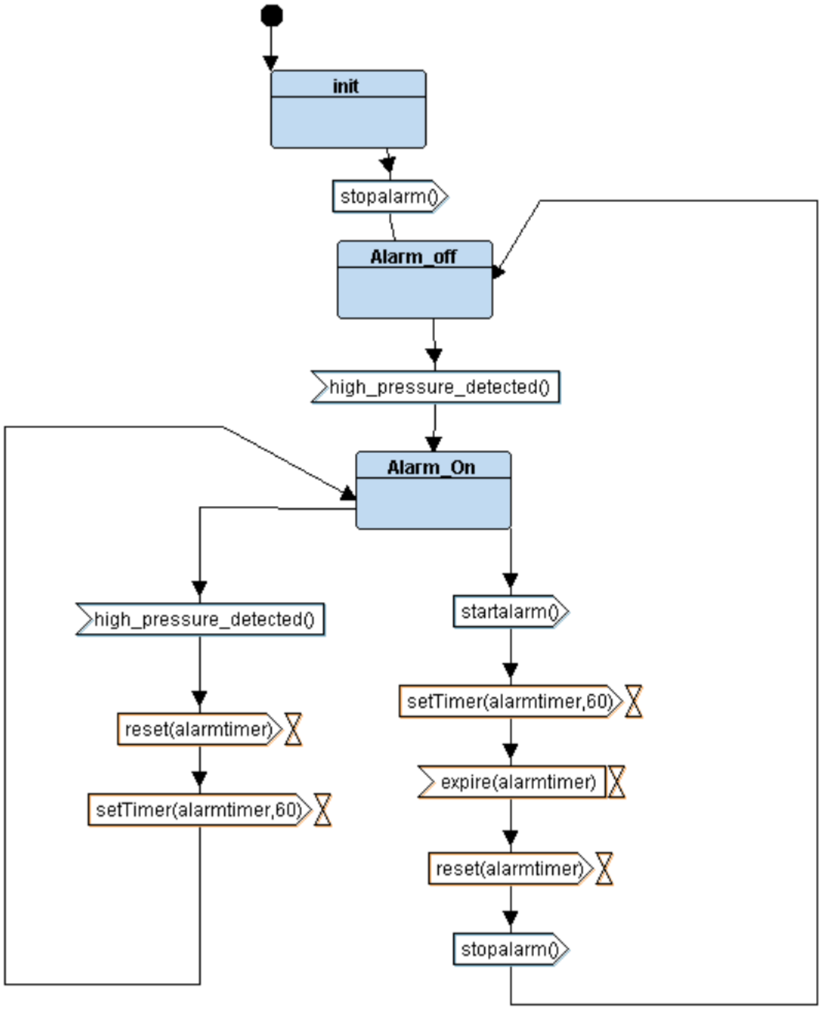
- Main Algorithm

The main algorithm takes the value from the pressure sensor and compare the value with the threshold if it is greater than the threshold a signal of high detected pressure is sent to the alarm monitor, if it is less than it rereads the pressure again.



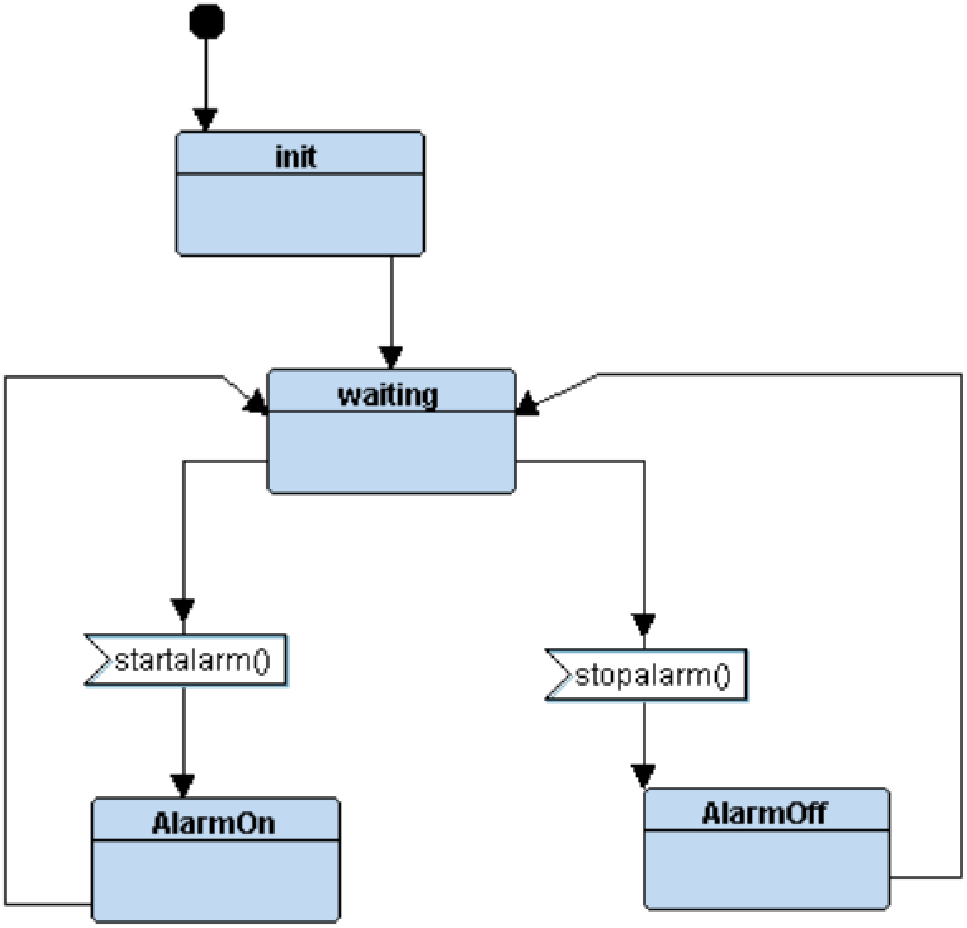
- Alarm Monitor

The Alarm Monitor initialized, and it stops the alarm.
If the high detected pressure signal is sent the alarm is started with the duration of 60 seconds. After the duration of the alarm is finished, it returns to alarm off, if another high-pressure signal is sent it reset the alarm timer and the alarm works for the specified duration.



- Alarm Actuator Driver:

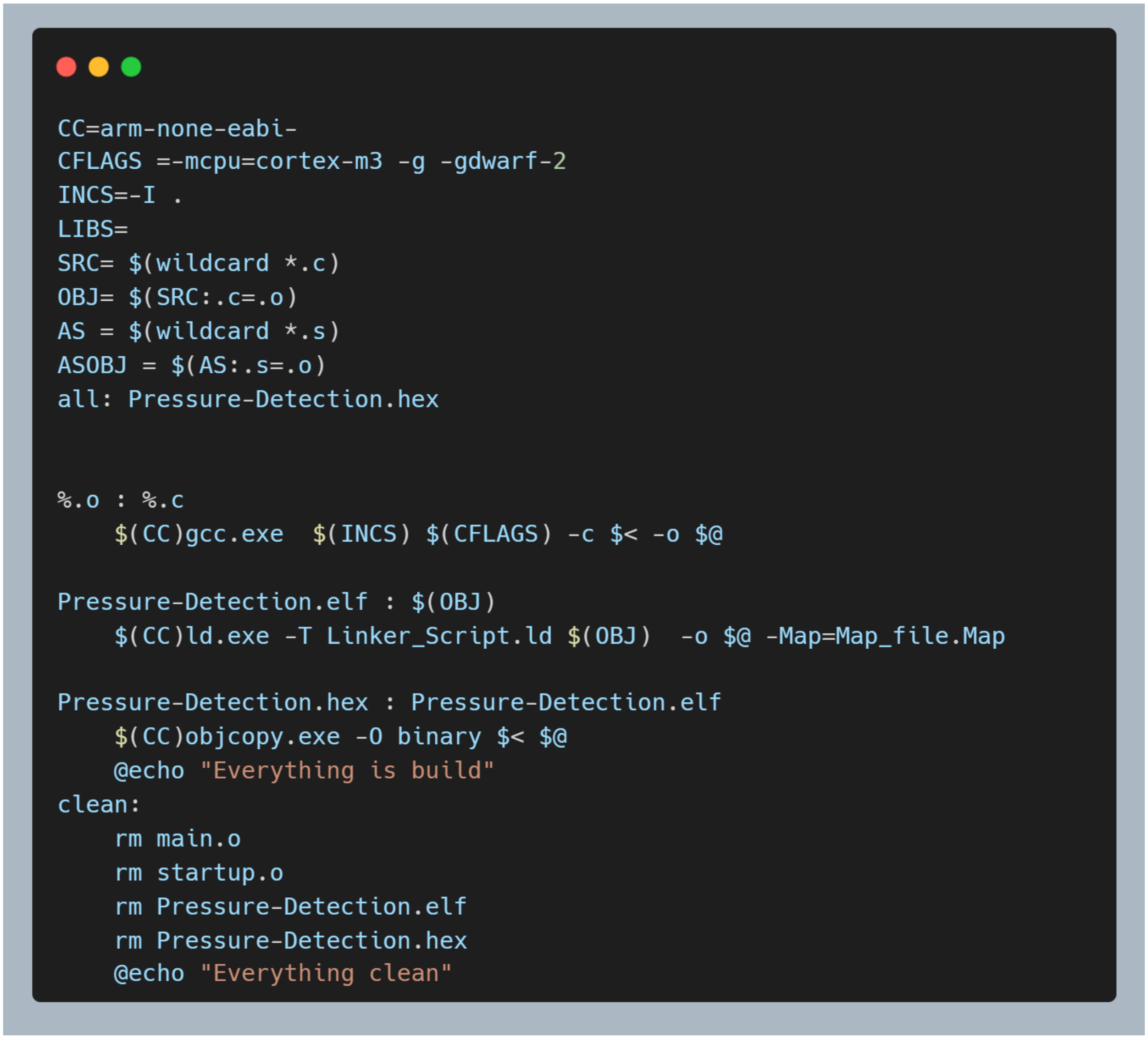
The Alarm Actuator is initialized and stays in the waiting state until a signal comes from the Alarm Monitor to start the alarm or stop it.



Implemenation

1.Make File

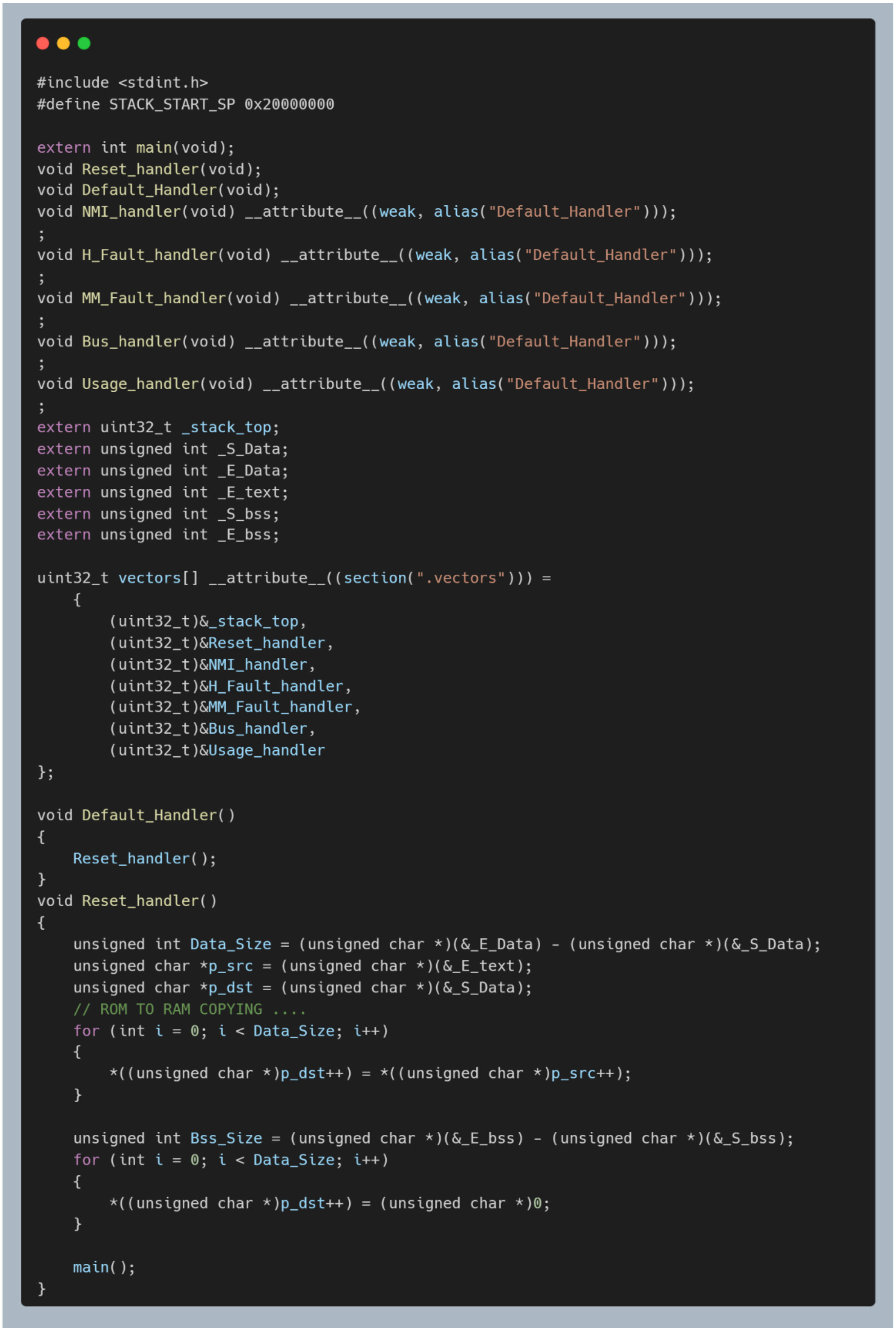
Make file is used to automate the build of our code files into an executable that can run on our board.



```
CC=arm-none-eabi-
CFLAGS =-mcpu=cortex-m3 -g -gdwarf-2
INCS=-I .
LIBS=
SRC= $(wildcard *.c)
OBJ= $(SRC:.c=.o)
AS = $(wildcard *.s)
ASOBJ = $(AS:.s=.o)
all: Pressure-Detection.hex


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

Pressure-Detection.elf : $(OBJ)
    $(CC)ld.exe -T Linker_Script.ld $(OBJ)  -o $@ -Map=Map_file.Map

Pressure-Detection.hex : Pressure-Detection.elf
    $(CC)objcopy.exe -O binary $< $@
    @echo "Everything is build"
clean:
    rm main.o
    rm startup.o
    rm Pressure-Detection.elf
    rm Pressure-Detection.hex
    @echo "Everything clean"
```

