Mastering Embedded System Online Diploma

Embedded System | Learn-IN-Depth

Name: **Ibrahim Shokry Ibrahim.**

Topic: **First term (Final project 1).**

Profile: ibrahimshokry98@gmail.com (learn-in-depth.com)

- 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.

Assumptions:

1- System set up and shut down procedures are not modeled.

2- System maintenance is not modeled.

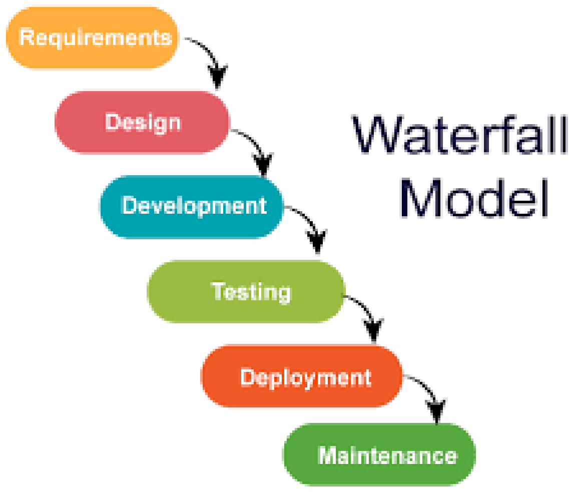
3- Pressure Sensor never fails.

4- The alarm never fails.

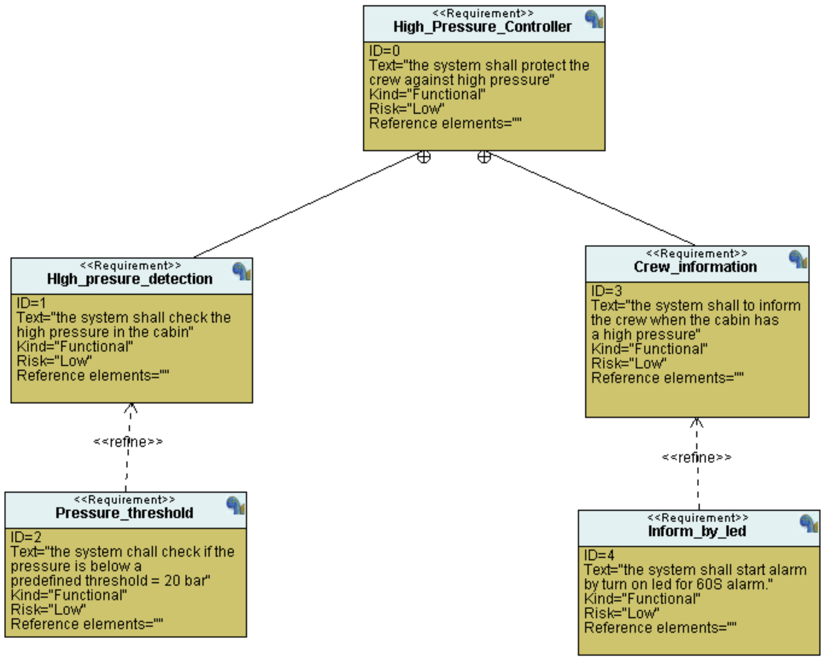
5- The system never faces cut off.

- Methodology:

Since the requirements are clear and will unlikely change, the system will use a straight-forward predictive model like the waterfall model. Every step will be taken sequentially and since the system is very simple, the implementation phase will take a very short time and we will have enough time for the testing phase.



- Requirements:


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

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

<<refine>>

<<Requirement>>
Pressure_threshold
ID=2
Text="the system chall check if the pressure is below a predefined threshold = 20 bar"
Kind="Functional"
Risk="Low"
Reference elements=""

<<Requirement>>
Crew_information
ID=3
Text="the system shall to inform the crew when the cabin has a high pressure"
Kind="Functional"
Risk="Low"
Reference elements=""

<<refine>>

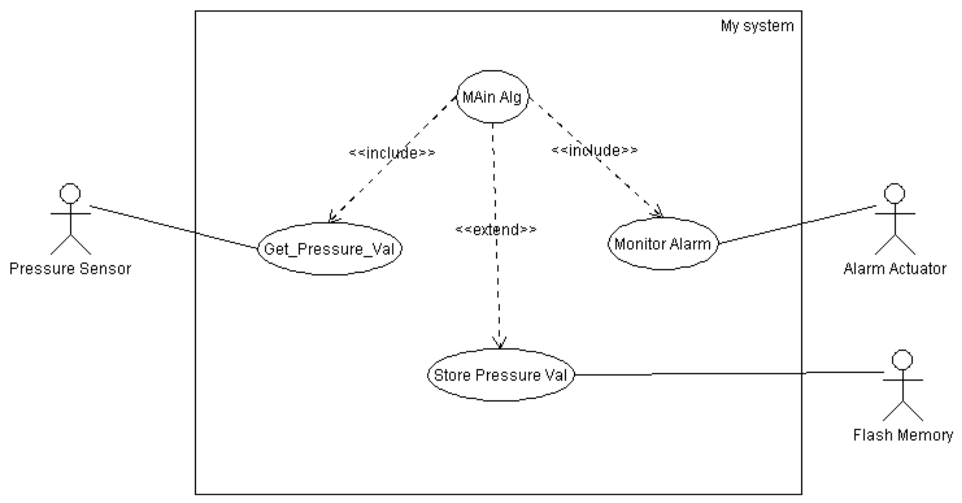
<<Requirement>>
Inform_by_led
ID=4
Text="the system shall start alarm by turn on led for 60S alarm."
Kind="Functional"
Risk="Low"
Reference elements=""

- Space exploration/partitioning:

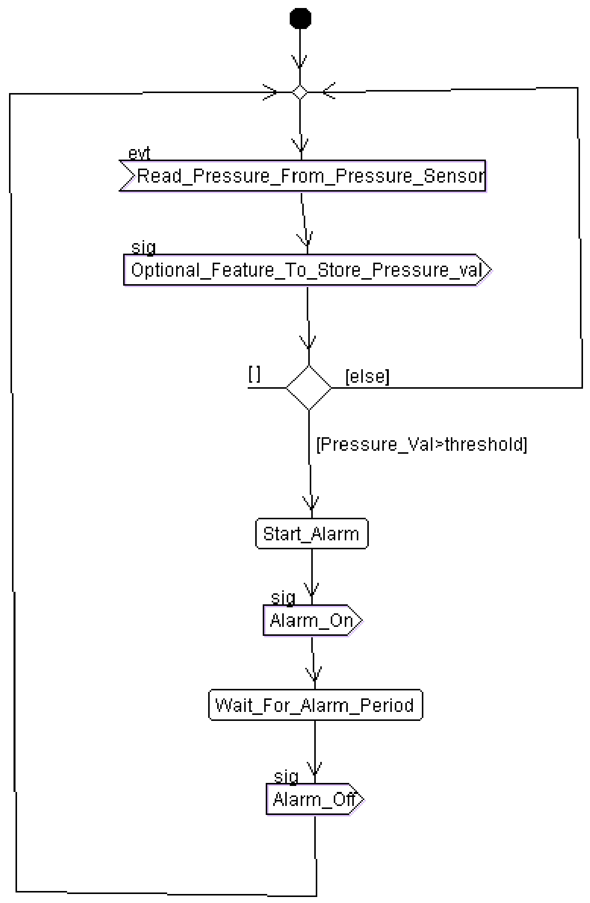
For the hardware, we have stm32 microcontroller with ARM Cortex M3 that will be more than enough for this application.

- System analysis:

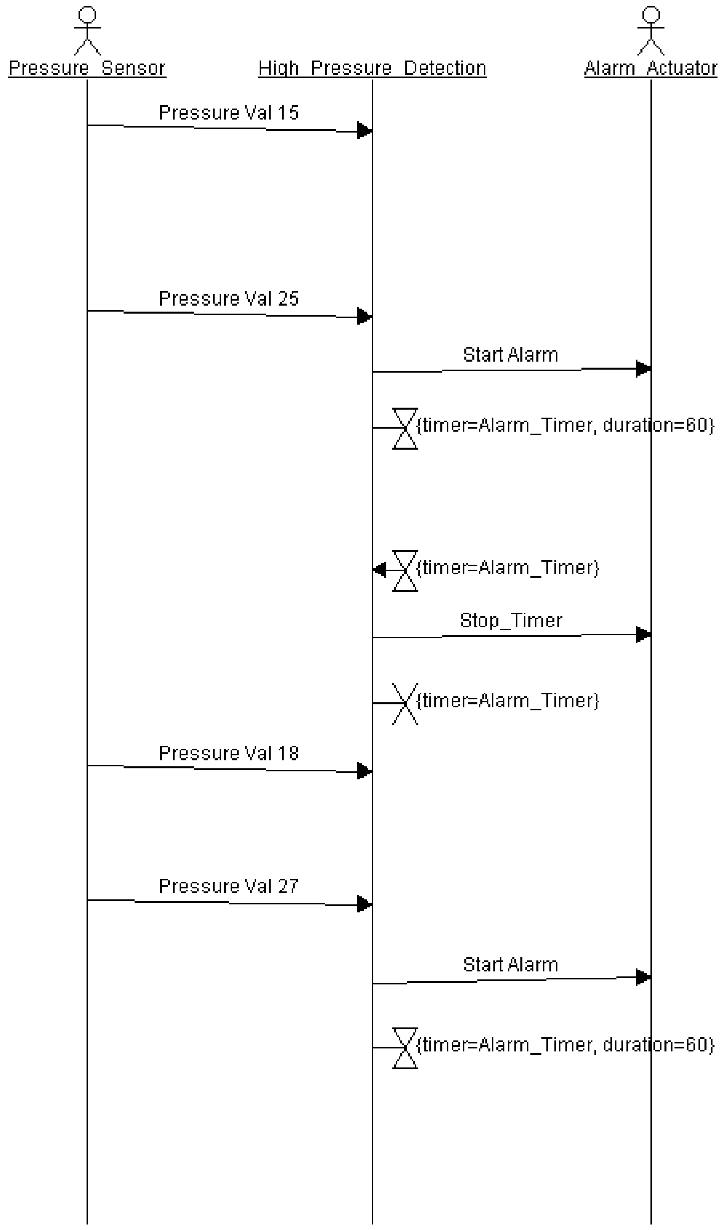
- Use case diagram:



- Activity diagram:

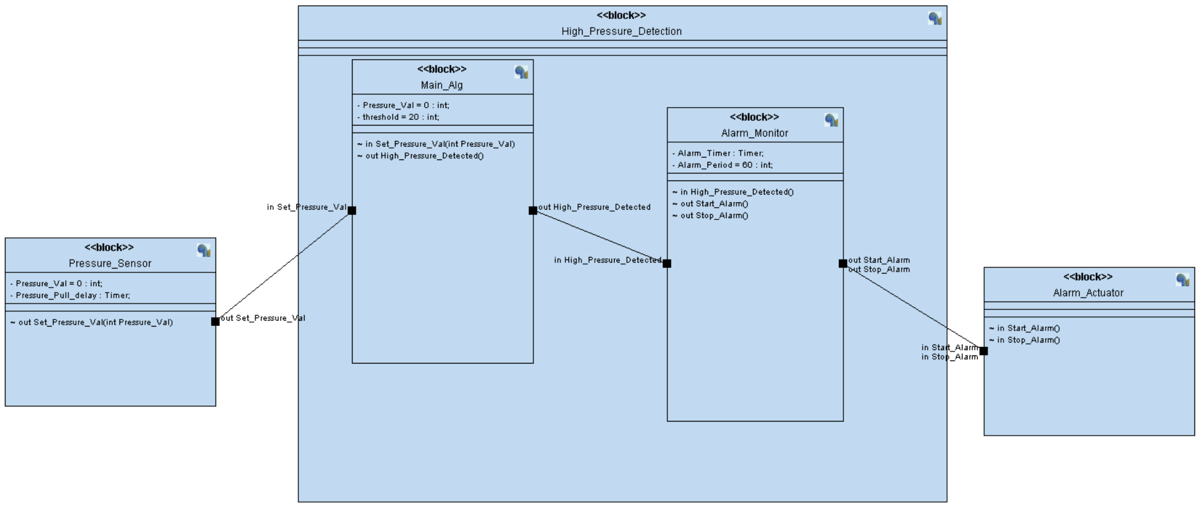


- **Sequence diagram:**

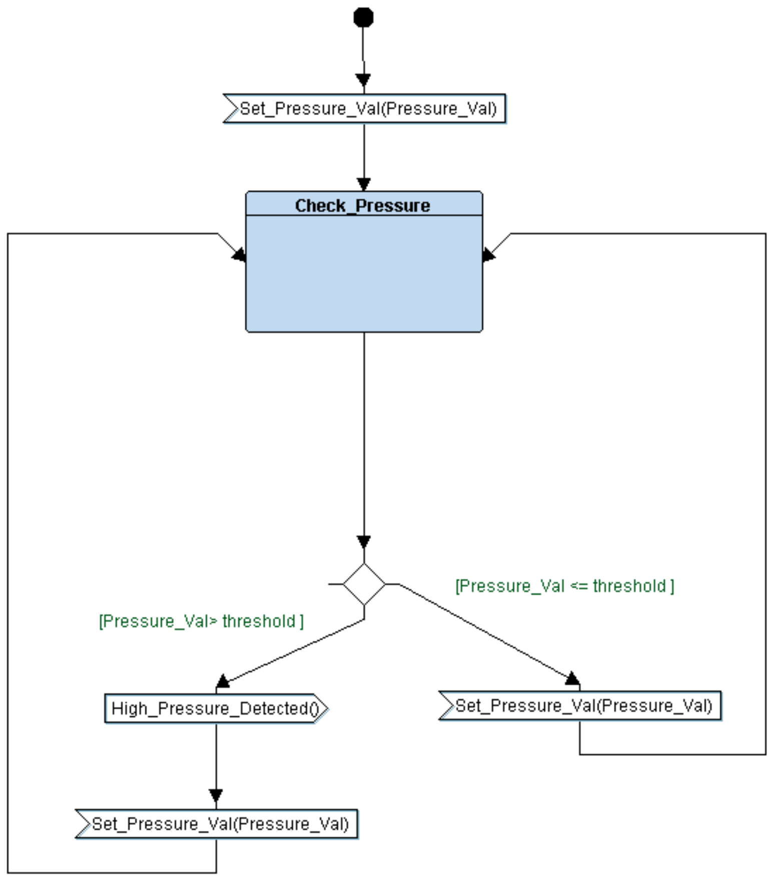


- System design:

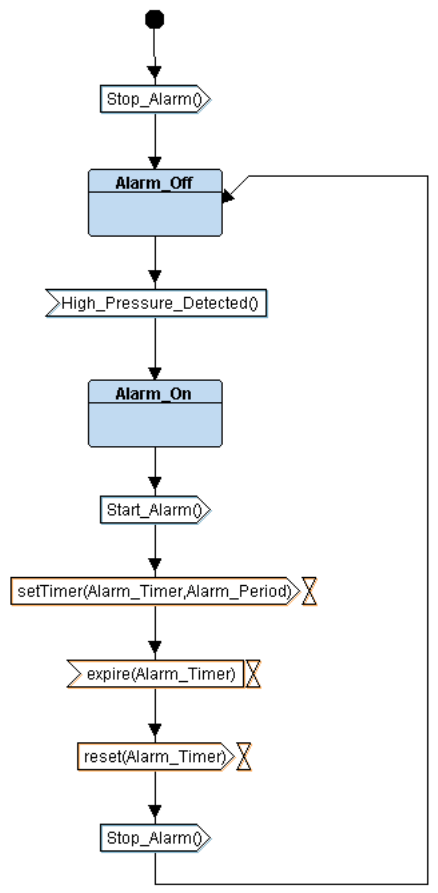
- Block diagram:



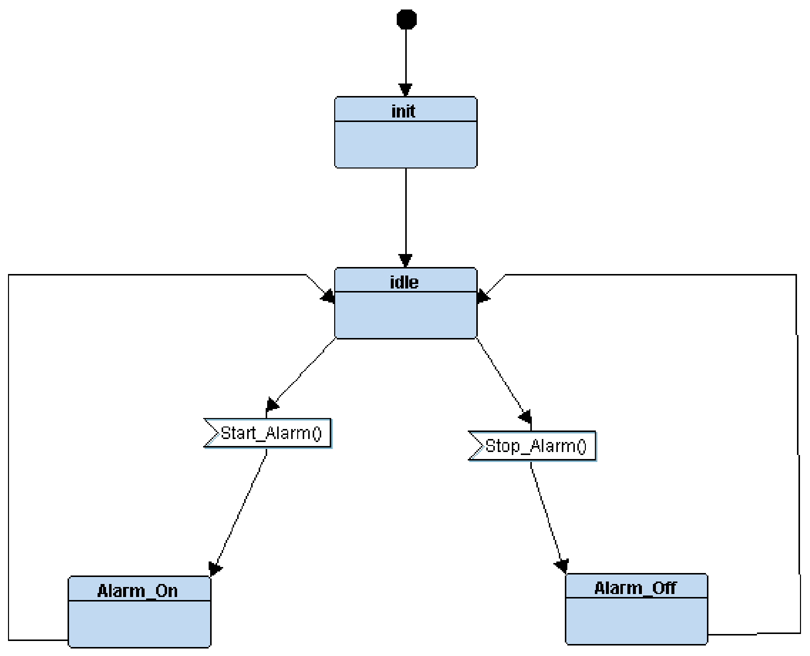
- Main Algorism” state machine:


Set_Pressure_Val(Pressure_Val)
Check_Pressure
[Pressure_Val> threshold]
[Pressure_Val <= threshold]
High_Pressure_Detected()
Set_Pressure_Val(Pressure_Val)
Set_Pressure_Val(Pressure_Val)

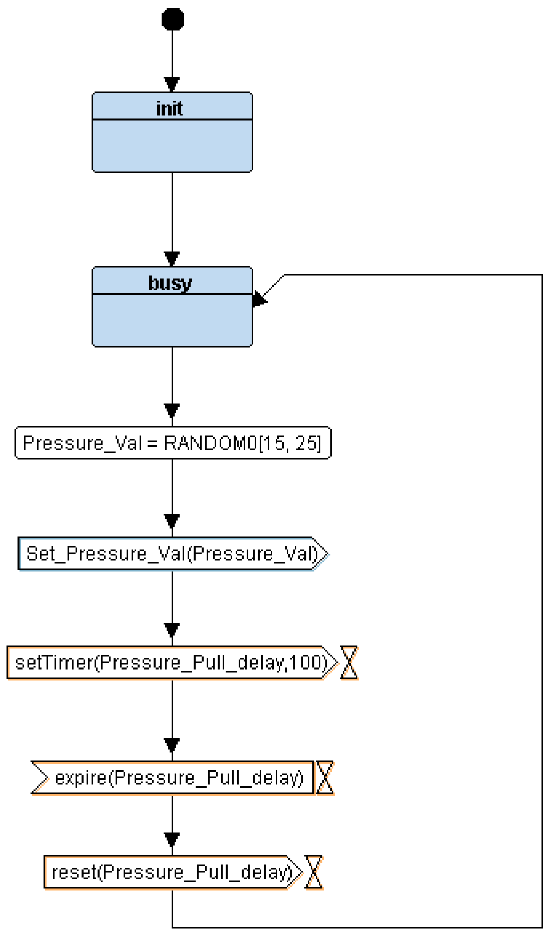
- Alarm monitor state machine:



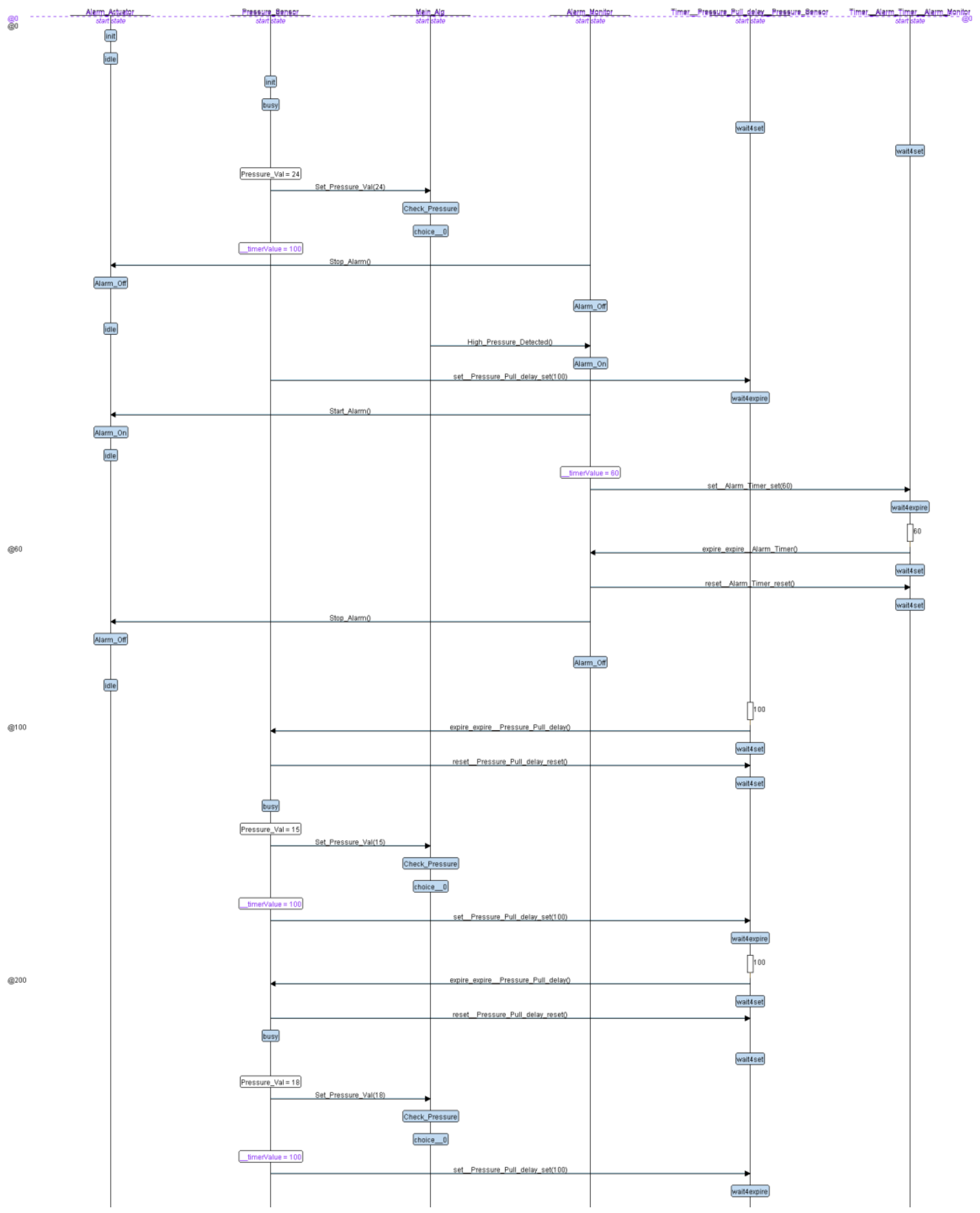
- Alarm actuator state machine:



- Pressure sensor state machine:

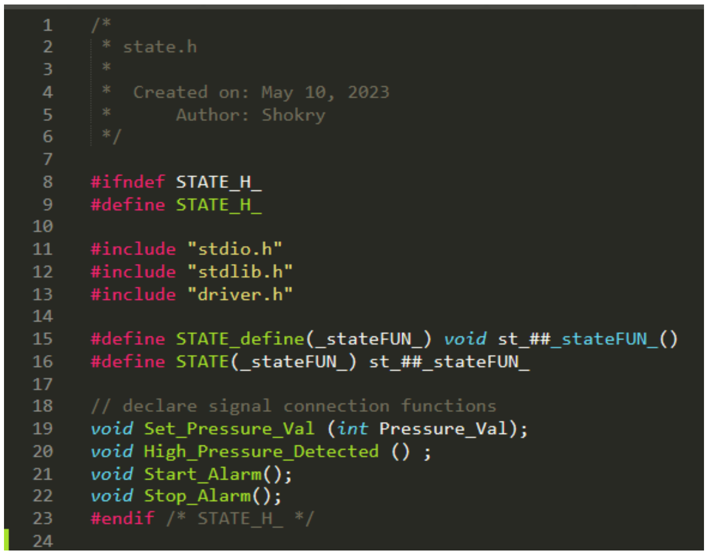

init
busy
Pressure_Val = RANDOM0[15, 25]
Set_Pressure_Val(Pressure_Val)
setTimer(Pressure_Pull_delay,100)
expire(Pressure_Pull_delay)
reset(Pressure_Pull_delay)

- System design simulation:

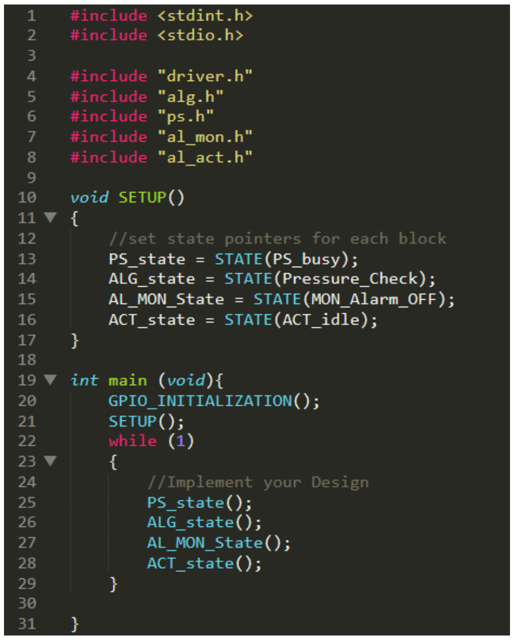


- Codes

- main.c & state.h

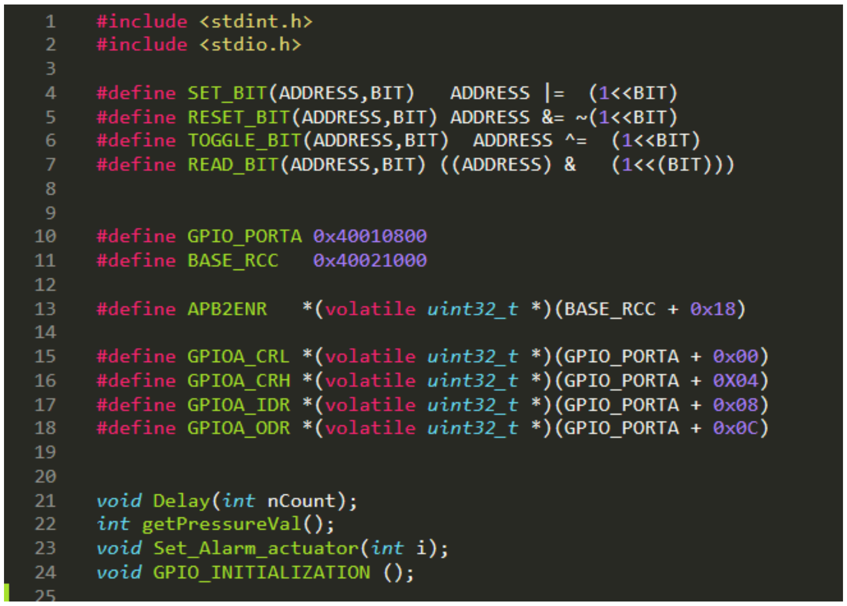


```c
/*
 * state.h
 *
 *  Created on: May 10, 2023
 *      Author: Shokry
 */

#ifndef STATE_H_
#define STATE_H_

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

#define STATE_define(_stateFUN_) void st_##_stateFUN_()
#define STATE(_stateFUN_) st_##_stateFUN_

// declare signal connection functions
void Set_Pressure_Val (int Pressure_Val);
void High_Pressure_Detected () ;
void Start_Alarm();
void Stop_Alarm();
#endif /* STATE_H_ */
```

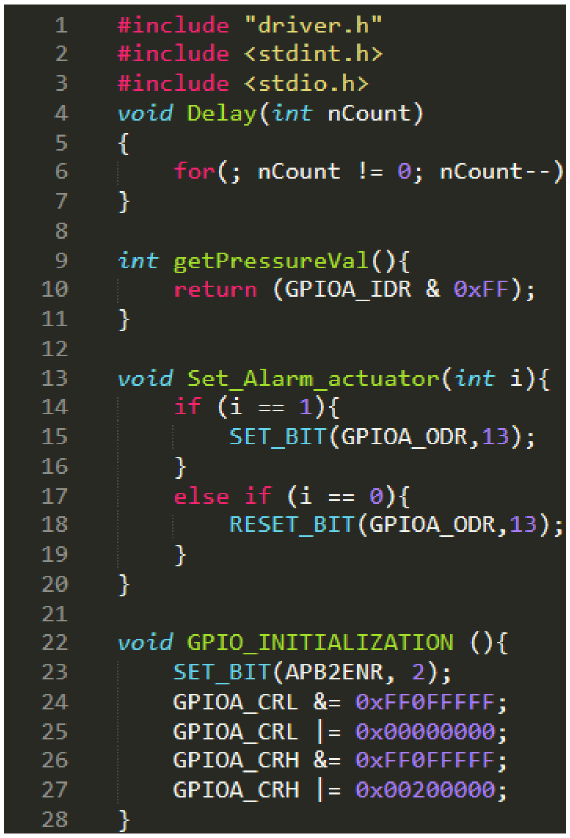


```c
#include <stdint.h>
#include <stdio.h>

#include "driver.h"
#include "alg.h"
#include "ps.h"
#include "al_mon.h"
#include "al_act.h"

void SETUP()
{
    //set state pointers for each block
    PS_state = STATE(PS_busy);
    ALG_state = STATE(Pressure_Check);
    AL_MON_State = STATE(MON_Alarm_OFF);
    ACT_state = STATE(ACT_idle);
}

int main (void){
    GPIO_INITIALIZATION();
    SETUP();
    while (1)
    {
        //Implement your Design
        PS_state();
        ALG_state();
        AL_MON_State();
        ACT_state();
    }

}
```

- driver

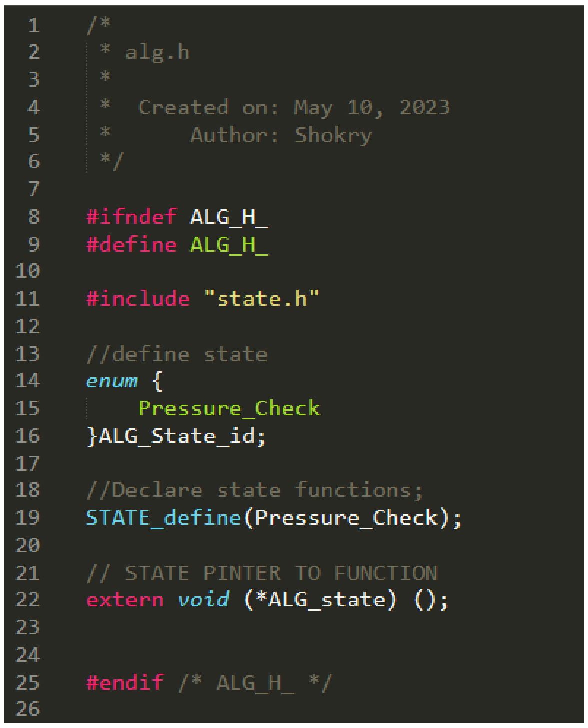


```c
#include <stdint.h>
#include <stdio.h>

#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 ();
```

