Mastering embedded system online diploma

https://www.learn-in-depth.com/

Pressure controller

First term (final project 1)

Eng. Hassan Ashraf Hassan Elsalakawy

My profile:

https://www.learn-in-depth.com/online-diploma/itshassan599%40gmail.com

Table of Contents

Case study

In this report, a design work for controlling pressure inside the aircraft cabin will be discussed by using a special sensor to measure atmospheric pressure, and the work that we will specialize in doing in that system will be determined based on the following Assumptions:

- The controller setup and shutdown procedure are not modeled.
- The controller maintenance is not modelled.
- The pressure sensor never fails.
- The alarm never fails.
- The controller never focuses on the power cut.

Versioning:

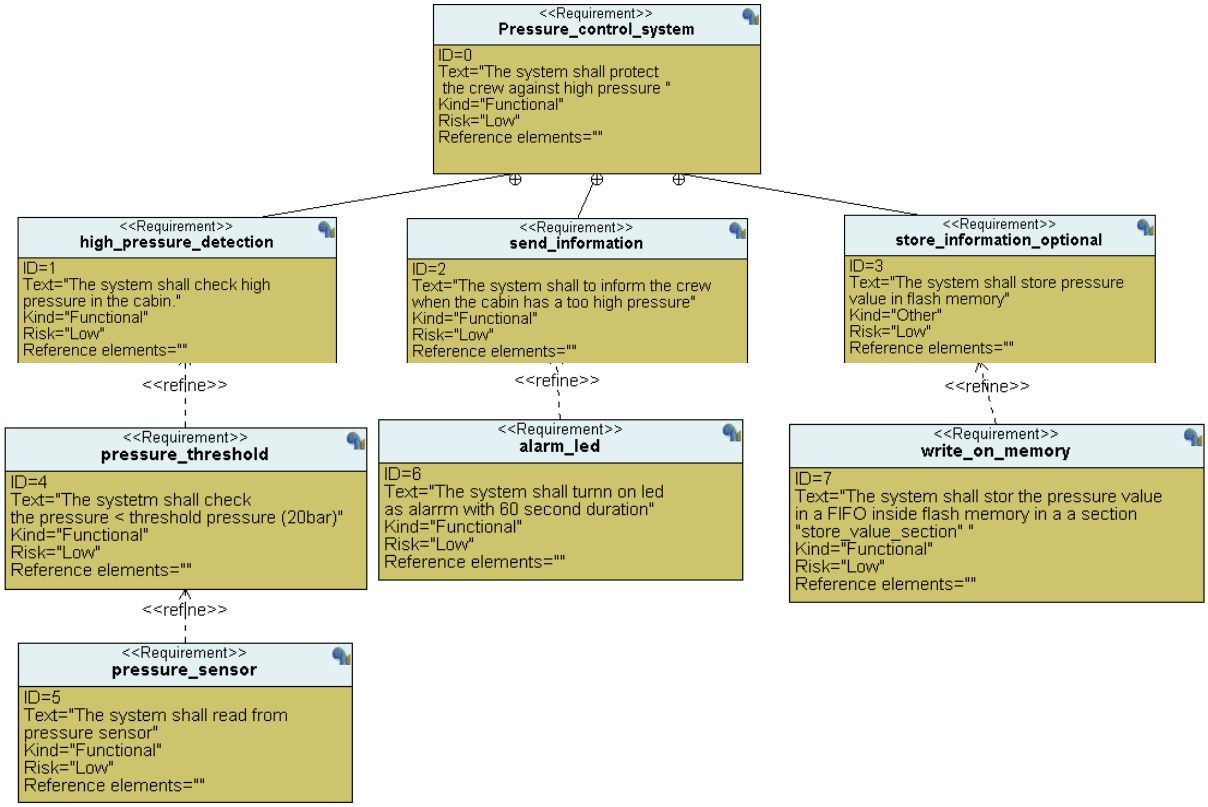
- The keep track of measured value option is not modeled in the first version.