2.Startup

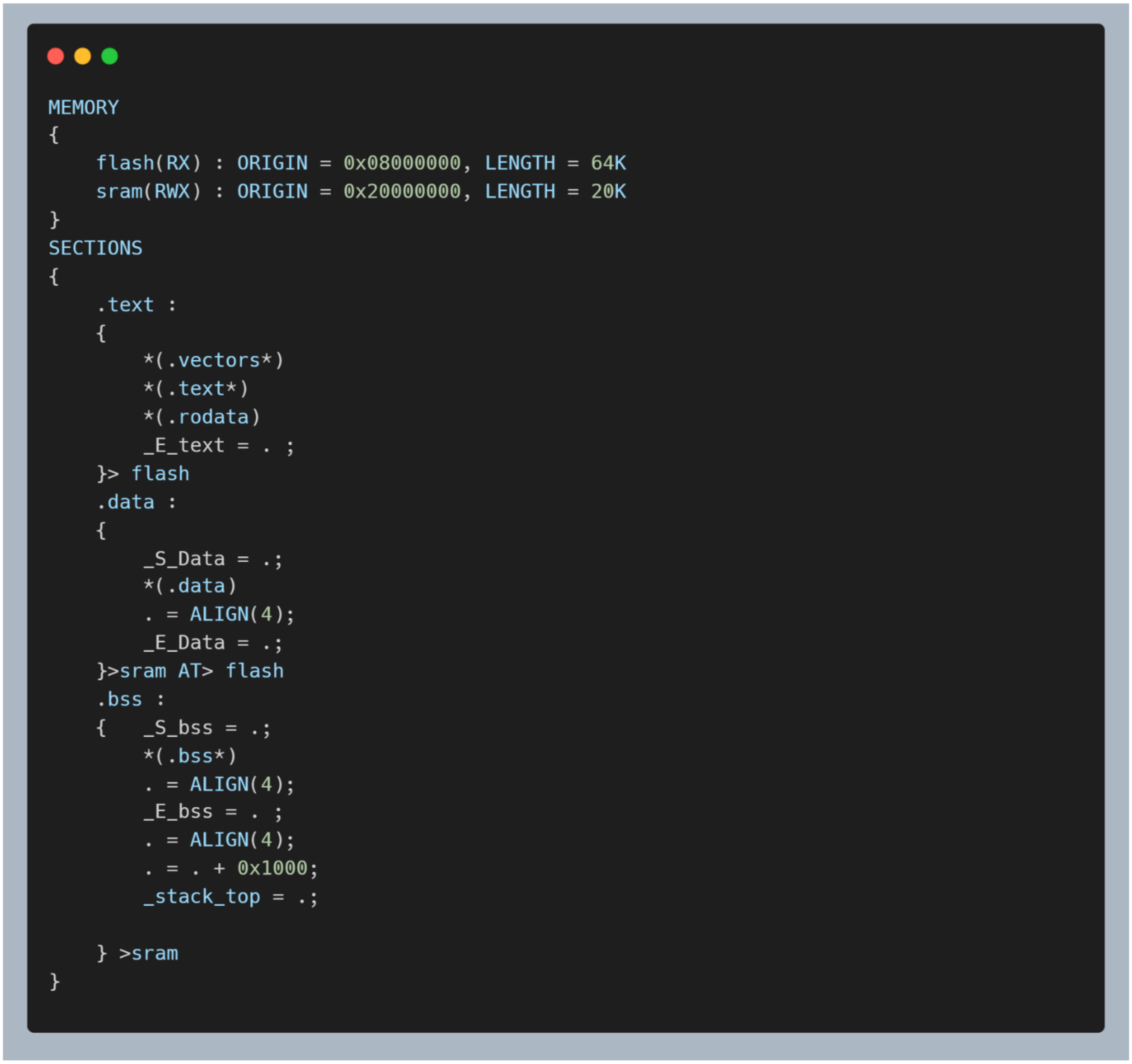
It is the file that run before the main when the board is in power on /Reset Mode.



```c
#include <stdint.h>
#define STACK_START_SP 0x20000000

extern int main(void);
void Reset_handler(void);
void Default_Handler(void);
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_handler(void) __attribute__((weak, alias("Default_Handler")));
;
void Usage_handler(void) __attribute__((weak, alias("Default_Handler")));
;
extern uint32_t _stack_top;
extern unsigned int _S_Data;
extern unsigned int _E_Data;
extern unsigned int _E_text;
extern unsigned int _S_bss;
extern unsigned int _E_bss;

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_handler,
        (uint32_t)&Usage_handler
};

void Default_Handler()
{
    Reset_handler();
}
void Reset_handler()
{
    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);
    // ROM TO RAM COPYING ....
    for (int i = 0; i < Data_Size; i++)
    {
        *((unsigned char *)p_dst++) = *((unsigned char *)p_src++);
    }

    unsigned int Bss_Size = (unsigned char *)(&_E_bss) - (unsigned char *)(&_S_bss);
    for (int i = 0; i < Data_Size; i++)
    {
        *((unsigned char *)p_dst++) = (unsigned char *)0;
    }

    main();
}
```

3.Linker Script.

The linker script is created to link our code and store them in different sections and locates them in flash Memory or RAM.



```
MEMORY
{
    flash(RX) : ORIGIN = 0x08000000, LENGTH = 64K
    sram(RWX) : ORIGIN = 0x20000000, LENGTH = 20K
}
SECTIONS
{
    .text :
    {
        *(.vectors*)
        *(.text*)
        *(.rodata)
        _E_text = . ;
    }> flash
    .data :
    {
        _S_Data = .;
        *(.data)
        . = ALIGN(4);
        _E_Data = .;
    }>sram AT> flash
    .bss :
    {   _S_bss = .;
        *(.bss*)
        . = ALIGN(4);
        _E_bss = . ;
        . = ALIGN(4);
        . = . + 0x1000;
        _stack_top = .;

    } >sram
}
```

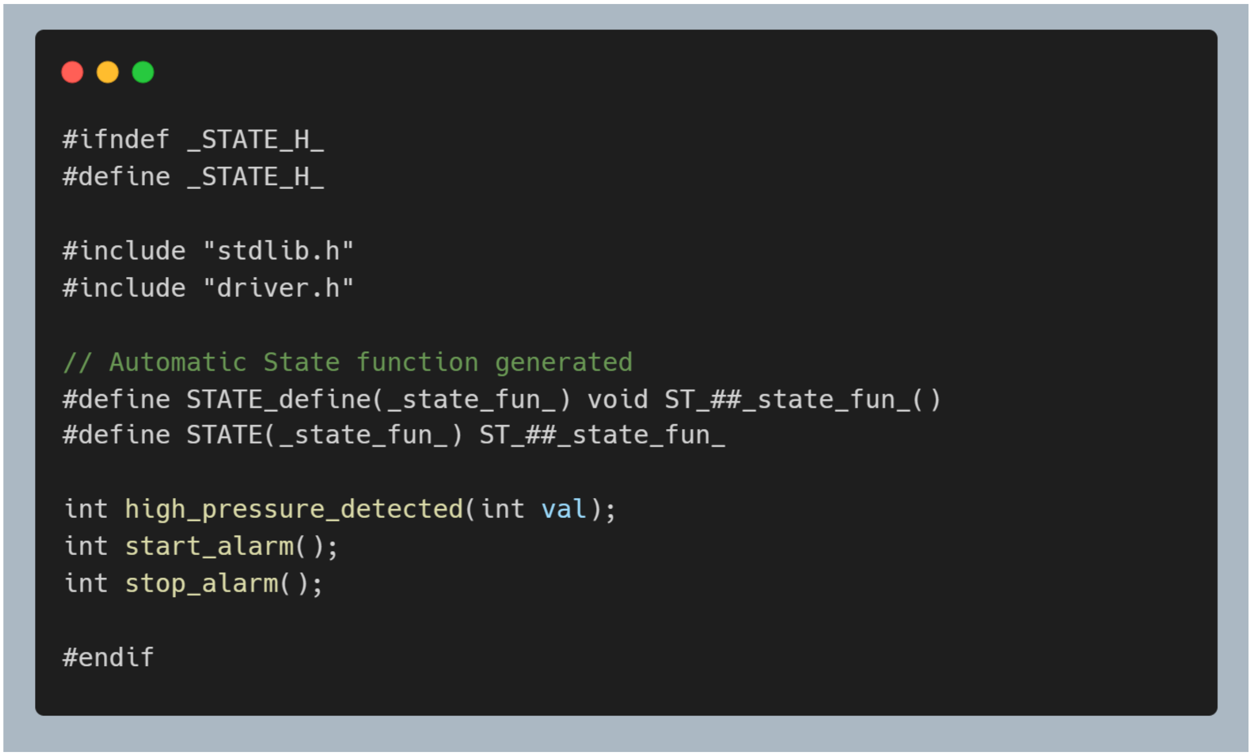
4.Cross Toolchain

We need arm toolchain downloaded on our host Machine to compile our code to the specifications of the target machine.

5.C Code.

- States

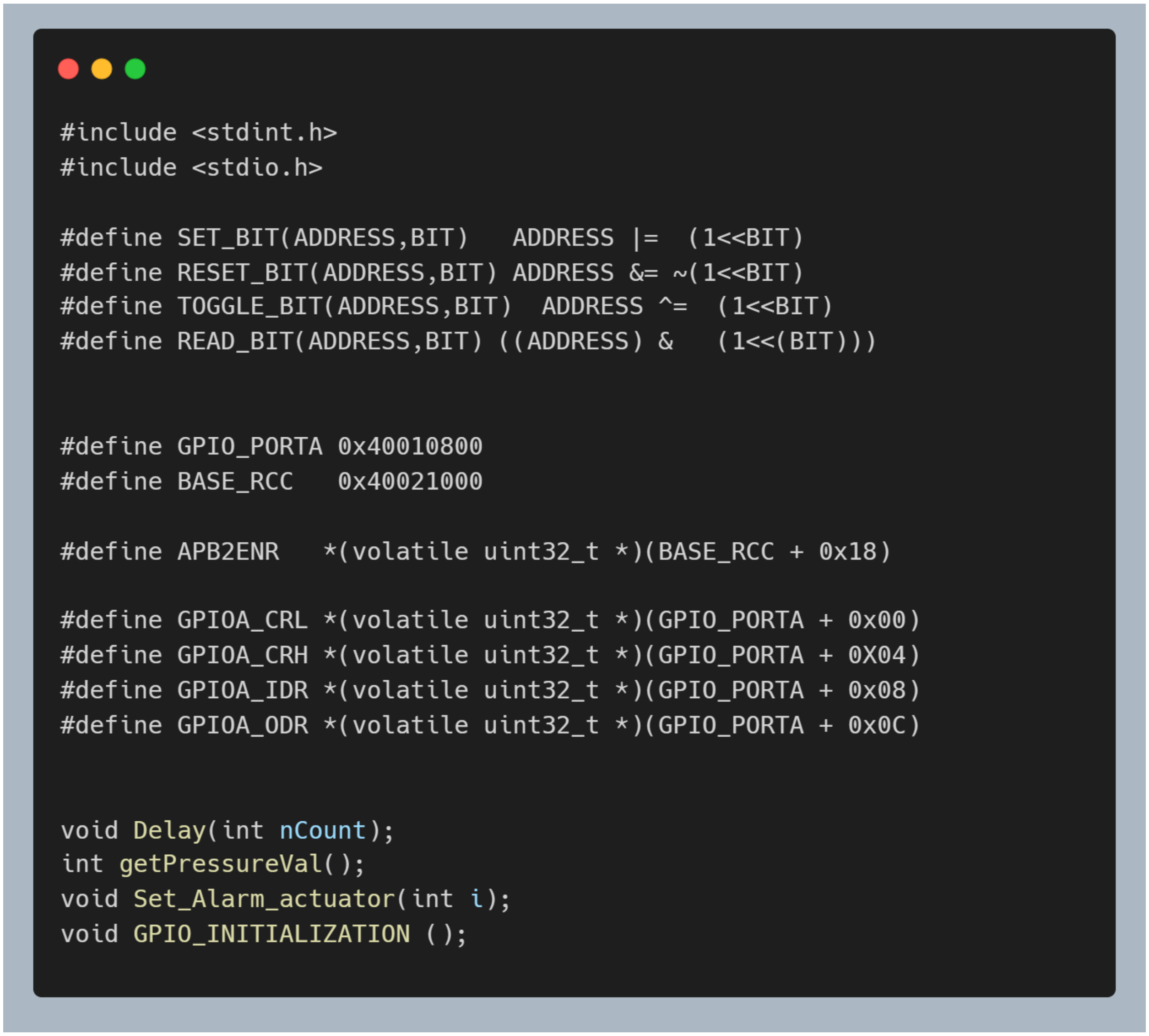
The State.h file is used to define any state machine and contains the signals sent from one block to another.



```
#ifndef _STATE_H_
#define _STATE_H_

#include "stdlib.h"
#include "driver.h"

// Automatic State function generated
#define STATE_define(_state_fun_) void ST_##_state_fun_()
#define STATE(_state_fun_) ST_##_state_fun_

int high_pressure_detected(int val);
int start_alarm();
int stop_alarm();

#endif
```