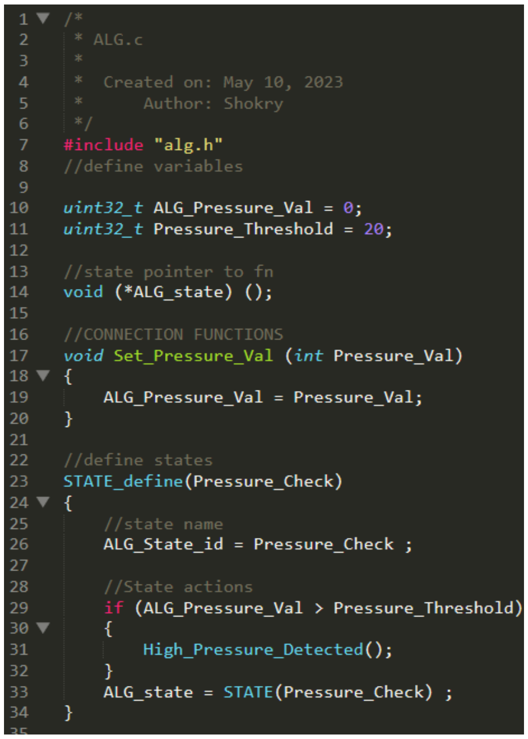


```c
#include "driver.h"
#include <stdint.h>
#include <stdio.h>
void Delay(int nCount)
{
    for(; nCount != 0; nCount--)
}

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 |= 0x00200000;
}
```

- main algorithm

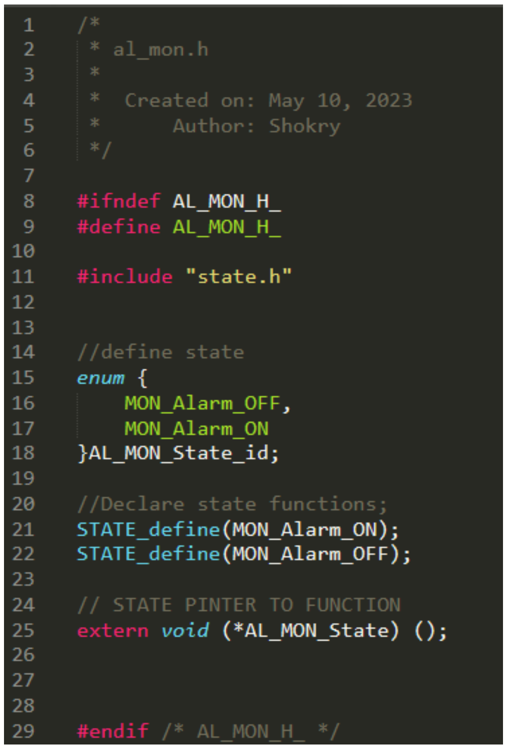


```c
/*
 * alg.h
 *
 *  Created on: May 10, 2023
 *      Author: Shokry
 */

#ifndef ALG_H_
#define ALG_H_

#include "state.h"

//define state
enum {
    Pressure_Check
}ALG_State_id;

//Declare state functions;
STATE_define(Pressure_Check);

// STATE PINTER TO FUNCTION
extern void (*ALG_state) ();


#endif /* ALG_H_ */
```

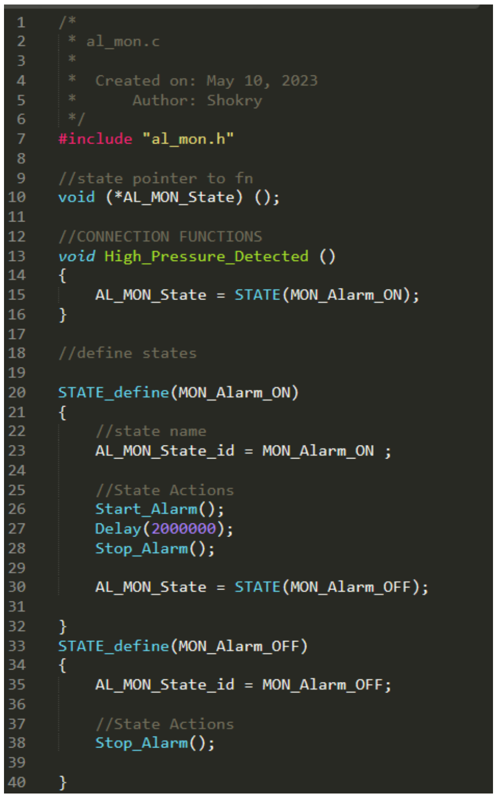


```c
/*
 * ALG.c
 *
 *  Created on: May 10, 2023
 *      Author: Shokry
 */
#include "alg.h"
//define variables

uint32_t ALG_Pressure_Val = 0;
uint32_t Pressure_Threshold = 20;

//state pointer to fn
void (*ALG_state) ();

//CONNECTION FUNCTIONS
void Set_Pressure_Val (int Pressure_Val)
{
    ALG_Pressure_Val = Pressure_Val;
}

//define states
STATE_define(Pressure_Check)
{
    //state name
    ALG_State_id = Pressure_Check ;

    //State actions
    if (ALG_Pressure_Val > Pressure_Threshold)
    {
        High_Pressure_Detected();
    }
    ALG_state = STATE(Pressure_Check) ;
}
```

- Alarm monitor

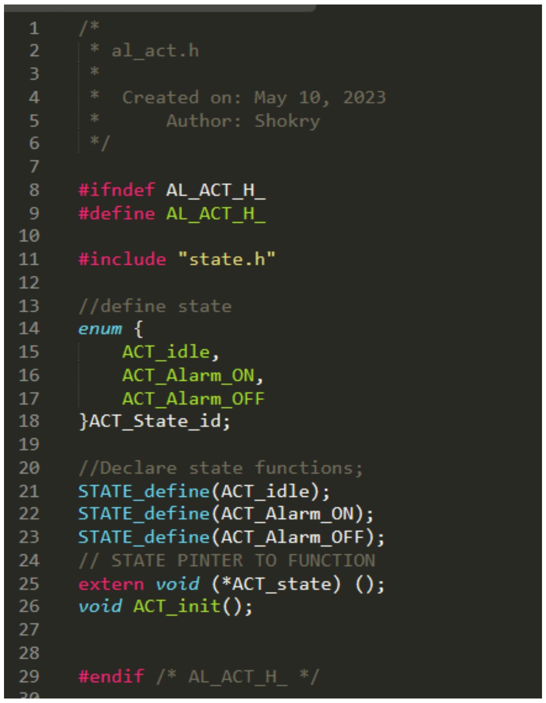


```c
/*
 * al_mon.h
 *
 *  Created on: May 10, 2023
 *      Author: Shokry
 */

#ifndef AL_MON_H_
#define AL_MON_H_

#include "state.h"


//define state
enum {
    MON_Alarm_OFF,
    MON_Alarm_ON
}AL_MON_State_id;

//Declare state functions;
STATE_define(MON_Alarm_ON);
STATE_define(MON_Alarm_OFF);

// STATE PINTER TO FUNCTION
extern void (*AL_MON_State) ();



#endif /* AL_MON_H_ */
```

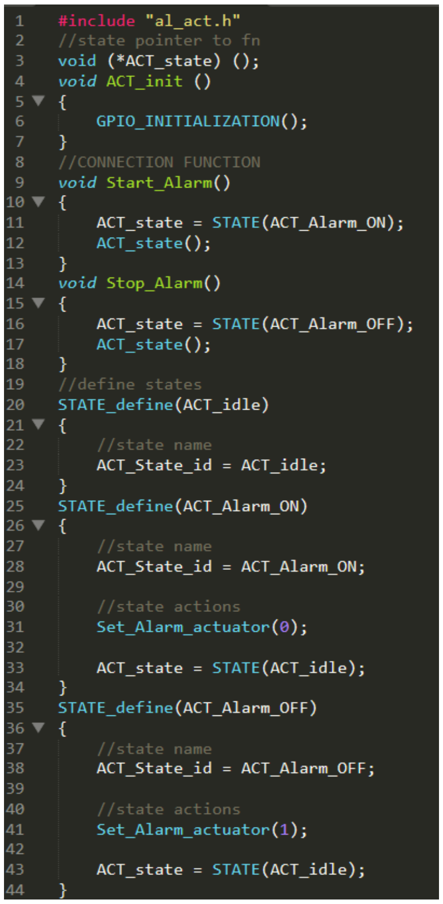


```c
/*
 * al_mon.c
 *
 *  Created on: May 10, 2023
 *      Author: Shokry
 */
#include "al_mon.h"

//state pointer to fn
void (*AL_MON_State) ();

//CONNECTION FUNCTIONS
void High_Pressure_Detected ()
{
    AL_MON_State = STATE(MON_Alarm_ON);
}

//define states

STATE_define(MON_Alarm_ON)
{
    //state name
    AL_MON_State_id = MON_Alarm_ON ;

    //State Actions
    Start_Alarm();
    Delay(2000000);
    Stop_Alarm();

    AL_MON_State = STATE(MON_Alarm_OFF);

}
STATE_define(MON_Alarm_OFF)
{
    AL_MON_State_id = MON_Alarm_OFF;

    //State Actions
    Stop_Alarm();

}
```

- Alarm actuator

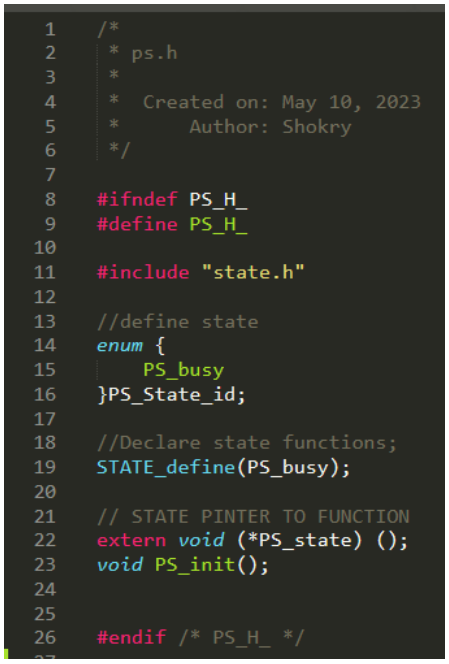


```c
/*
 * al_act.h
 *
 *  Created on: May 10, 2023
 *      Author: Shokry
 */

#ifndef AL_ACT_H_
#define AL_ACT_H_

#include "state.h"

//define state
enum {
    ACT_idle,
    ACT_Alarm_ON,
    ACT_Alarm_OFF
}ACT_State_id;

//Declare state functions;
STATE_define(ACT_idle);
STATE_define(ACT_Alarm_ON);
STATE_define(ACT_Alarm_OFF);
// STATE PINTER TO FUNCTION
extern void (*ACT_state) ();
void ACT_init();


#endif /* AL_ACT_H_ */
```