Method

We have chosen the system V-cycle as it will provide us with the following factors:

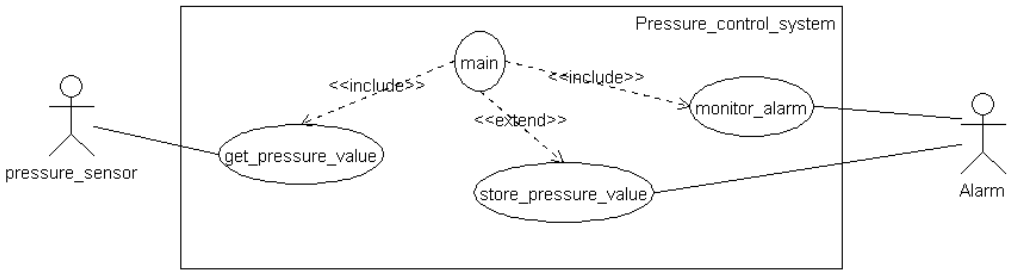
- Easy to set up.
- Productivity improvement.
- Time saving.
- Money saving.
- Improvement of the quality of the delivered product.

Requirements

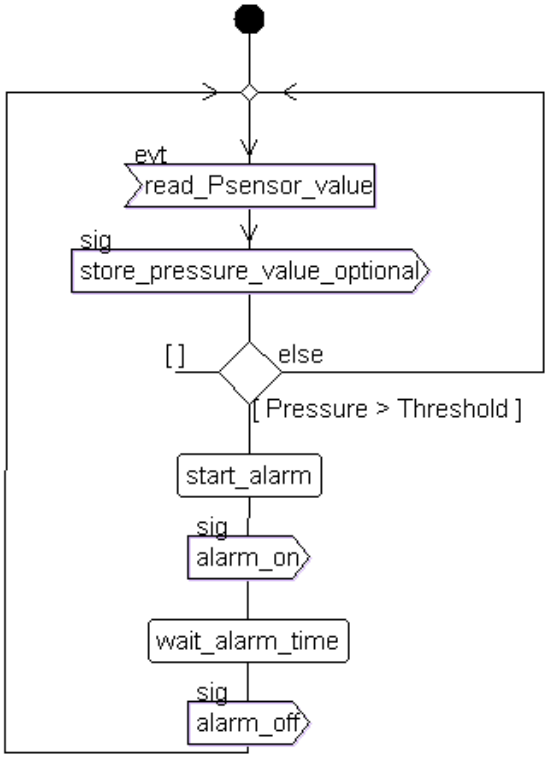


System analysis

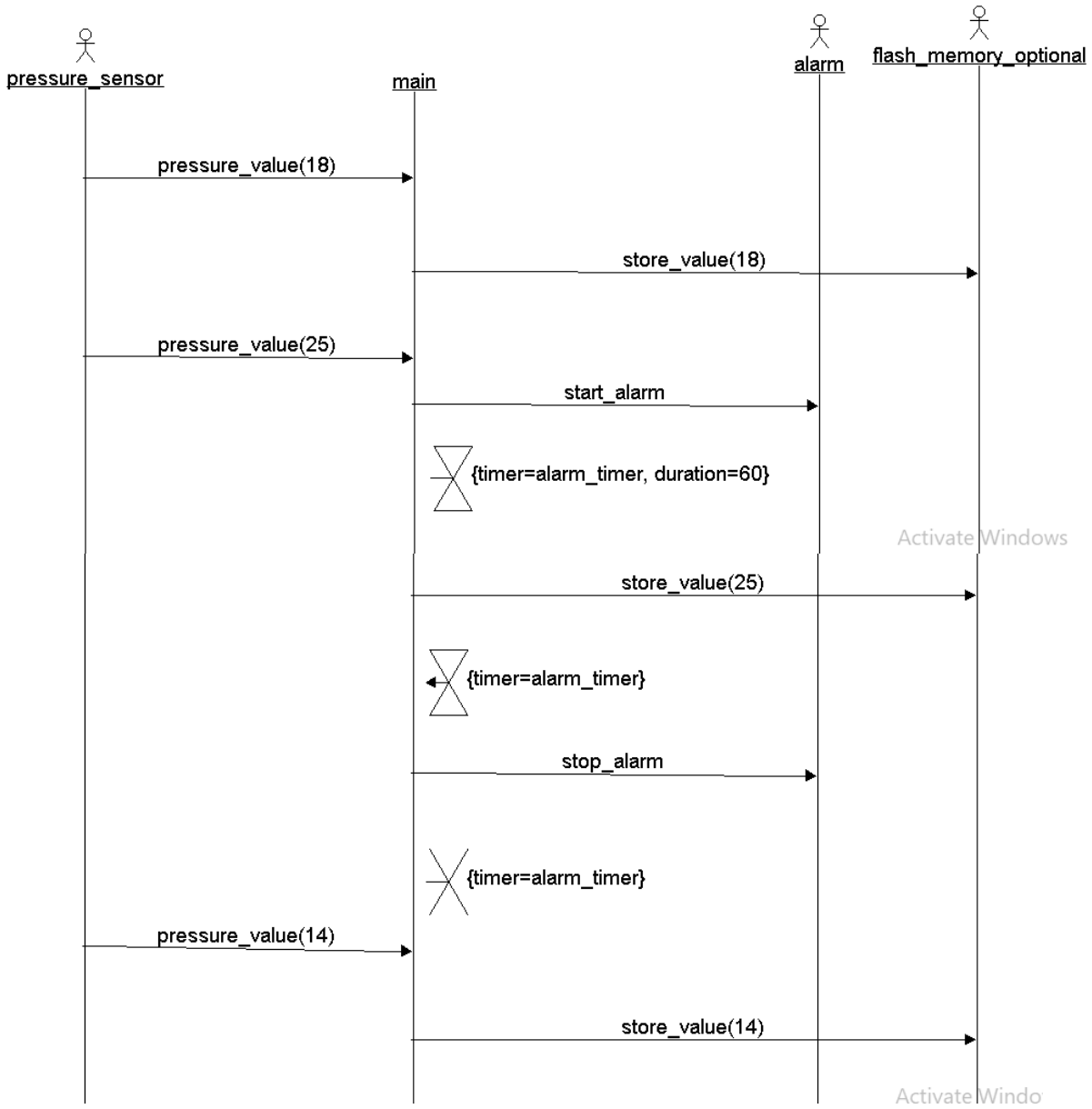
- **Use case diagram.**



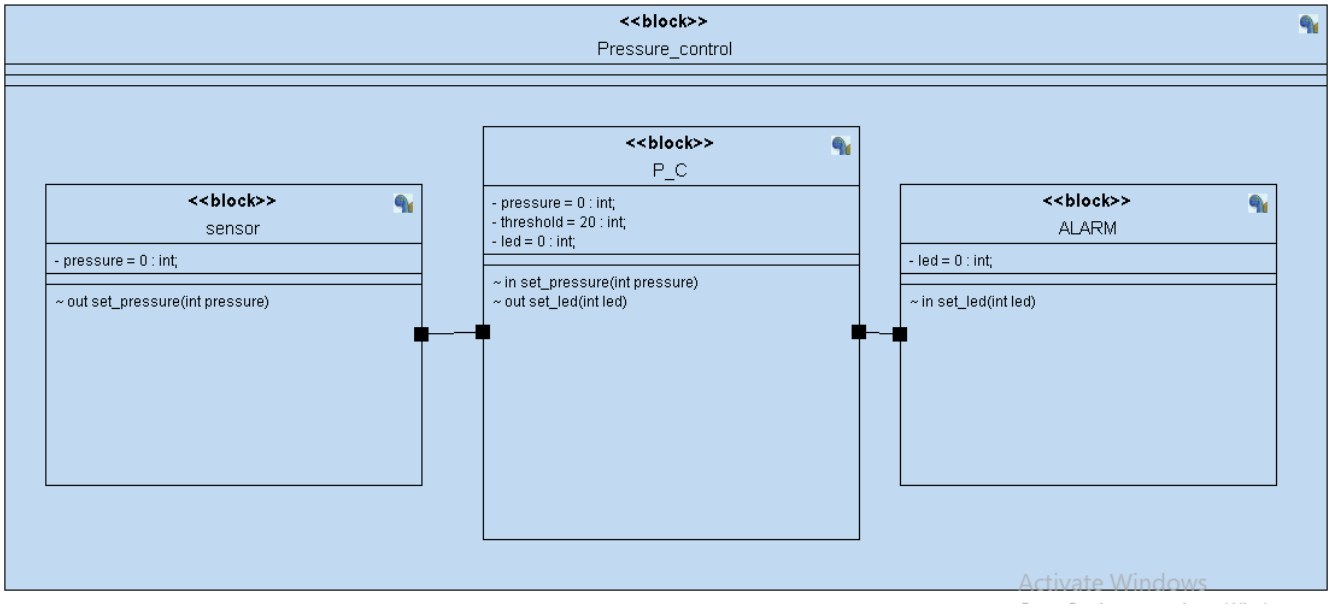
- **Activity diagram.**

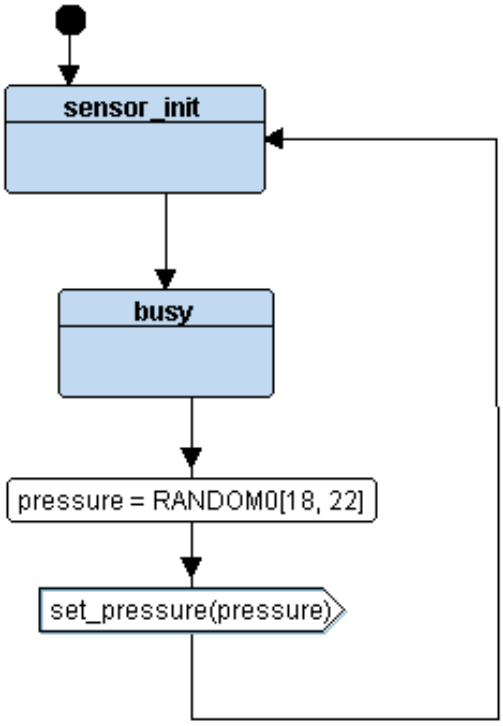


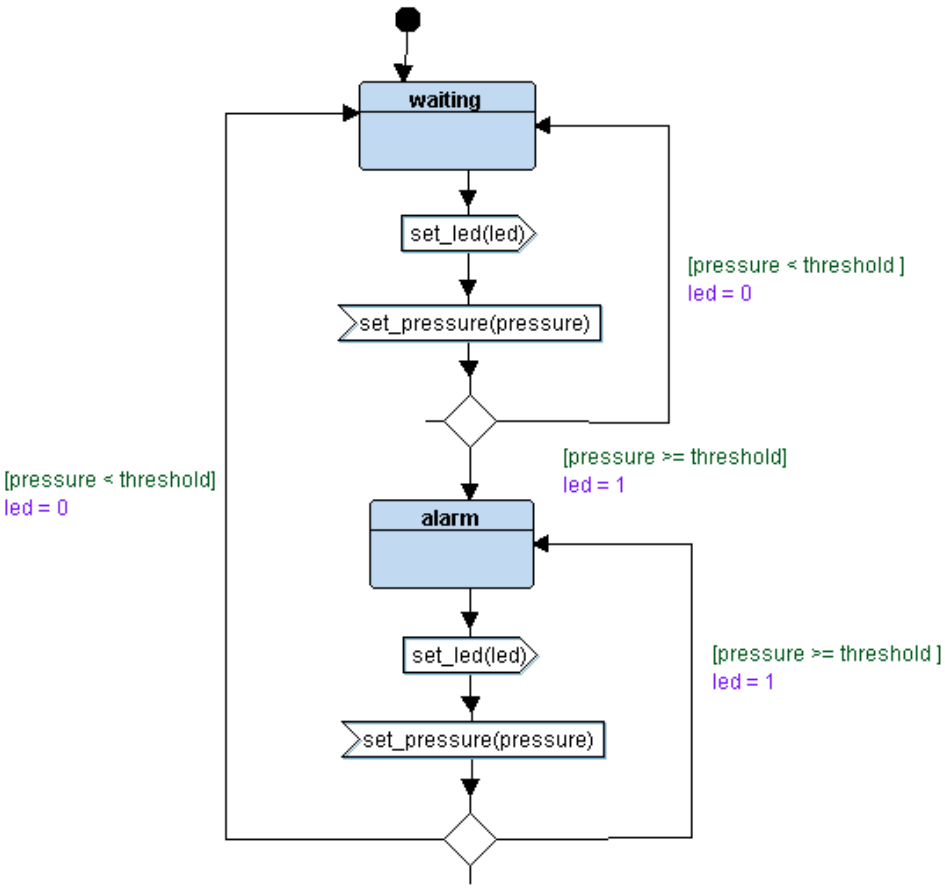
- **Sequence diagram**

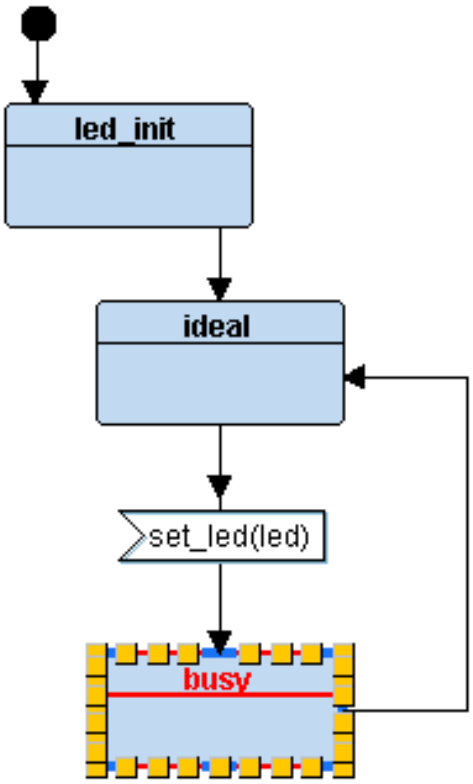


System design






waiting
set_led(led)
set_pressure(pressure)
[pressure < threshold]
led = 0
[pressure >= threshold]
led = 1
[pressure < threshold]
led = 0
alarm
set_led(led)
set_pressure(pressure)