- Drivers

- Driver.h

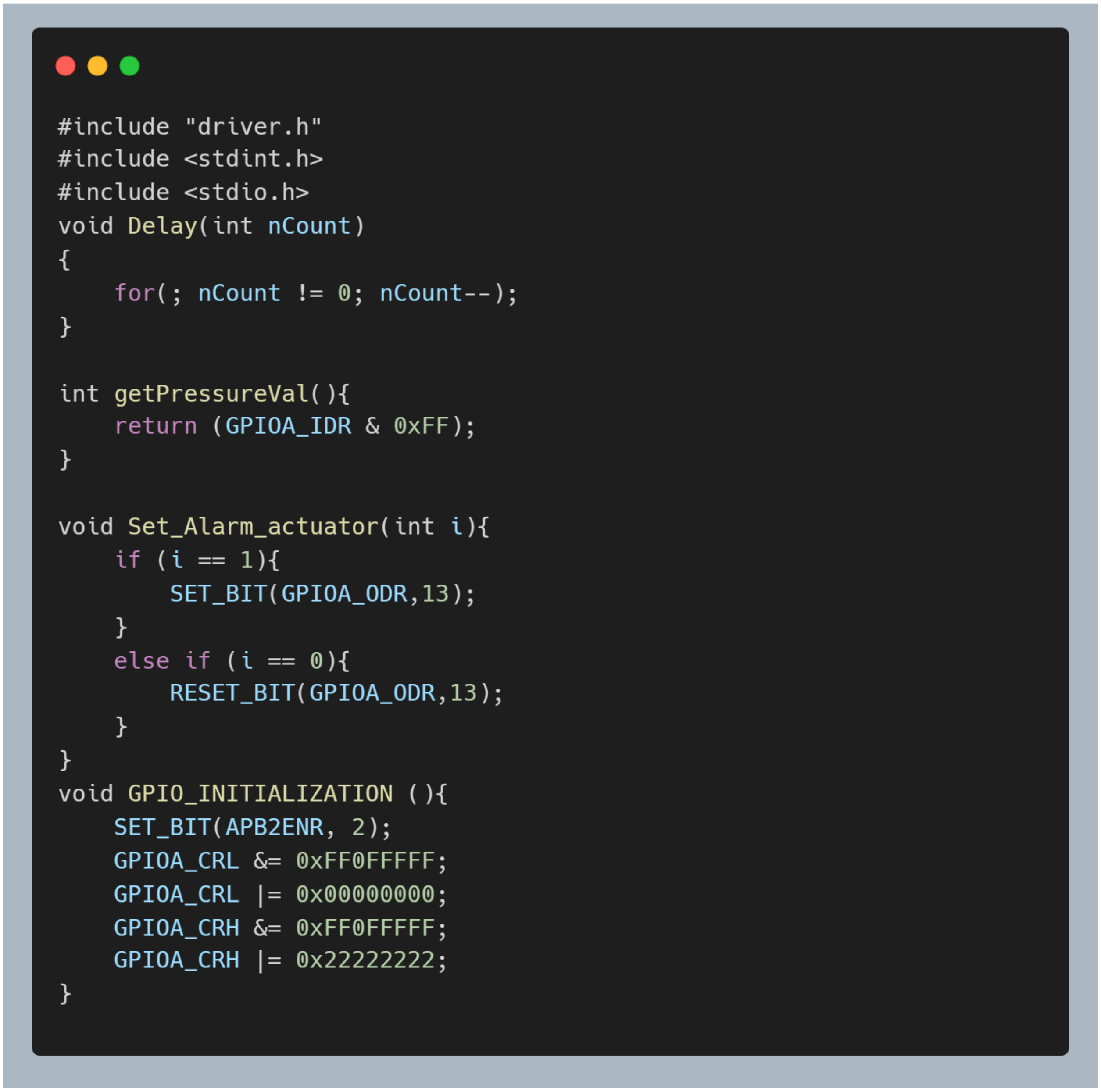


```
#include <stdint.h>
#include <stdio.h>

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


#define GPIO_PORTA 0x40010800
#define BASE_RCC   0x40021000

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

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


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

- Driver.c

It has the definitions of

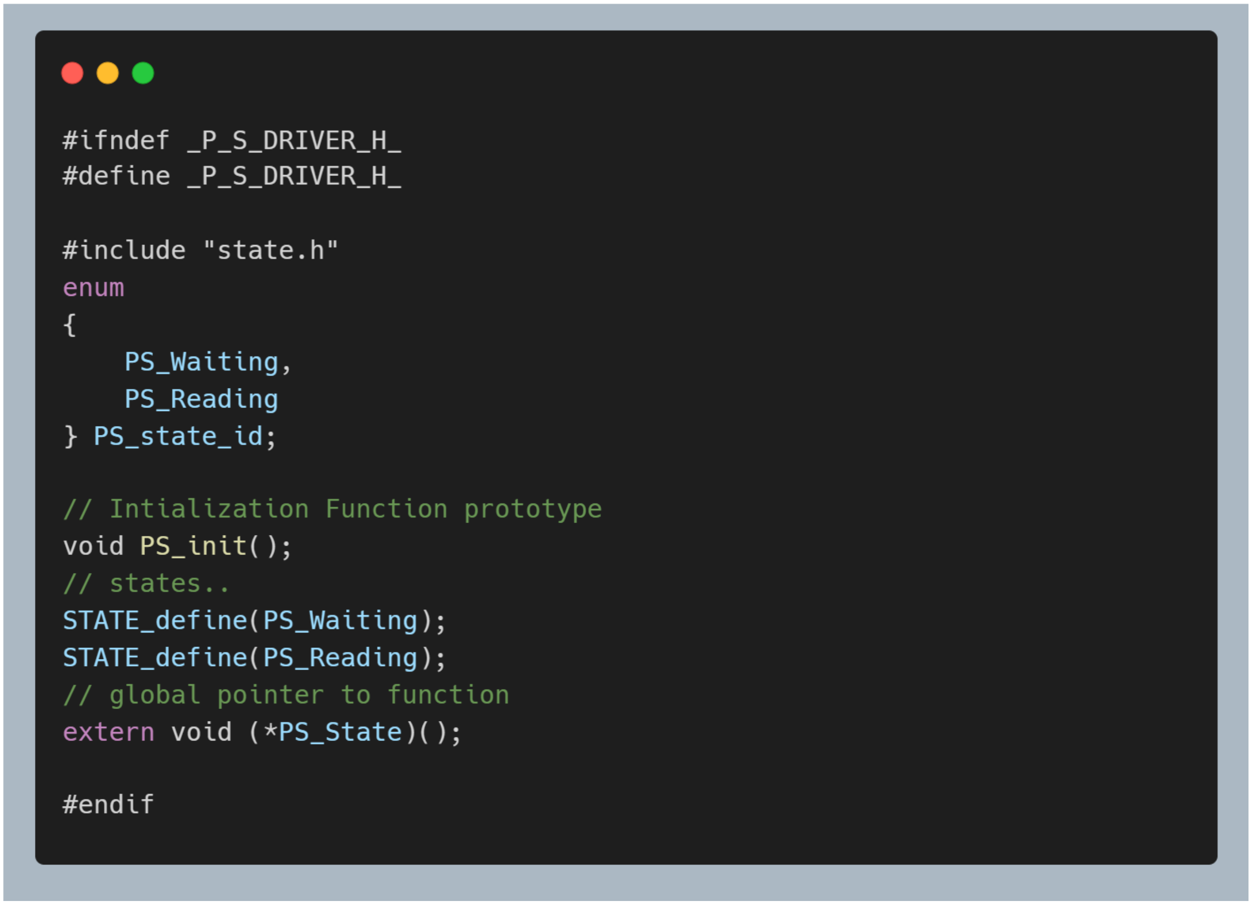
1. Delay function to be used as a delay for a few seconds.
2. Set alarm actuator to set the led or reset it according to the pressure value.
3. Gpio Initialization to initialize the gpio of the board.
4. Get pressure value to get the value of the pressure according to the input.



```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;
}
```

- Pressure sensor driver.

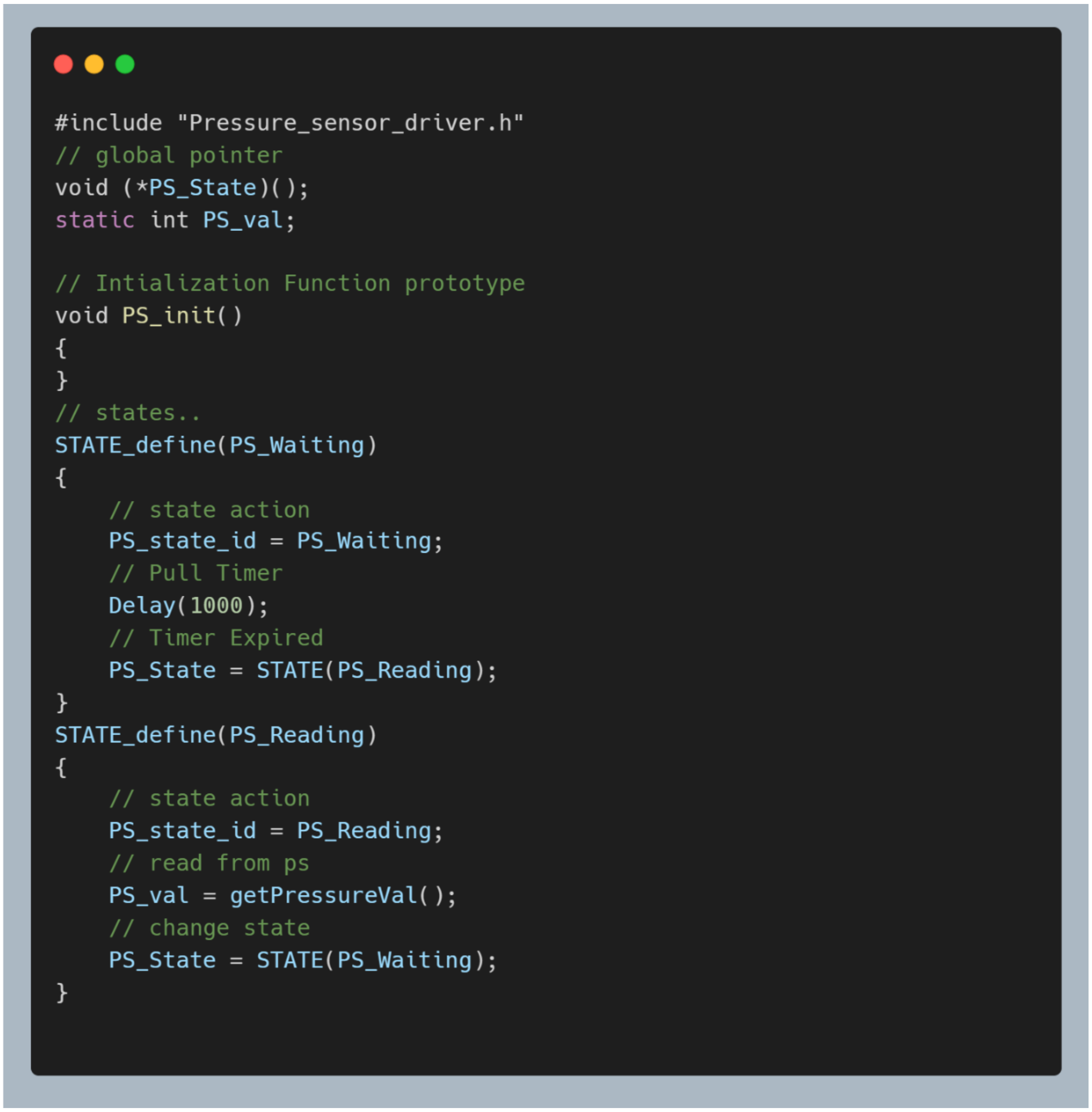
- Pressure_sensor.h