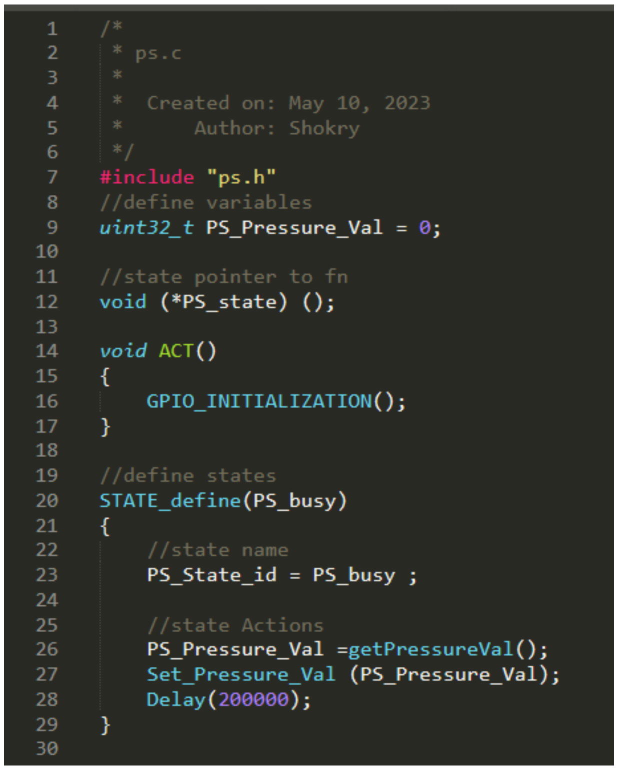


```c
#include "al_act.h"
//state pointer to fn
void (*ACT_state) ();
void ACT_init ()
{
    GPIO_INITIALIZATION();
}
//CONNECTION FUNCTION
void Start_Alarm()
{
    ACT_state = STATE(ACT_Alarm_ON);
    ACT_state();
}
void Stop_Alarm()
{
    ACT_state = STATE(ACT_Alarm_OFF);
    ACT_state();
}
//define states
STATE_define(ACT_idle)
{
    //state name
    ACT_State_id = ACT_idle;
}
STATE_define(ACT_Alarm_ON)
{
    //state name
    ACT_State_id = ACT_Alarm_ON;

    //state actions
    Set_Alarm_actuator(0);

    ACT_state = STATE(ACT_idle);
}
STATE_define(ACT_Alarm_OFF)
{
    //state name
    ACT_State_id = ACT_Alarm_OFF;

    //state actions
    Set_Alarm_actuator(1);

    ACT_state = STATE(ACT_idle);
}
```

- Pressure sensor

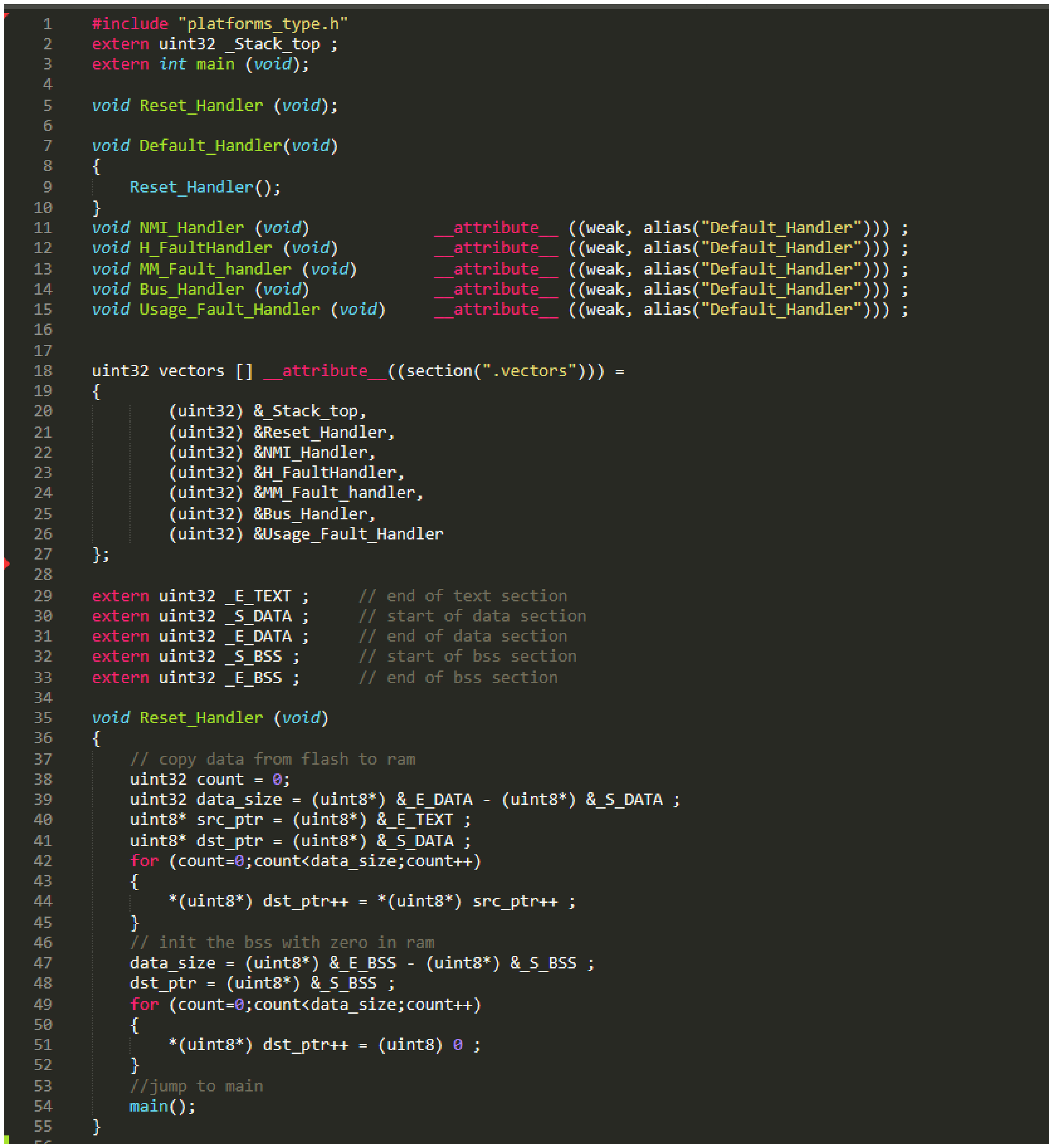


```c
/*
 * ps.h
 *
 *  Created on: May 10, 2023
 *      Author: Shokry
 */

#ifndef PS_H_
#define PS_H_

#include "state.h"

//define state
enum {
    PS_busy
}PS_State_id;

//Declare state functions;
STATE_define(PS_busy);

// STATE PINTER TO FUNCTION
extern void (*PS_state) ();
void PS_init();


#endif /* PS_H_ */
```



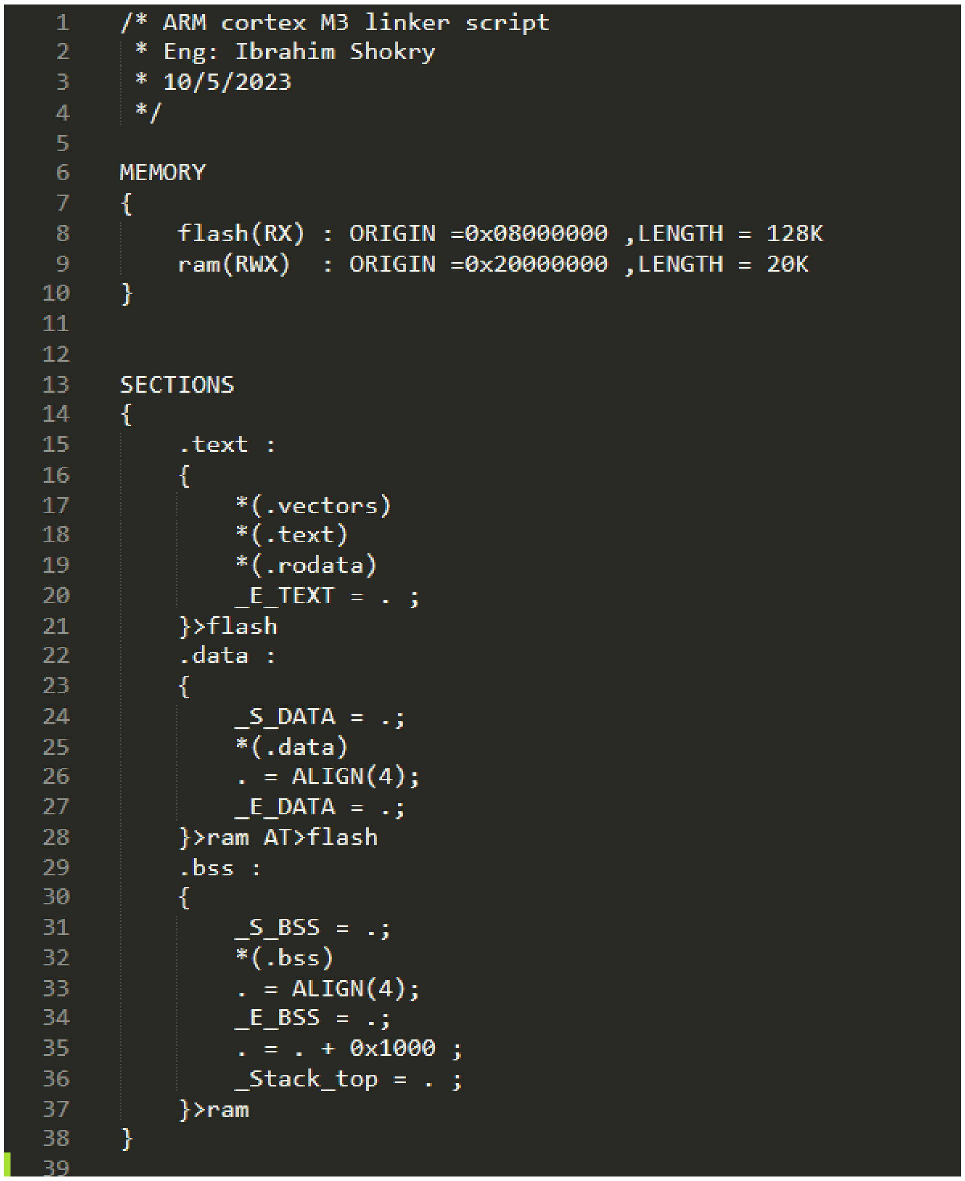
```c
/*
 * ps.c
 *
 *  Created on: May 10, 2023
 *      Author: Shokry
 */
#include "ps.h"
//define variables
uint32_t PS_Pressure_Val = 0;

//state pointer to fn
void (*PS_state) ();

void ACT()
{
    GPIO_INITIALIZATION();
}

//define states
STATE_define(PS_busy)
{
    //state name
    PS_State_id = PS_busy ;

    //state Actions
    PS_Pressure_Val =getPressureVal();
    Set_Pressure_Val (PS_Pressure_Val);
    Delay(200000);
}