```c
#ifndef _P_S_DRIVER_H_
#define _P_S_DRIVER_H_

#include "state.h"
enum
{
    PS_Waiting,
    PS_Reading
} PS_state_id;

// Intialization Function prototype
void PS_init();
// states..
STATE_define(PS_Waiting);
STATE_define(PS_Reading);
// global pointer to function
extern void (*PS_State)();

#endif
```

- Pressure_sensor.c

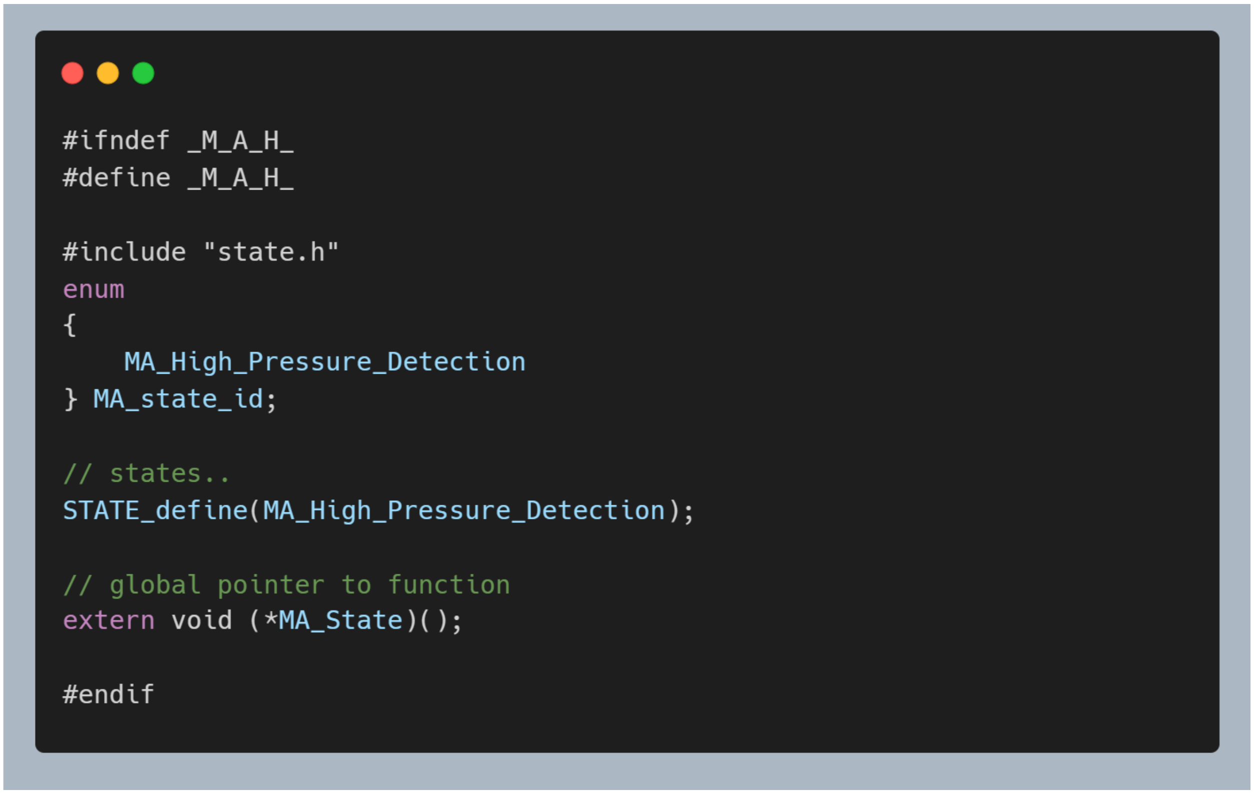
   It contains the state machines definitions.



```c
#include "Pressure_sensor_driver.h"
// global pointer
void (*PS_State)();
static int PS_val;

// Intialization Function prototype
void PS_init()
{
}
// states..
STATE_define(PS_Waiting)
{
    // state action
    PS_state_id = PS_Waiting;
    // Pull Timer
    Delay(1000);
    // Timer Expired
    PS_State = STATE(PS_Reading);
}
STATE_define(PS_Reading)
{
    // state action
    PS_state_id = PS_Reading;
    // read from ps
    PS_val = getPressureVal();
    // change state
    PS_State = STATE(PS_Waiting);
}
```

- Main Algorithm.

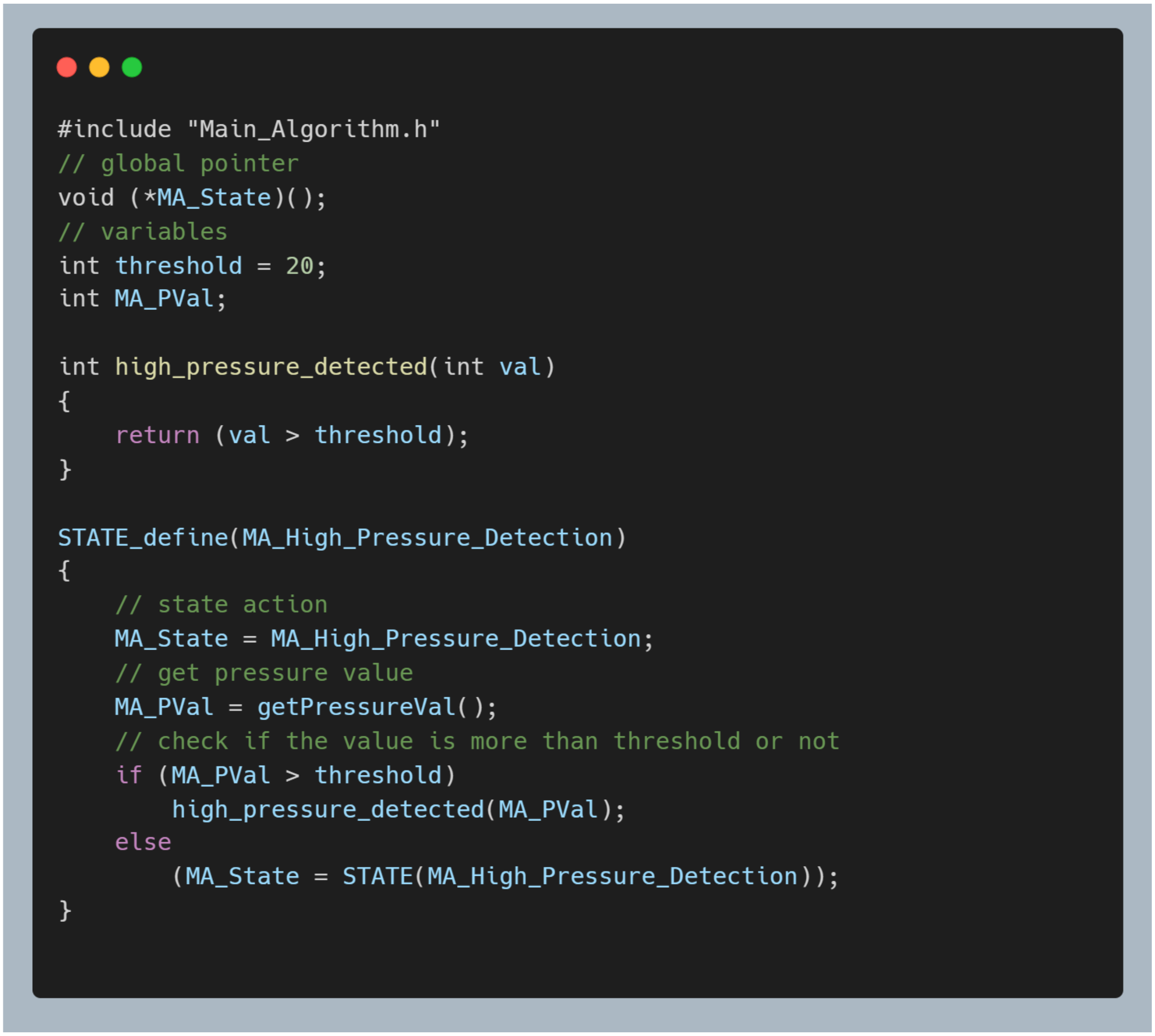
- Main_Algorithm.h



```c
#ifndef _M_A_H_
#define _M_A_H_

#include "state.h"
enum
{
    MA_High_Pressure_Detection
} MA_state_id;

// states..
STATE_define(MA_High_Pressure_Detection);

// global pointer to function
extern void (*MA_State)();

#endif
```

- Main_Algorithm.c

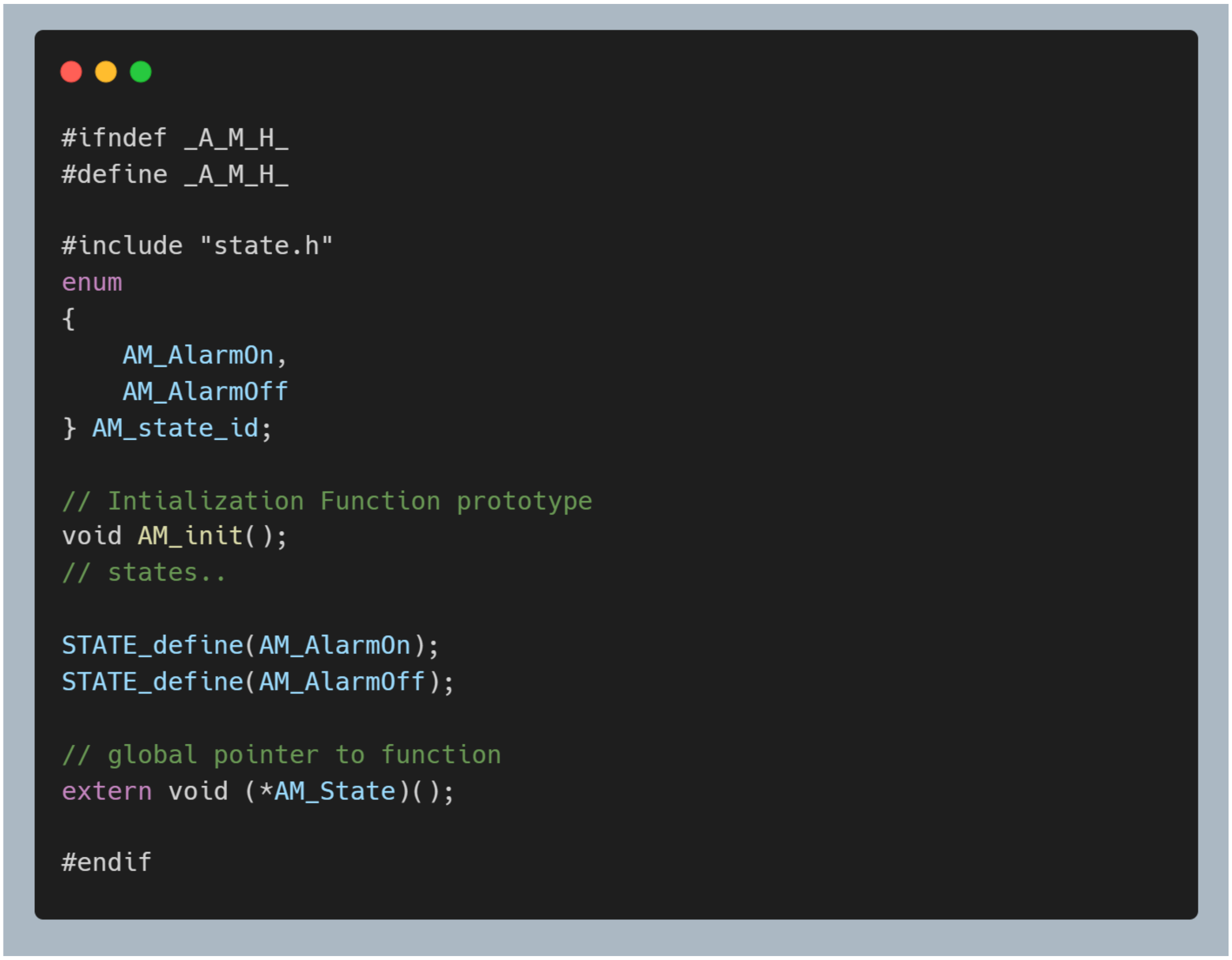
It includes the Implementation of high pressure detected function and the state that checks for high pressure.



```c
#include "Main_Algorithm.h"
// global pointer
void (*MA_State)();
// variables
int threshold = 20;
int MA_PVal;

int high_pressure_detected(int val)
{
    return (val > threshold);
}

STATE_define(MA_High_Pressure_Detection)
{
    // state action
    MA_State = MA_High_Pressure_Detection;
    // get pressure value
    MA_PVal = getPressureVal();
    // check if the value is more than threshold or not
    if (MA_PVal > threshold)
        high_pressure_detected(MA_PVal);
    else
        (MA_State = STATE(MA_High_Pressure_Detection));
}
```

- Alarm Monitor.

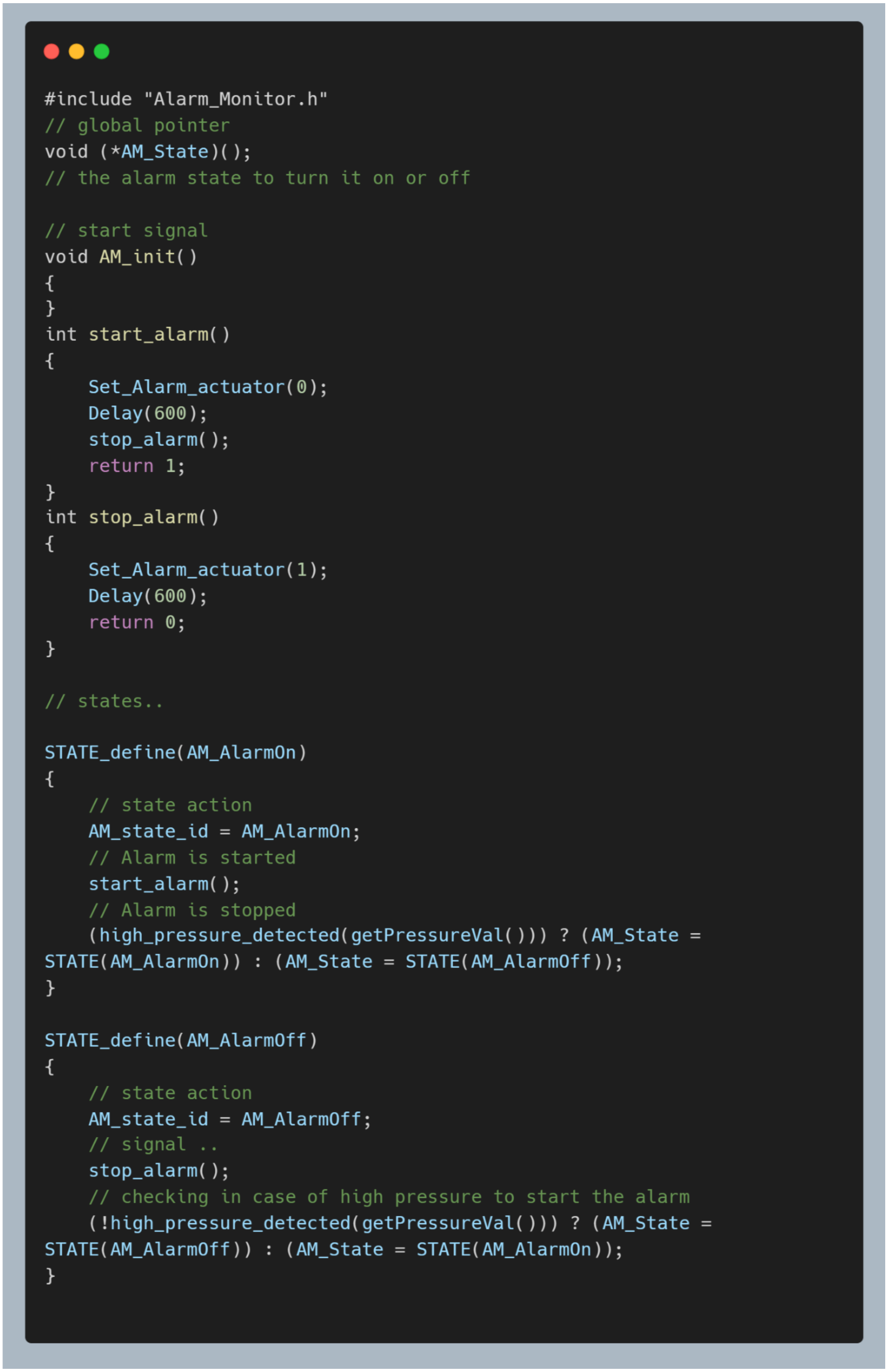
- Alarm_Monitor.h



```c
#ifndef _A_M_H_
#define _A_M_H_

#include "state.h"
enum
{
    AM_AlarmOn,
    AM_AlarmOff
} AM_state_id;

// Intialization Function prototype
void AM_init();
// states..

STATE_define(AM_AlarmOn);
STATE_define(AM_AlarmOff);

// global pointer to function
extern void (*AM_State)();

#endif
```

- Alarm_Monitor.c

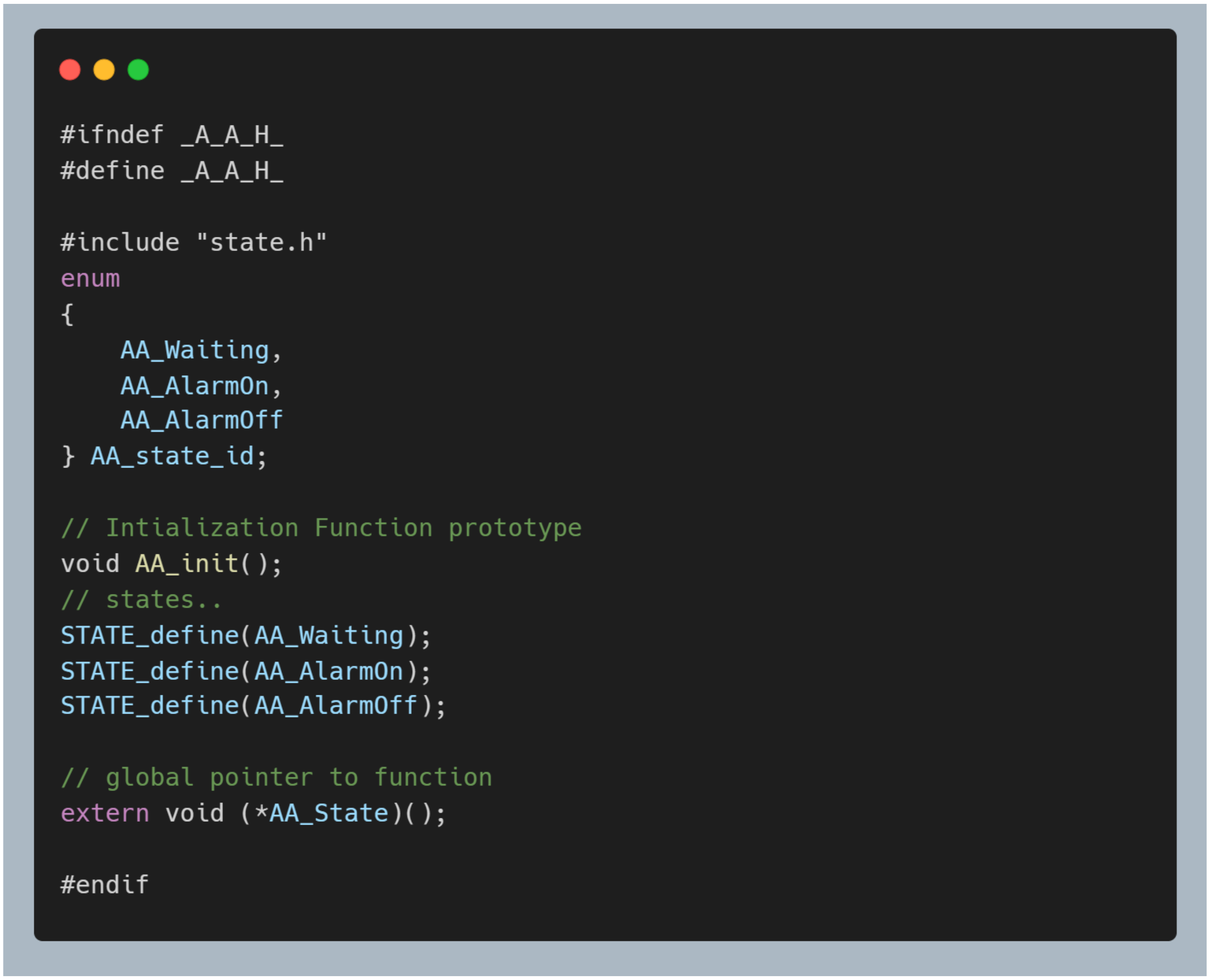
It implements the start alarm function where we set the led to work for a duration of 60 seconds and stop alarm function which stops the led again.



```
#include "Alarm_Monitor.h"
// global pointer
void (*AM_State)();
// the alarm state to turn it on or off

// start signal
void AM_init()
{
}
int start_alarm()
{
    Set_Alarm_actuator(0);
    Delay(600);
    stop_alarm();
    return 1;
}
int stop_alarm()
{
    Set_Alarm_actuator(1);
    Delay(600);
    return 0;
}

// states..

STATE_define(AM_AlarmOn)
{
    // state action
    AM_state_id = AM_AlarmOn;
    // Alarm is started
    start_alarm();
    // Alarm is stopped
    (high_pressure_detected(getPressureVal())) ? (AM_State =
STATE(AM_AlarmOn)) : (AM_State = STATE(AM_AlarmOff));
}

STATE_define(AM_AlarmOff)
{
    // state action
    AM_state_id = AM_AlarmOff;
    // signal ..
    stop_alarm();
    // checking in case of high pressure to start the alarm
    (!high_pressure_detected(getPressureVal())) ? (AM_State =
STATE(AM_AlarmOff)) : (AM_State = STATE(AM_AlarmOn));
}
```

- Alarm Actuator Driver

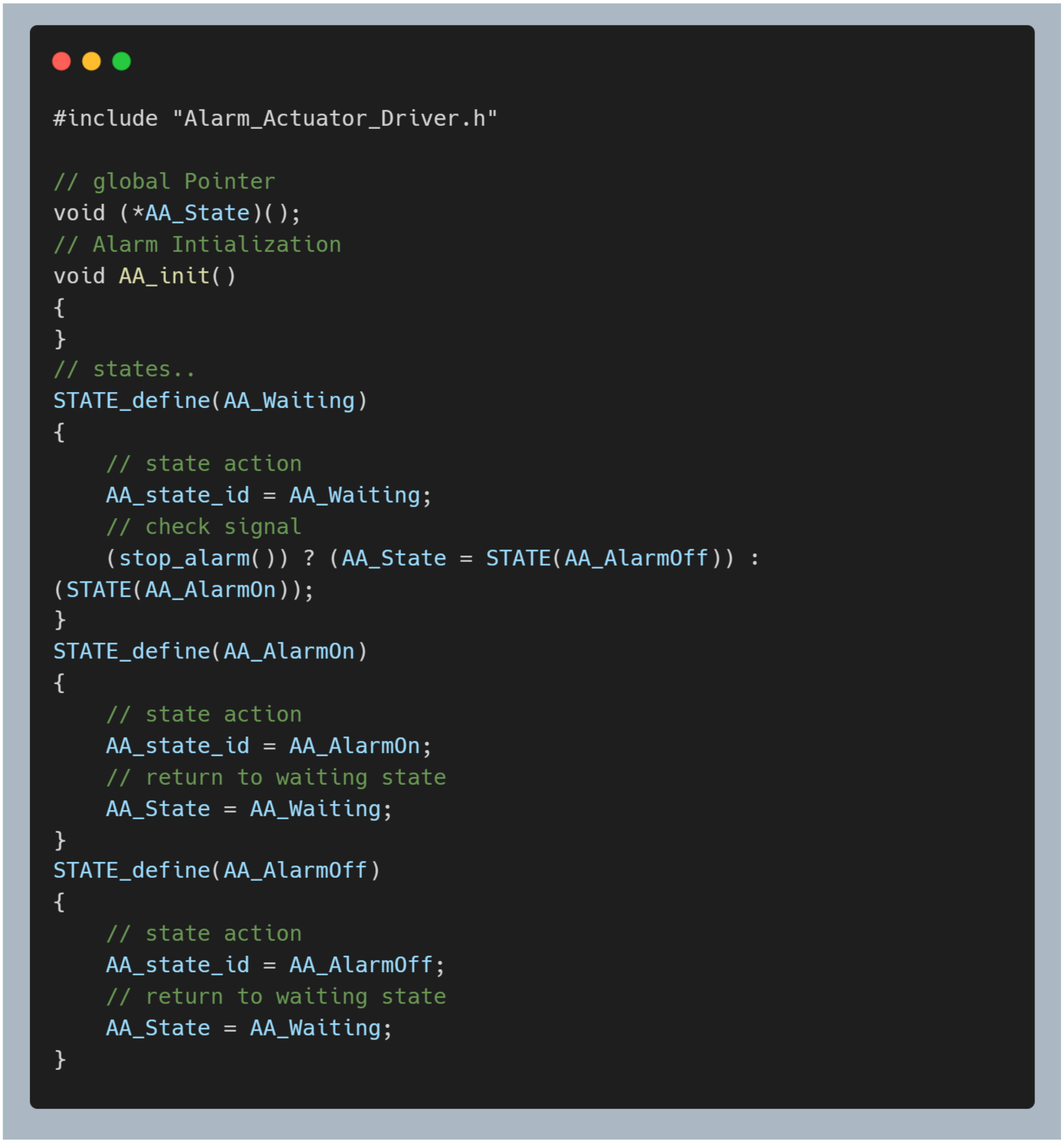
- Alarm_Actuator.h