```

- **Startup**



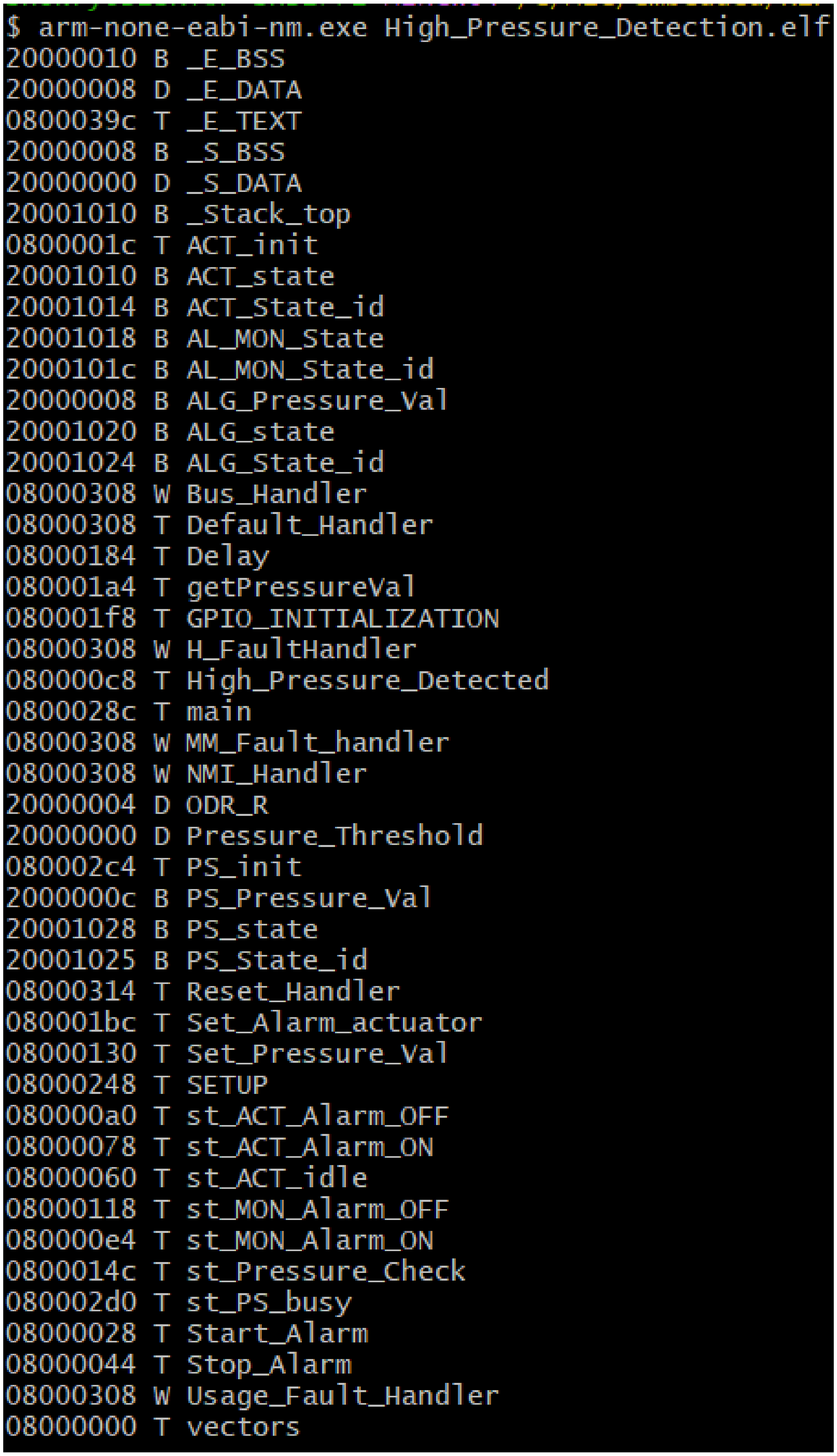
```c
#include "platforms_type.h"
extern uint32 _Stack_top ;
extern int main (void);

void Reset_Handler (void);

void Default_Handler(void)
{
    Reset_Handler();
}
void NMI_Handler (void)             __attribute__ ((weak, alias("Default_Handler"))) ;
void H_FaultHandler (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_Fault_Handler (void)     __attribute__ ((weak, alias("Default_Handler"))) ;


uint32 vectors [] __attribute__((section(".vectors"))) =
{
        (uint32) &_Stack_top,
        (uint32) &Reset_Handler,
        (uint32) &NMI_Handler,
        (uint32) &H_FaultHandler,
        (uint32) &MM_Fault_handler,
        (uint32) &Bus_Handler,
        (uint32) &Usage_Fault_Handler
};

extern uint32 _E_TEXT ;     // end of text section
extern uint32 _S_DATA ;     // start of data section
extern uint32 _E_DATA ;     // end of data section
extern uint32 _S_BSS ;      // start of bss section
extern uint32 _E_BSS ;      // end of bss section

void Reset_Handler (void)
{
    // copy data from flash to ram
    uint32 count = 0;
    uint32 data_size = (uint8*) &_E_DATA - (uint8*) &_S_DATA ;
    uint8* src_ptr = (uint8*) &_E_TEXT ;
    uint8* dst_ptr = (uint8*) &_S_DATA ;
    for (count=0;count<data_size;count++)
    {
        *(uint8*) dst_ptr++ = *(uint8*) src_ptr++ ;
    }
    // init the bss with zero in ram
    data_size = (uint8*) &_E_BSS - (uint8*) &_S_BSS ;
    dst_ptr = (uint8*) &_S_BSS ;
    for (count=0;count<data_size;count++)
    {
        *(uint8*) dst_ptr++ = (uint8) 0 ;
    }
    //jump to main
    main();
}
```

- Linkerscript



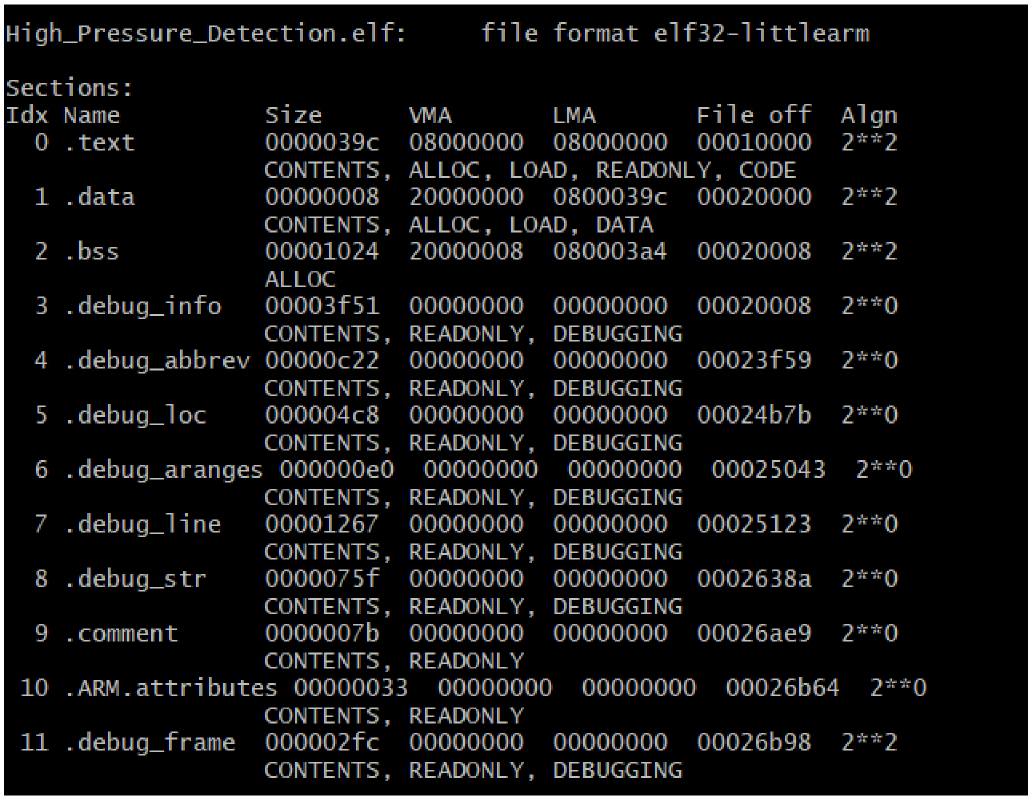
```
/* ARM cortex M3 linker script
 * Eng: Ibrahim Shokry
 * 10/5/2023
 */

MEMORY
{
    flash(RX) : ORIGIN =0x08000000 ,LENGTH = 128K
    ram(RWX)  : ORIGIN =0x20000000 ,LENGTH = 20K
}


SECTIONS
{
    .text :
    {
        *(.vectors)
        *(.text)
        *(.rodata)
        _E_TEXT = . ;
    }>flash
    .data :
    {
        _S_DATA = .;
        *(.data)
        . = ALIGN(4);
        _E_DATA = .;
    }>ram AT>flash
    .bss :
    {
        _S_BSS = .;
        *(.bss)
        . = ALIGN(4);
        _E_BSS = .;
        . = . + 0x1000 ;
        _Stack_top = . ;
    }>ram
}