```
#ifndef _A_A_H_
#define _A_A_H_

#include "state.h"
enum
{
    AA_Waiting,
    AA_AlarmOn,
    AA_AlarmOff
} AA_state_id;

// Intialization Function prototype
void AA_init();
// states..
STATE_define(AA_Waiting);
STATE_define(AA_AlarmOn);
STATE_define(AA_AlarmOff);

// global pointer to function
extern void (*AA_State)();

#endif
```

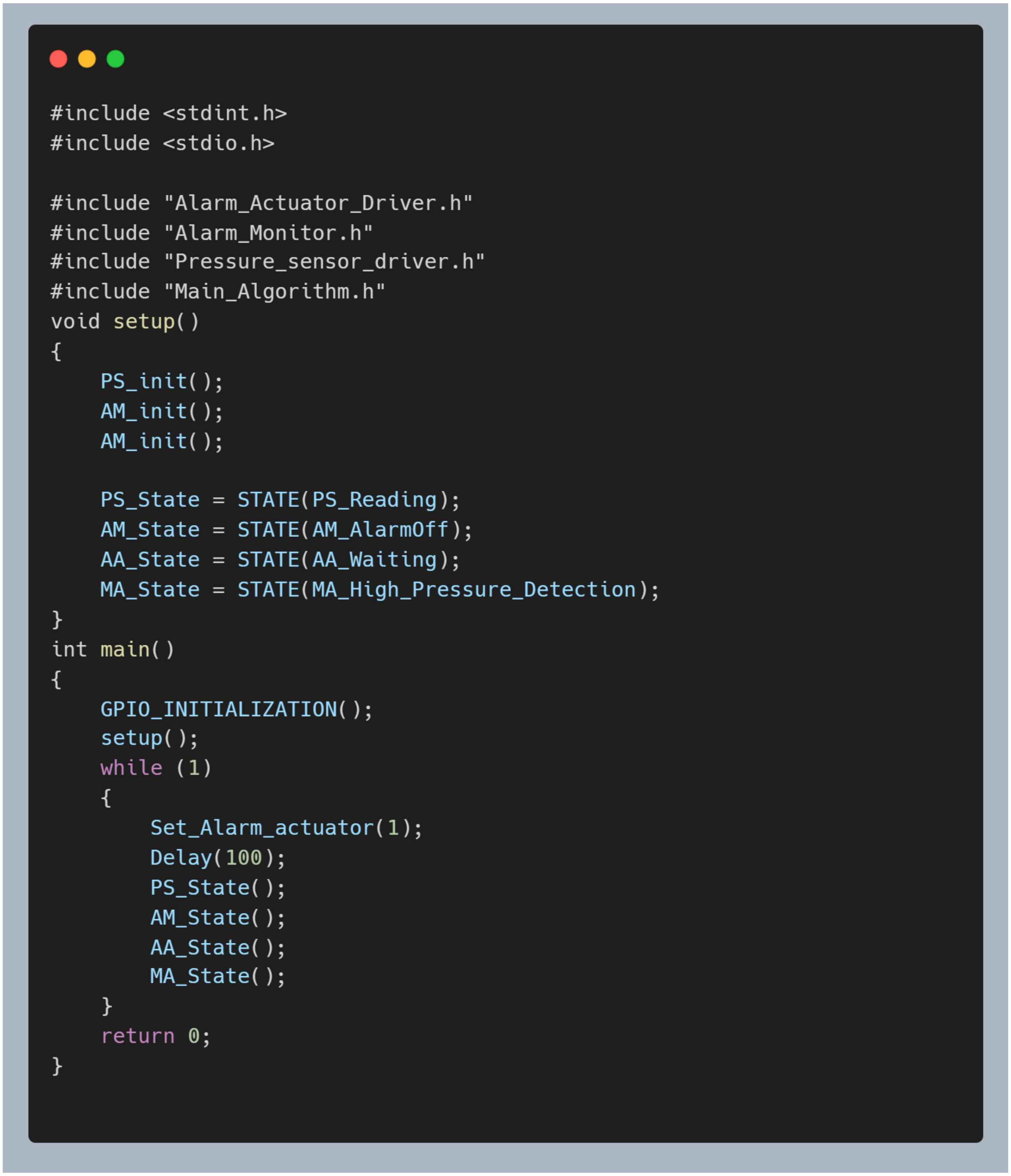
- Alarm_Actuator.c

It implements the checking of the signal so it can stay in alarm off or go to alarm on.



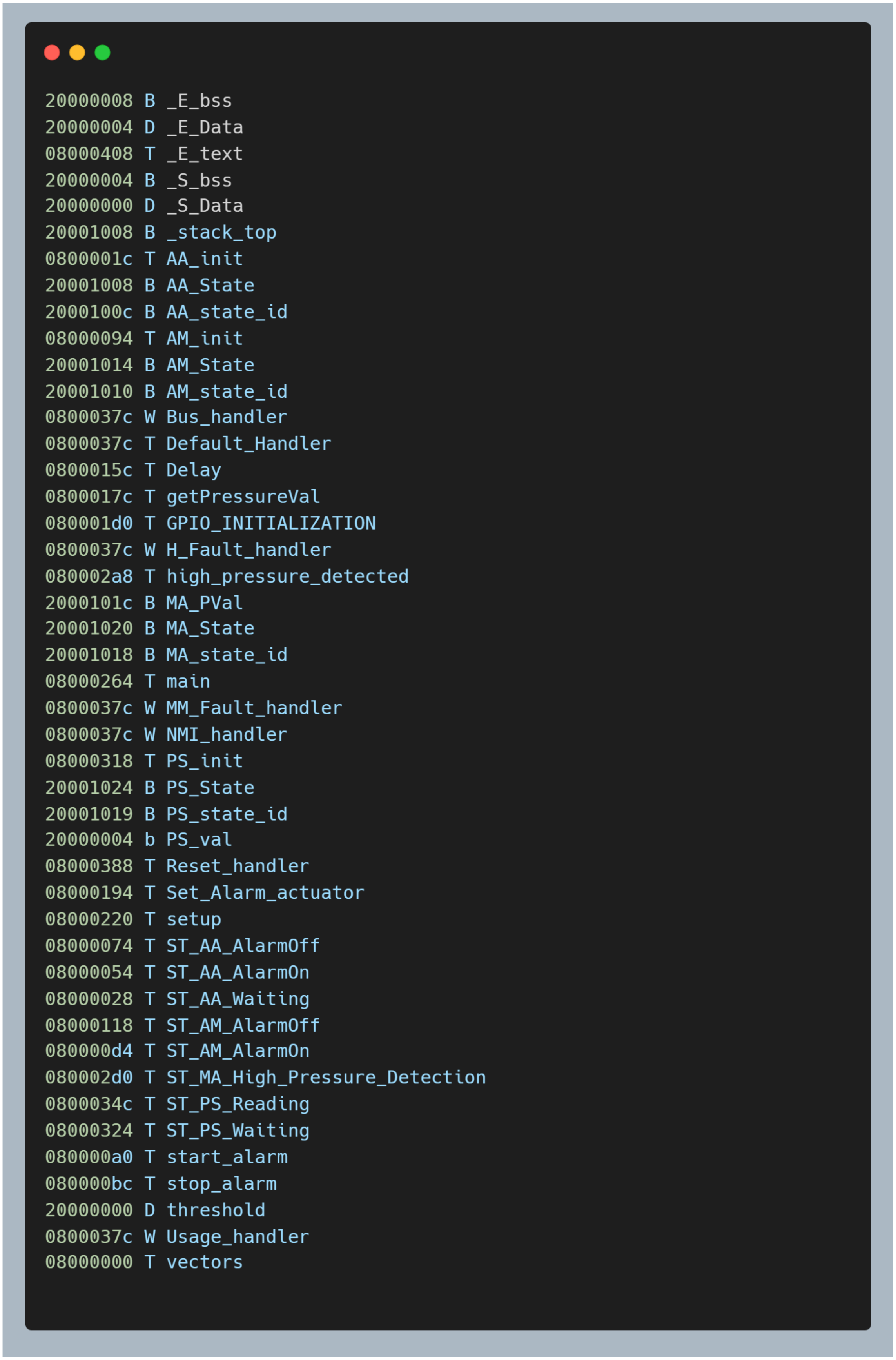
```
#include "Alarm_Actuator_Driver.h"

// global Pointer
void (*AA_State)();
// Alarm Intialization
void AA_init()
{
}
// states..
STATE_define(AA_Waiting)
{
    // state action
    AA_state_id = AA_Waiting;
    // check signal
    (stop_alarm()) ? (AA_State = STATE(AA_AlarmOff)) :
(STATE(AA_AlarmOn));
}
STATE_define(AA_AlarmOn)
{
    // state action
    AA_state_id = AA_AlarmOn;
    // return to waiting state
    AA_State = AA_Waiting;
}
STATE_define(AA_AlarmOff)
{
    // state action
    AA_state_id = AA_AlarmOff;
    // return to waiting state
    AA_State = AA_Waiting;
}
```

- Main



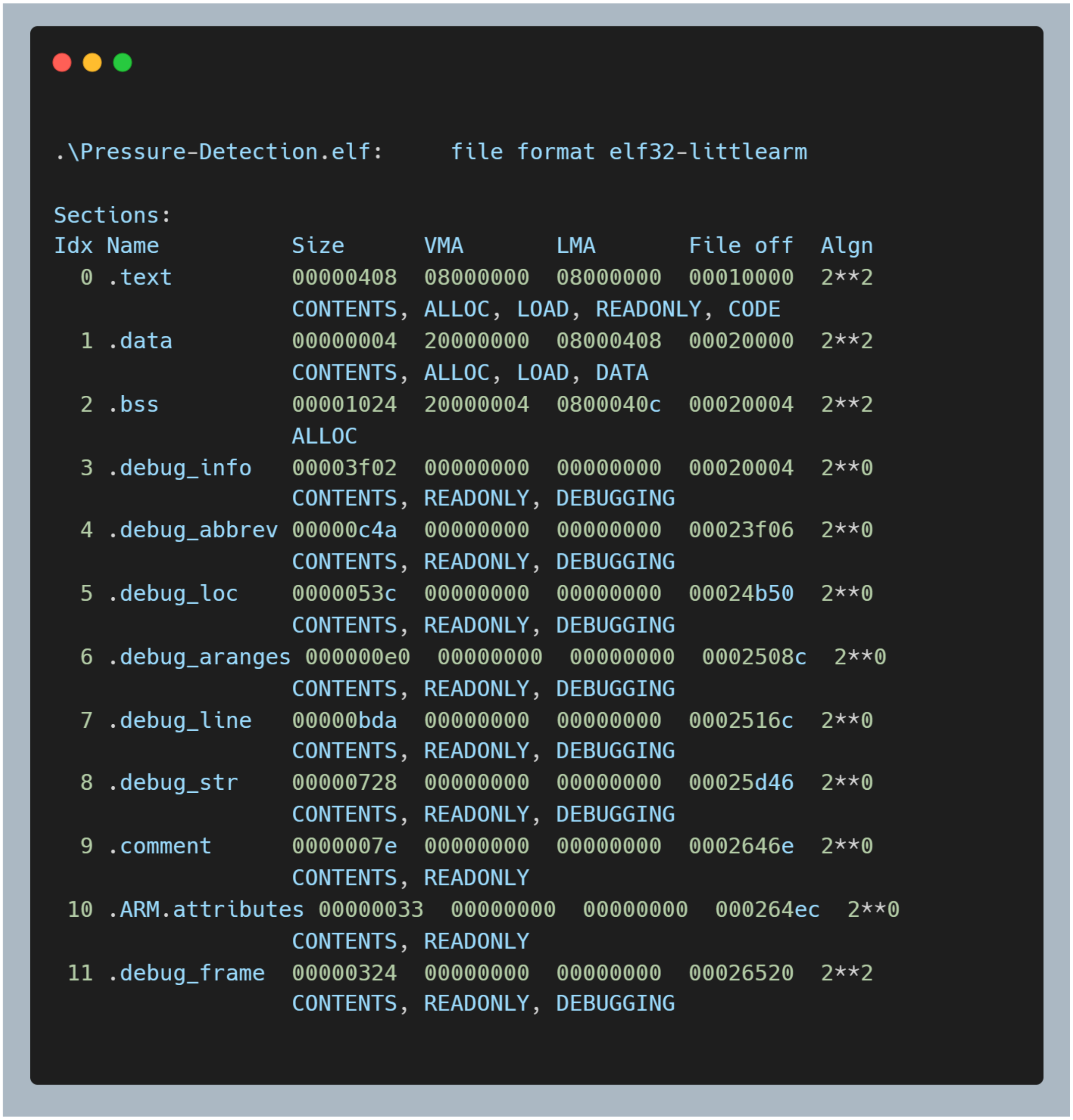
```c
#include <stdint.h>
#include <stdio.h>