```

Symbol tables



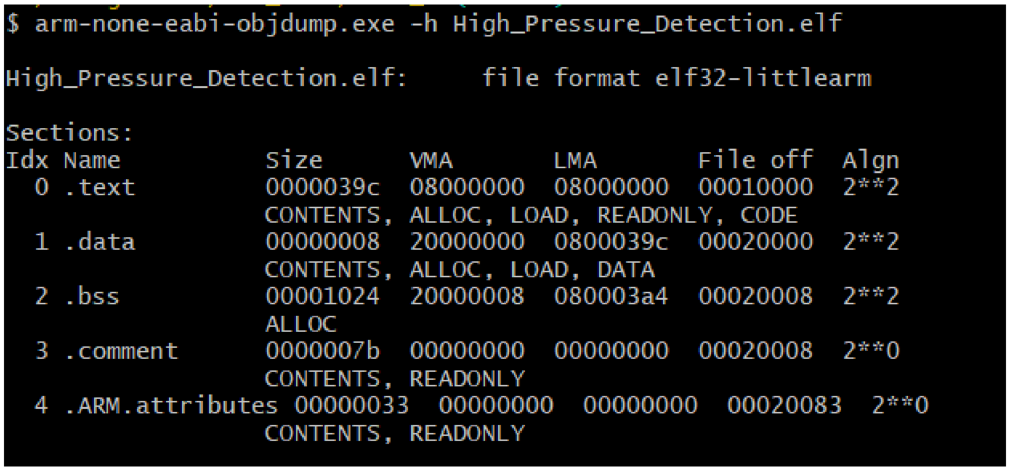
```
$ arm-none-eabi-nm.exe High_Pressure_Detection.elf
20000010 B _E_BSS
20000008 D _E_DATA
0800039c T _E_TEXT
20000008 B _S_BSS
20000000 D _S_DATA
20001010 B _Stack_top
0800001c T ACT_init
20001010 B ACT_state
20001014 B ACT_State_id
20001018 B AL_MON_State
2000101c B AL_MON_State_id
20000008 B ALG_Pressure_Val
20001020 B ALG_state
20001024 B ALG_State_id
08000308 W Bus_Handler
08000308 T Default_Handler
08000184 T Delay
080001a4 T getPressureVal
080001f8 T GPIO_INITIALIZATION
08000308 W H_FaultHandler
080000c8 T High_Pressure_Detected
0800028c T main
08000308 W MM_Fault_handler
08000308 W NMI_Handler
20000004 D ODR_R
20000000 D Pressure_Threshold
080002c4 T PS_init
2000000c B PS_Pressure_Val
20001028 B PS_state
20001025 B PS_State_id
08000314 T Reset_Handler
080001bc T Set_Alarm_actuator
08000130 T Set_Pressure_Val
08000248 T SETUP
080000a0 T st_ACT_Alarm_OFF
08000078 T st_ACT_Alarm_ON
08000060 T st_ACT_idle
08000118 T st_MON_Alarm_OFF
080000e4 T st_MON_Alarm_ON
0800014c T st_Pressure_Check
080002d0 T st_PS_busy
08000028 T Start_Alarm
08000044 T Stop_Alarm
08000308 W Usage_Fault_Handler
08000000 T vectors
```

Section table with debug information



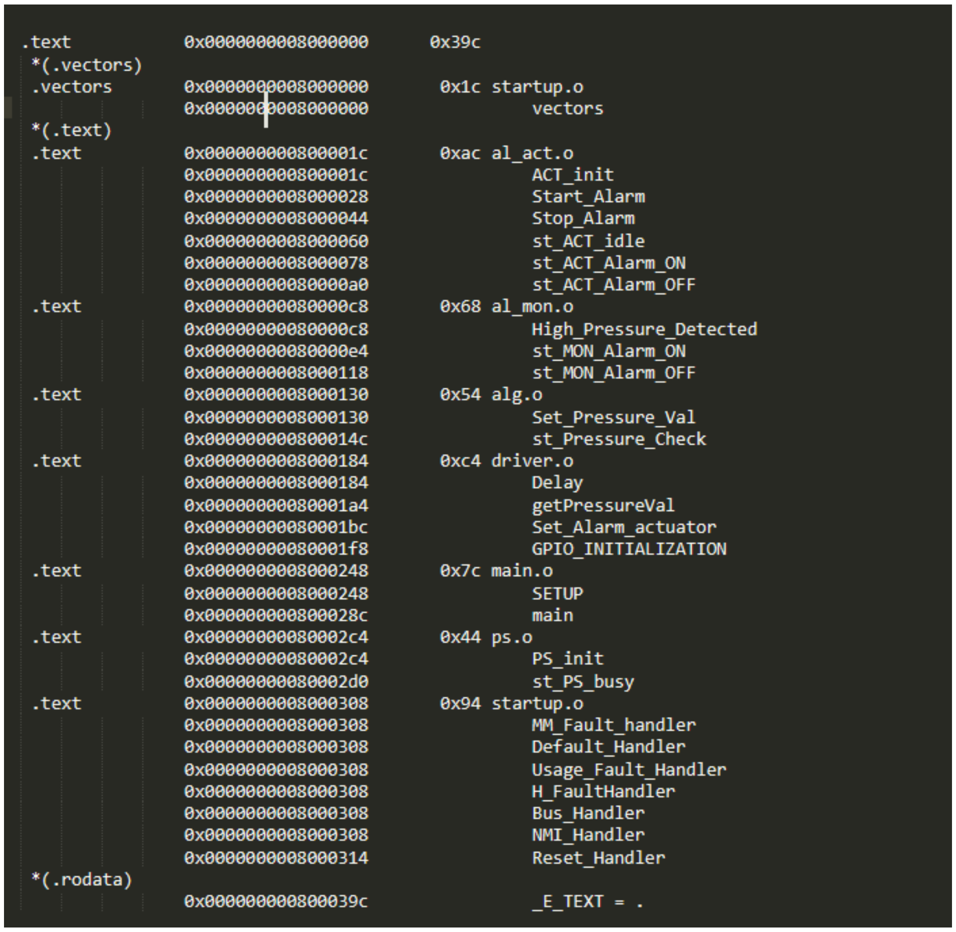
```
High_Pressure_Detection.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000039c  08000000  08000000  00010000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000008  20000000  0800039c  00020000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00001024  20000008  080003a4  00020008  2**2
                  ALLOC
  3 .debug_info   00003f51  00000000  00000000  00020008  2**0
                  CONTENTS, READONLY, DEBUGGING
  4 .debug_abbrev 00000c22  00000000  00000000  00023f59  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    000004c8  00000000  00000000  00024b7b  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 000000e0  00000000  00000000  00025043  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_line   00001267  00000000  00000000  00025123  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .debug_str    0000075f  00000000  00000000  0002638a  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      0000007b  00000000  00000000  00026ae9  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  00026b64  2**0
                  CONTENTS, READONLY
 11 .debug_frame  000002fc  00000000  00000000  00026b98  2**2
                  CONTENTS, READONLY, DEBUGGING
```

Section table without debug information

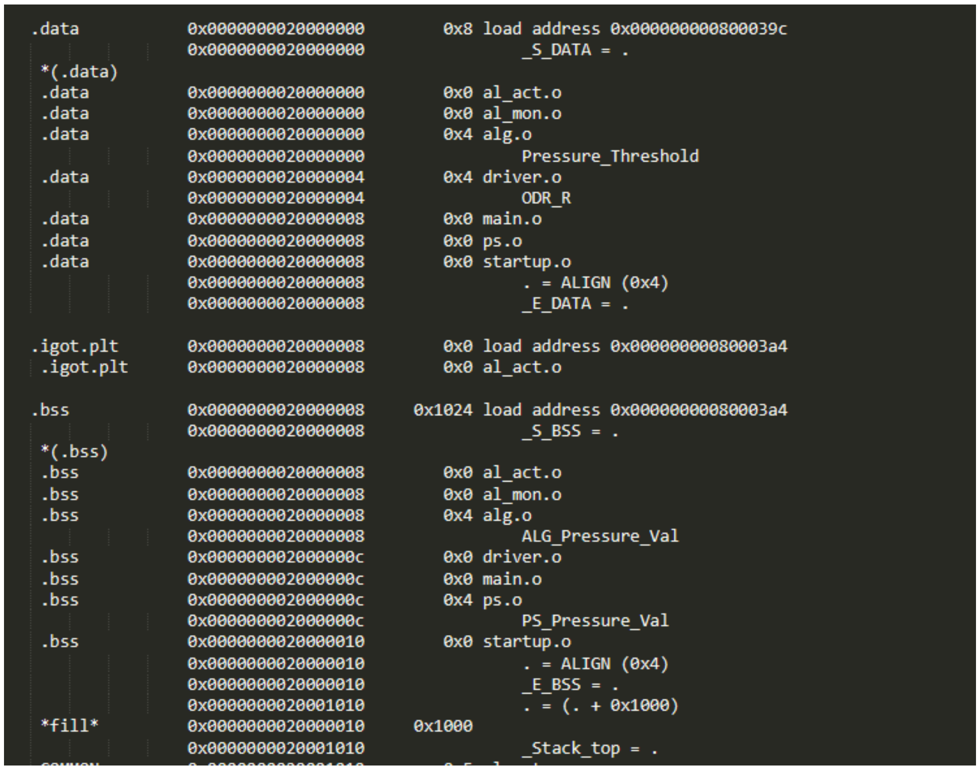


```
$ arm-none-eabi-objdump.exe -h High_Pressure_Detection.elf

High_Pressure_Detection.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000039c  08000000  08000000  00010000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000008  20000000  0800039c  00020000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00001024  20000008  080003a4  00020008  2**2
                  ALLOC
  3 .comment      0000007b  00000000  00000000  00020008  2**0
                  CONTENTS, READONLY
  4 .ARM.attributes 00000033  00000000  00000000  00020083  2**0
                  CONTENTS, READONLY
```

- Map file

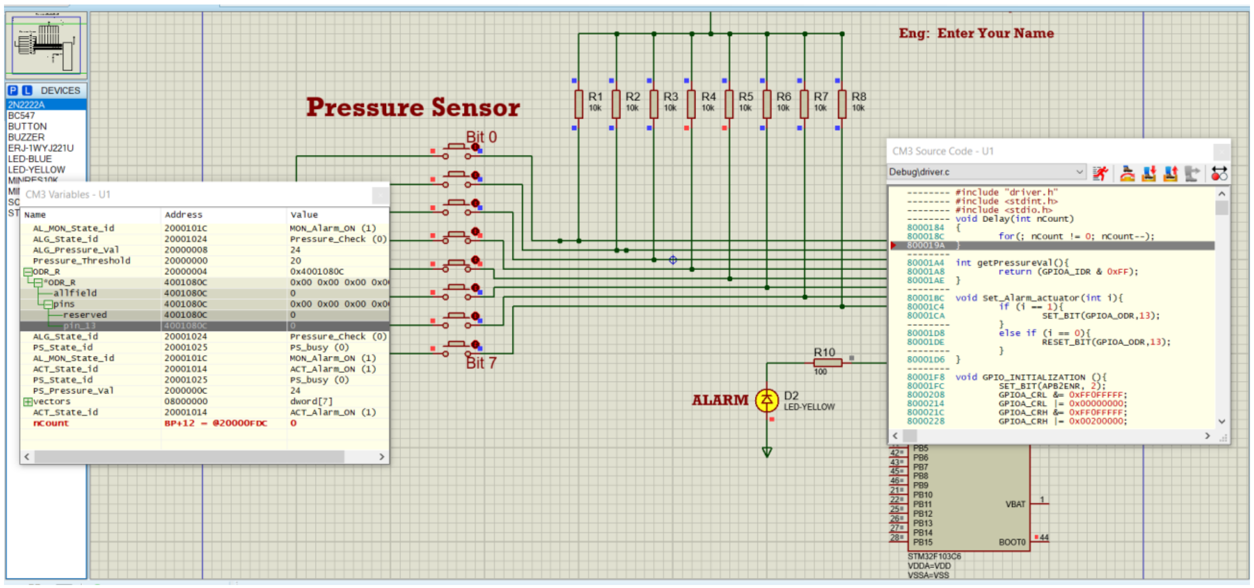


```
.text           0x0000000008000000      0x39c
 *(.vectors)
 .vectors       0x0000000008000000       0x1c startup.o
                0x0000000008000000                vectors
 *(.text)
 .text          0x000000000800001c       0xac al_act.o
                0x000000000800001c                ACT_init
                0x0000000008000028                Start_Alarm
                0x0000000008000044                Stop_Alarm
                0x0000000008000060                st_ACT_idle
                0x0000000008000078                st_ACT_Alarm_ON
                0x00000000080000a0                st_ACT_Alarm_OFF
 .text          0x00000000080000c8       0x68 al_mon.o
                0x00000000080000c8                High_Pressure_Detected
                0x00000000080000e4                st_MON_Alarm_ON
                0x0000000008000118                st_MON_Alarm_OFF
 .text          0x0000000008000130       0x54 alg.o
                0x0000000008000130                Set_Pressure_Val
                0x000000000800014c                st_Pressure_Check
 .text          0x0000000008000184       0xc4 driver.o
                0x0000000008000184                Delay
                0x00000000080001a4                getPressureVal
                0x00000000080001bc                Set_Alarm_actuator
                0x00000000080001f8                GPIO_INITIALIZATION
 .text          0x0000000008000248       0x7c main.o
                0x0000000008000248                SETUP
                0x000000000800028c                main
 .text          0x00000000080002c4       0x44 ps.o
                0x00000000080002c4                PS_init
                0x00000000080002d0                st_PS_busy
 .text          0x0000000008000308       0x94 startup.o
                0x0000000008000308                MM_Fault_handler
                0x0000000008000308                Default_Handler
                0x0000000008000308                Usage_Fault_Handler
                0x0000000008000308                H_FaultHandler
                0x0000000008000308                Bus_Handler
                0x0000000008000308                NMI_Handler
                0x0000000008000314                Reset_Handler
 *(.rodata)
                0x000000000800039c                _E_TEXT = .
```



```
.data           0x0000000020000000        0x8 load address 0x000000000800039c
                0x0000000020000000                _S_DATA = .
 *(.data)
 .data          0x0000000020000000        0x0 al_act.o
 .data          0x0000000020000000        0x0 al_mon.o
 .data          0x0000000020000000        0x4 alg.o
                0x0000000020000000                Pressure_Threshold
 .data          0x0000000020000004        0x4 driver.o
                0x0000000020000004                ODR_R
 .data          0x0000000020000008        0x0 main.o
 .data          0x0000000020000008        0x0 ps.o
 .data          0x0000000020000008        0x0 startup.o
                0x0000000020000008                . = ALIGN (0x4)
                0x0000000020000008                _E_DATA = .

.igot.plt       0x0000000020000008        0x0 load address 0x00000000080003a4
 .igot.plt      0x0000000020000008        0x0 al_act.o

.bss            0x0000000020000008     0x1024 load address 0x00000000080003a4
                0x0000000020000008                _S_BSS = .
 *(.bss)
 .bss           0x0000000020000008        0x0 al_act.o
 .bss           0x0000000020000008        0x0 al_mon.o
 .bss           0x0000000020000008        0x4 alg.o
                0x0000000020000008                ALG_Pressure_Val
 .bss           0x000000002000000c        0x0 driver.o
 .bss           0x000000002000000c        0x0 main.o
 .bss           0x000000002000000c        0x4 ps.o
                0x000000002000000c                PS_Pressure_Val
 .bss           0x0000000020000010        0x0 startup.o
                0x0000000020000010                . = ALIGN (0x4)
                0x0000000020000010                _E_BSS = .
                0x0000000020001010                . = (. + 0x1000)
 *fill*         0x0000000020000010     0x1000
                0x0000000020001010                _Stack_top = .
```

- Proteus simulation



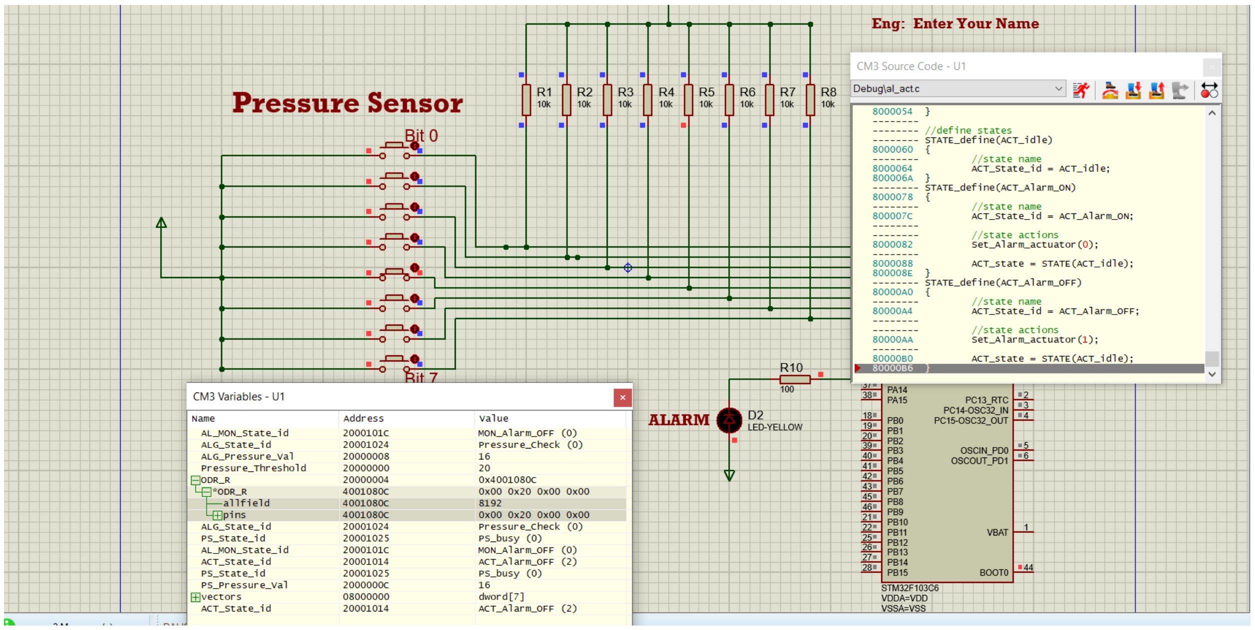
CM3 Variables - U1

Name	Address	Value
AL_MON_State_id	2000101C	MON_Alarm_ON (1)
ALG_State_id	20001024	Pressure_Check (0)
ALG_Pressure_Val	20000008	24
Pressure_Threshold	20000000	20
ODR_R	20000004	0x4001080C
*ODR_R	4001080C	0x00 0x00 0x00 0x0
allfield	4001080C	0
pins	4001080C	0x00 0x00 0x00 0x0
reserved	4001080C	0
pin_13	4001080C	0
ALG_State_id	20001024	Pressure_Check (0)
PS_State_id	20001025	PS_busy (0)
AL_MON_State_id	2000101C	MON_Alarm_ON (1)
ACT_State_id	20001014	ACT_Alarm_ON (1)
PS_State_id	20001025	PS_busy (0)
PS_Pressure_Val	2000000C	24
vectors	08000000	dword[7]
ACT_State_id	20001014	ACT_Alarm_ON (1)
nCount	BP+12 – @20000FDC	0

CM3 Source Code - U1

Debug\driver.c

```
-------- #include "driver.h"
-------- #include <stdint.h>
-------- #include <stdio.h>
-------- void Delay(int nCount)
8000184  {
800018C        for(; nCount != 0; nCount--);
800019A  }
--------
80001A4  int getPressureVal(){
80001A8        return (GPIOA_IDR & 0xFF);
80001AE  }
--------
80001BC  void Set_Alarm_actuator(int i){
80001C4        if (i == 1){
80001CA              SET_BIT(GPIOA_ODR,13);
--------       }
80001D8        else if (i == 0){
80001DE              RESET_BIT(GPIOA_ODR,13);
--------       }
80001D6  }
--------
80001F8  void GPIO_INITIALIZATION (){
80001FC        SET_BIT(APB2ENR, 2);
8000208        GPIOA_CRL &= 0xFF0FFFFF;
8000214        GPIOA_CRL |= 0x00000000;
800021C        GPIOA_CRH &= 0xFF0FFFFF;
8000228        GPIOA_CRH |= 0x00200000;
```



CM3 Source Code - U1

Debug\al_act.c

```
8000054  }
--------
-------- //define states
-------- STATE_define(ACT_idle)
8000060  {
--------       //state name
8000064        ACT_State_id = ACT_idle;
800006A  }
-------- STATE_define(ACT_Alarm_ON)
8000078  {
--------       //state name
800007C        ACT_State_id = ACT_Alarm_ON;
--------
--------       //state actions
8000082        Set_Alarm_actuator(0);
--------
8000088        ACT_state = STATE(ACT_idle);
800008E  }
-------- STATE_define(ACT_Alarm_OFF)
80000A0  {
--------       //state name
80000A4        ACT_State_id = ACT_Alarm_OFF;
--------
--------       //state actions
80000AA        Set_Alarm_actuator(1);
--------
80000B0        ACT_state = STATE(ACT_idle);
80000B6  }
```

CM3 Variables - U1

Name	Address	Value
AL_MON_State_id	2000101C	MON_Alarm_OFF (0)
ALG_State_id	20001024	Pressure_Check (0)
ALG_Pressure_Val	20000008	16
Pressure_Threshold	20000000	20
ODR_R	20000004	0x4001080C
*ODR_R	4001080C	0x00 0x20 0x00 0x00
allfield	4001080C	8192
pins	4001080C	0x00 0x20 0x00 0x00
ALG_State_id	20001024	Pressure_Check (0)
PS_State_id	20001025	PS_busy (0)
AL_MON_State_id	2000101C	MON_Alarm_OFF (0)
ACT_State_id	20001014	ACT_Alarm_OFF (2)
PS_State_id	20001025	PS_busy (0)
PS_Pressure_Val	2000000C	16
vectors	08000000	dword[7]
ACT_State_id	20001014	ACT_Alarm_OFF (2)