#include "Alarm_Actuator_Driver.h"
#include "Alarm_Monitor.h"
#include "Pressure_sensor_driver.h"
#include "Main_Algorithm.h"
void setup()
{
    PS_init();
    AM_init();
    AM_init();

    PS_State = STATE(PS_Reading);
    AM_State = STATE(AM_AlarmOff);
    AA_State = STATE(AA_Waiting);
    MA_State = STATE(MA_High_Pressure_Detection);
}
int main()
{
    GPIO_INITIALIZATION();
    setup();
    while (1)
    {
        Set_Alarm_actuator(1);
        Delay(100);
        PS_State();
        AM_State();
        AA_State();
        MA_State();
    }
    return 0;
}
```

6.Symbol Table.



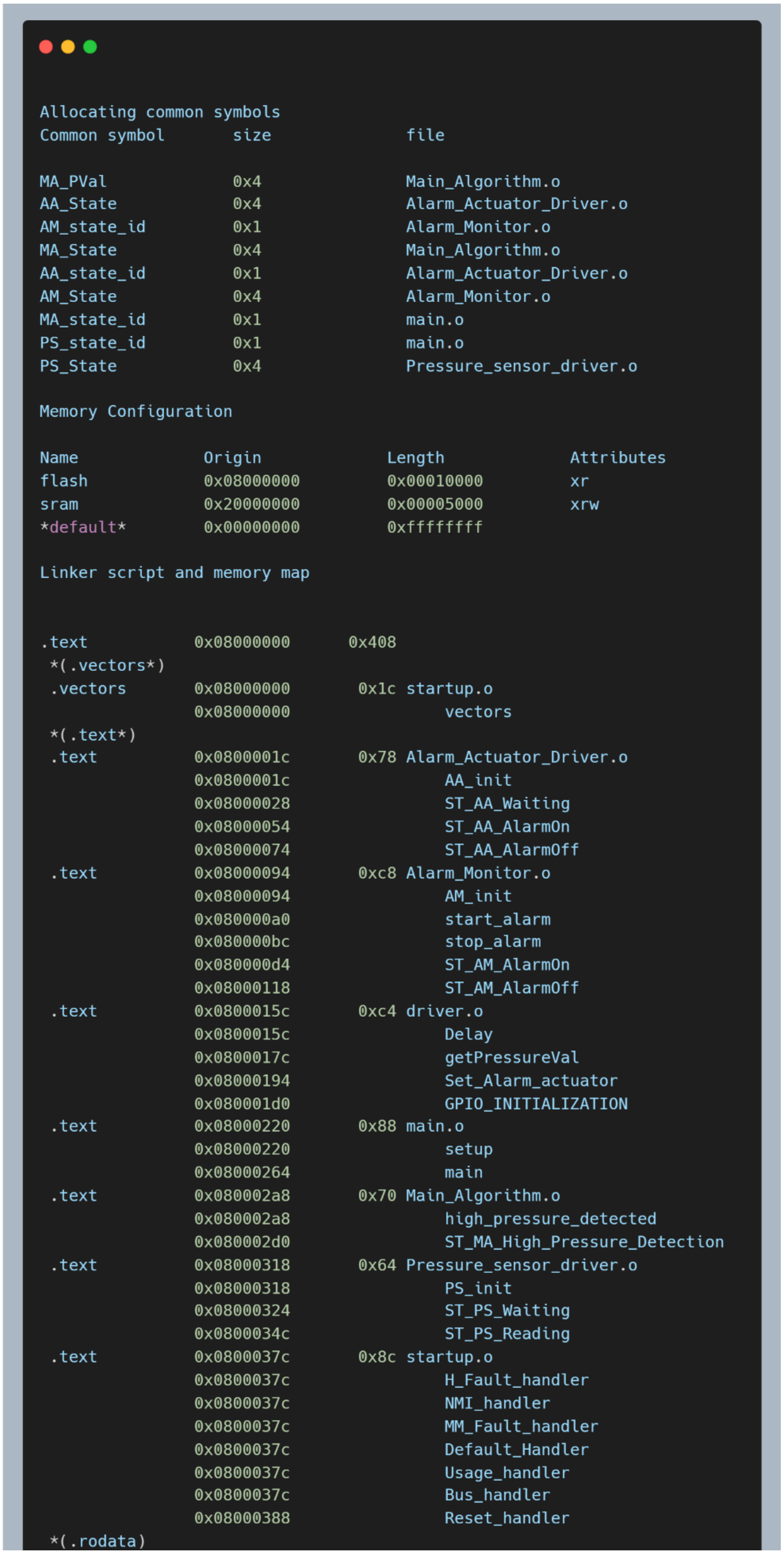
```
20000008 B _E_bss
20000004 D _E_Data
08000408 T _E_text
20000004 B _S_bss
20000000 D _S_Data
20001008 B _stack_top
0800001c T AA_init
20001008 B AA_State
2000100c B AA_state_id
08000094 T AM_init
20001014 B AM_State
20001010 B AM_state_id
0800037c W Bus_handler
0800037c T Default_Handler
0800015c T Delay
0800017c T getPressureVal
080001d0 T GPIO_INITIALIZATION
0800037c W H_Fault_handler
080002a8 T high_pressure_detected
2000101c B MA_PVal
20001020 B MA_State
20001018 B MA_state_id
08000264 T main
0800037c W MM_Fault_handler
0800037c W NMI_handler
08000318 T PS_init
20001024 B PS_State
20001019 B PS_state_id
20000004 b PS_val
08000388 T Reset_handler
08000194 T Set_Alarm_actuator
08000220 T setup
08000074 T ST_AA_AlarmOff
08000054 T ST_AA_AlarmOn
08000028 T ST_AA_Waiting
08000118 T ST_AM_AlarmOff
080000d4 T ST_AM_AlarmOn
080002d0 T ST_MA_High_Pressure_Detection
0800034c T ST_PS_Reading
08000324 T ST_PS_Waiting
080000a0 T start_alarm
080000bc T stop_alarm
20000000 D threshold
0800037c W Usage_handler
08000000 T vectors
```

7.Section Table



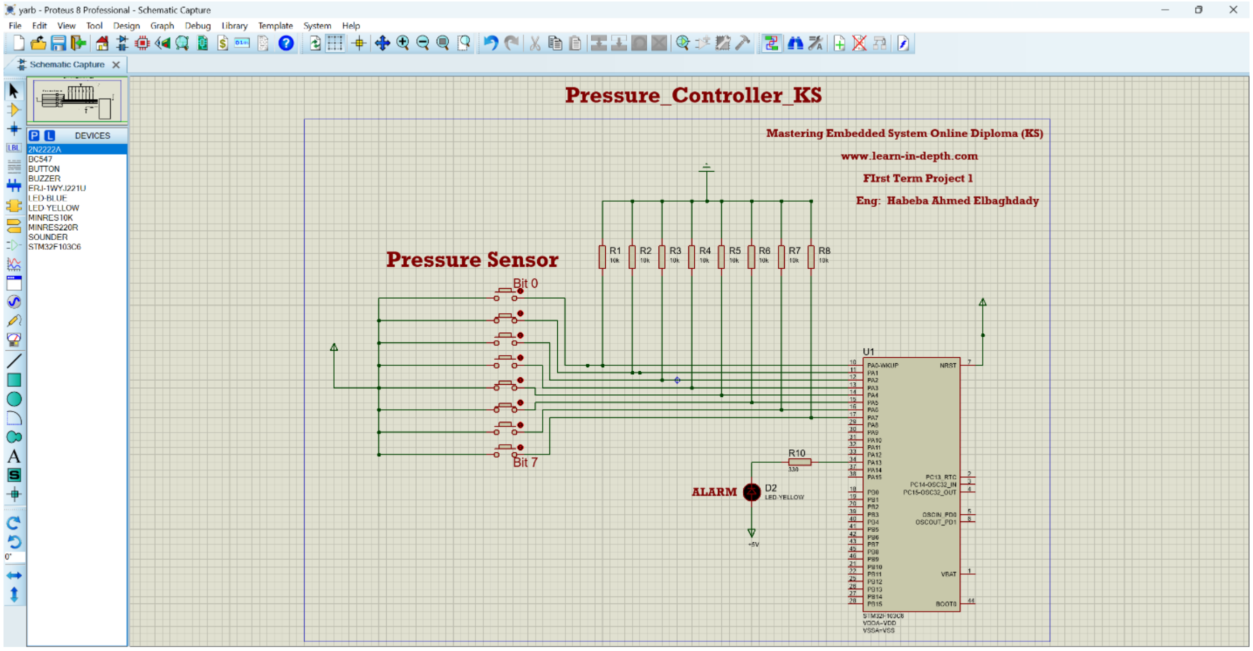
```
.\Pressure-Detection.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000408  08000000  08000000  00010000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000004  20000000  08000408  00020000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00001024  20000004  0800040c  00020004  2**2
                  ALLOC
  3 .debug_info   00003f02  00000000  00000000  00020004  2**0
                  CONTENTS, READONLY, DEBUGGING
  4 .debug_abbrev 00000c4a  00000000  00000000  00023f06  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    0000053c  00000000  00000000  00024b50  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 000000e0  00000000  00000000  0002508c  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_line   00000bda  00000000  00000000  0002516c  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .debug_str    00000728  00000000  00000000  00025d46  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      0000007e  00000000  00000000  0002646e  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  000264ec  2**0
                  CONTENTS, READONLY
 11 .debug_frame  00000324  00000000  00000000  00026520  2**2
                  CONTENTS, READONLY, DEBUGGING
```

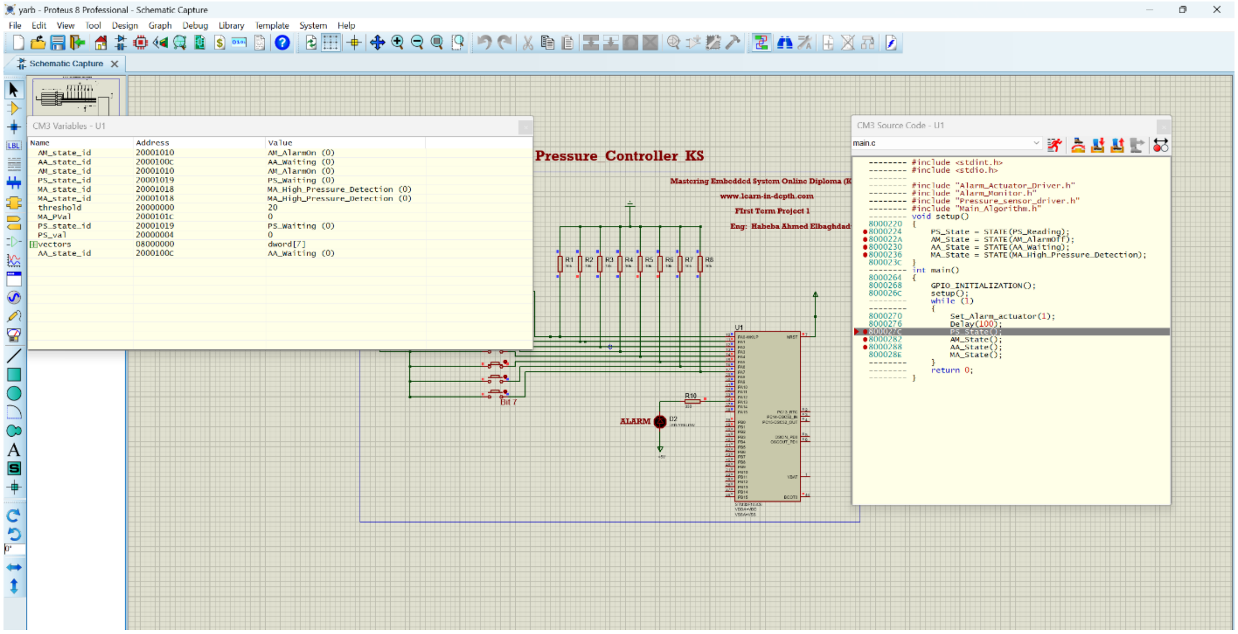
8.Map file.



```
Allocating common symbols
Common symbol       size              file

MA_PVal             0x4               Main_Algorithm.o
AA_State            0x4               Alarm_Actuator_Driver.o
AM_state_id         0x1               Alarm_Monitor.o
MA_State            0x4               Main_Algorithm.o
AA_state_id         0x1               Alarm_Actuator_Driver.o
AM_State            0x4               Alarm_Monitor.o
MA_state_id         0x1               main.o
PS_state_id         0x1               main.o
PS_State            0x4               Pressure_sensor_driver.o

Memory Configuration

Name             Origin             Length             Attributes
flash            0x08000000         0x00010000         xr
sram             0x20000000         0x00005000         xrw
*default*        0x00000000         0xffffffff

Linker script and memory map


.text           0x08000000      0x408
 *(.vectors*)
 .vectors       0x08000000       0x1c startup.o
                0x08000000           vectors
 *(.text*)
 .text          0x0800001c       0x78 Alarm_Actuator_Driver.o
                0x0800001c           AA_init
                0x08000028           ST_AA_Waiting
                0x08000054           ST_AA_AlarmOn
                0x08000074           ST_AA_AlarmOff
 .text          0x08000094       0xc8 Alarm_Monitor.o
                0x08000094           AM_init
                0x080000a0           start_alarm
                0x080000bc           stop_alarm
                0x080000d4           ST_AM_AlarmOn
                0x08000118           ST_AM_AlarmOff
 .text          0x0800015c       0xc4 driver.o
                0x0800015c           Delay
                0x0800017c           getPressureVal
                0x08000194           Set_Alarm_actuator
                0x080001d0           GPIO_INITIALIZATION
 .text          0x08000220       0x88 main.o
                0x08000220           setup
                0x08000264           main
 .text          0x080002a8       0x70 Main_Algorithm.o
                0x080002a8           high_pressure_detected
                0x080002d0           ST_MA_High_Pressure_Detection
 .text          0x08000318       0x64 Pressure_sensor_driver.o
                0x08000318           PS_init
                0x08000324           ST_PS_Waiting
                0x0800034c           ST_PS_Reading
 .text          0x0800037c       0x8c startup.o
                0x0800037c           H_Fault_handler
                0x0800037c           NMI_handler
                0x0800037c           MM_Fault_handler
                0x0800037c           Default_Handler
                0x0800037c           Usage_handler
                0x0800037c           Bus_handler
                0x08000388           Reset_handler
 *(.rodata)
```

9.Hardware




yarb - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
Pressure_Controller_KS
Mastering Embedded System Online Diploma (K
www.learn-in-depth.com
FIrst Term Project 1
Eng: Habeba Ahmed Elbaghdad
ALARM

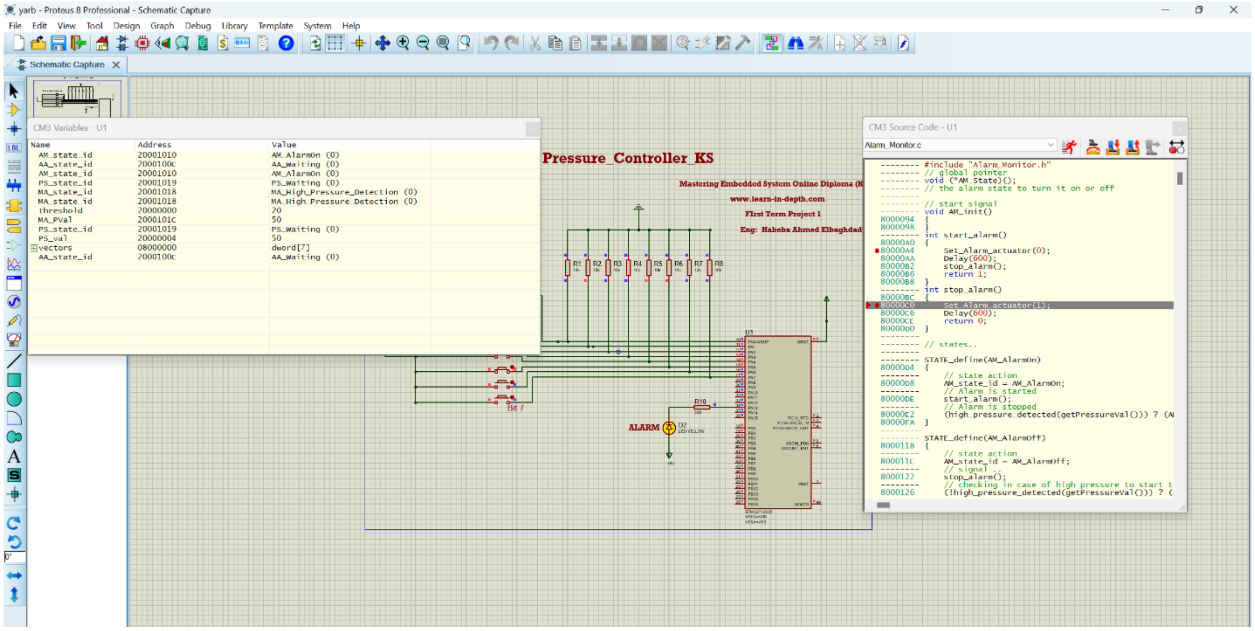
CM3 Variables - U1

Name	Address	Value
AM_state_id	20001010	AM_AlarmOn (0)
AA_state_id	2000100c	AA_Waiting (0)
AM_state_id	20001010	AM_AlarmOn (0)
PS_state_id	20001019	PS_Waiting (0)
MA_state_id	20001018	MA_High_Pressure_Detection (0)
MA_state_id	20001018	MA_High_Pressure_Detection (0)
threshold	20000000	20
MA_PVal	2000101c	0
PS_state_id	20001019	PS_Waiting (0)
PS_val	20000004	0
⊞vectors	08000000	dword[7]
AA_state_id	2000100c	AA_Waiting (0)

CM3 Source Code - U1

main.c

```
-------- #include <stdint.h>
-------- #include <stdio.h>
--------
-------- #include "Alarm_Actuator_Driver.h"
-------- #include "Alarm_Monitor.h"
-------- #include "Pressure_sensor_driver.h"
-------- #include "Main_Algorithm.h"
-------- void setup()
8000220  {
●8000224      PS_State = STATE(PS_Reading);
●800022A      AM_State = STATE(AM_AlarmOff);
●8000230      AA_State = STATE(AA_Waiting);
●8000236      MA_State = STATE(MA_High_Pressure_Detection);
800023C  }
-------- int main()
8000264  {
8000268      GPIO_INITIALIZATION();
800026C      setup();
--------     while (1)
--------     {
8000270          Set_Alarm_actuator(1);
8000276          Delay(100);
▶●800027C          PS_State();
●8000282          AM_State();
●8000288          AA_State();
800028E          MA_State();
--------     }
--------     return 0;
-------- }
```


yarb - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
Pressure_Controller_KS
Mastering Embedded System Online Diploma (K
www.learn-in-depth.com
First Term Project 1
Eng: Habeba Ahmed Elbaghdad
R1 R2 R3 R4 R5 R6 R7 R8
R10
ALARM
D2
LED-YELLOW
U1
Bit 7

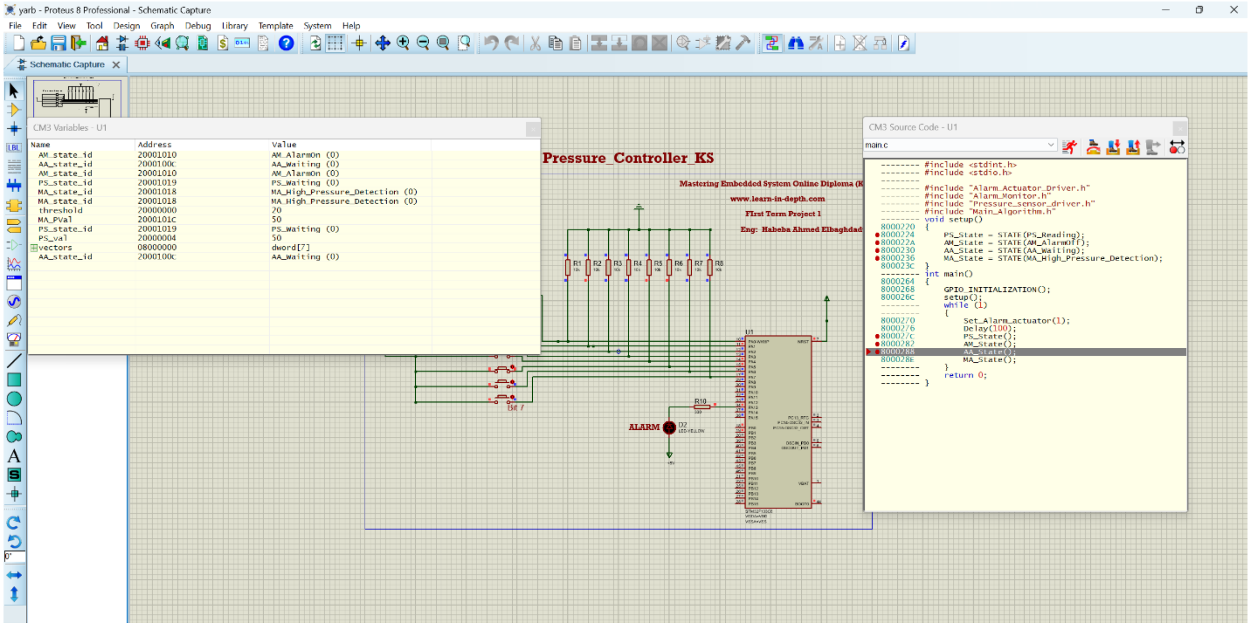
CM3 Variables - U1

Name	Address	Value
AM_state_id	20001010	AM_AlarmOn (0)
AA_state_id	2000100c	AA_Waiting (0)
AM_state_id	20001010	AM_AlarmOn (0)
PS_state_id	20001019	PS_Waiting (0)
MA_state_id	20001018	MA_High_Pressure_Detection (0)
MA_state_id	20001018	MA_High_Pressure_Detection (0)
threshold	20000000	20
MA_PVal	2000101c	50
PS_state_id	20001019	PS_Waiting (0)
PS_val	20000004	50
⊞vectors	08000000	dword[7]
AA_state_id	2000100c	AA_Waiting (0)

CM3 Source Code - U1

Alarm_Monitor.c

```
 -------- #include "Alarm_Monitor.h"
 -------- // global pointer
 -------- void (*AM_State)();
 -------- // the alarm state to turn it on or off
 --------
 -------- // start signal
 -------- void AM_init()
 8000094  {
 8000098  }
 -------- int start_alarm()
 80000A0  {
●80000A4      Set_Alarm_actuator(0);
 80000AA      Delay(600);
 80000B2      stop_alarm();
 80000B6      return 1;
 80000B8  }
 -------- int stop_alarm()
 80000BC  {
▶●80000C0     Set_Alarm_actuator(1);
 80000C6      Delay(600);
 80000CE      return 0;
 80000D0  }
 --------
 -------- // states..
 --------
 -------- STATE_define(AM_AlarmOn)
 80000D4  {
 --------     // state action
 80000D8      AM_state_id = AM_AlarmOn;
 --------     // Alarm is started
 80000DE      start_alarm();
 --------     // Alarm is stopped
 80000E2      (high_pressure_detected(getPressureVal())) ? (A
 80000FA  }
 --------
 -------- STATE_define(AM_AlarmOff)
 8000118  {
 --------     // state action
 800011c      AM_state_id = AM_AlarmOff;
 --------     // signal ..
 8000122      stop_alarm();
 --------     // checking in case of high pressure to start t
 8000126      (!high_pressure_detected(getPressureVal())) ? (
```


yarb - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
Pressure_Controller_KS
Mastering Embedded System Online Diploma (K
www.learn-in-depth.com
First Term Project 1
Eng: Habeba Ahmed Elbaghdad
R1 R2 R3 R4 R5 R6 R7 R8
R10
ALARM
D2
LED-YELLOW
U1
Bit 7

CM3 Variables - U1

Name	Address	Value
AM_state_id	20001010	AM_AlarmOn (0)
AA_state_id	2000100c	AA_Waiting (0)
AM_state_id	20001010	AM_AlarmOn (0)
PS_state_id	20001019	PS_Waiting (0)
MA_state_id	20001018	MA_High_Pressure_Detection (0)
MA_state_id	20001018	MA_High_Pressure_Detection (0)
threshold	20000000	20
MA_PVal	2000101c	50
PS_state_id	20001019	PS_Waiting (0)
PS_val	20000004	50
⊞vectors	08000000	dword[7]
AA_state_id	2000100c	AA_Waiting (0)

CM3 Source Code - U1

main.c

```
 -------- #include <stdint.h>
 -------- #include <stdio.h>
 --------
 -------- #include "Alarm_Actuator_Driver.h"
 -------- #include "Alarm_Monitor.h"
 -------- #include "Pressure_sensor_driver.h"
 -------- #include "Main_Algorithm.h"
 -------- void setup()
 8000220  {
●8000224      PS_State = STATE(PS_Reading);
●800022A      AM_State = STATE(AM_AlarmOff);
●8000230      AA_State = STATE(AA_Waiting);
●8000236      MA_State = STATE(MA_High_Pressure_Detection);
 800023c  }
 -------- int main()
 8000264  {
 8000268      GPIO_INITIALIZATION();
 800026c      setup();
 --------     while (1)
 --------     {
 8000270          Set_Alarm_actuator(1);
 8000276          Delay(100);
●800027c          PS_State();
●8000282          AM_State();
▶●8000288         AA_State();
 800028E          MA_State();
 --------     }
 --------     return 0;
 -------- }
```