[pressure >= threshold]
led = 1


led_init
ideal
set_led(led)
busy

Implementation code

- **(.c) &(.h) files**

- **PS.c**

```c
#include "PS.h"

//variables
int PS_pressure = 0;
PS_enum PS_state_id;

void (*PS_state)();

void PS_init(){
    //printf("=====PS_INIT=====\n");
}

STATE_def(PS_busy){

    //State Name
    PS_state_id = PS_busy;

    //State Action
    PS_pressure = getPressureVal();
    //printf("PS_busy state: pressure = %d \n", PS_pressure);
    PS_set_pressure(PS_pressure);
    PS_state = STATE(PS_busy);
}
```

- **PS.h**

```c

#ifndef PS_H_
#define PS_H_
#include "state.h"

//define state
typedef enum{
    PS_busy
} PS_enum;
extern PS_enum PS_state_id;

void PS_init();

STATE_def(PS_busy);

extern void (*PS_state)();

#endif /* PS_H_ */
```

- **CP.c**

```c
#include "CP.h"
//variables
int CP_led = 0;
int CP_pressure = 0;
int CP_threshold = 20;

CP_enum CP_state_id;
void (*CP_state)();

void PS_set_pressure(int pressure){
    CP_pressure = pressure;
    (CP_pressure <= CP_threshold)? (CP_state = STATE(CP_waiting)) : (CP_state = STATE(CP_driving));
}

STATE_def(CP_waiting){
    //State Name
    CP_state_id = CP_waiting;
    //State Action
    CP_led = 1;
    Set_Alarm_actuator(CP_led);
}
STATE_def(CP_driving){
    //State Name
    CP_state_id = CP_driving;
    //State Action
    CP_led =0;
    //ALARM_set_led(CP_led);
    Set_Alarm_actuator(CP_led);
}
```

- **CP.h**

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

typedef enum{
    CP_waiting,
    CP_driving
} CP_enum;
extern CP_enum CP_state_id;

STATE_def(CP_waiting);
STATE_def(CP_driving);

extern void (*CP_state)();

#endif /* CP_H_ */
```

- **alarm.c**

```c
#include "alarm.h"
int ALARM_led = 0;
ALARM_enum ALARM_state_id;
void (*ALARM_state)();
void ALARM_init(){
    //printf("=====ALARM_INIT===== \n");
}
void Set_Alarm_actuator(int led){
    ALARM_led = led;
    ALARM_state = STATE(ALARM_busy);
    if (led == 1){
        SET_BIT(GPIOA_ODR,13);
    }
    else if (led == 0){
        RESET_BIT(GPIOA_ODR,13);
    }
}
STATE_def(ALARM_ideal){
    //State Name
    ALARM_state_id = ALARM_ideal;
    //State Action
}
STATE_def(ALARM_busy){
    //State Name
    ALARM_state_id = ALARM_busy;
    //State Action
    ALARM_state = STATE(ALARM_ideal);
}
```

- **alarm.h**

```c
#ifndef ALARM_H_
#define ALARM_H_

#include "state.h"

typedef enum{
    ALARM_ideal,
    ALARM_busy
} ALARM_enum;
extern ALARM_enum ALARM_state_id;

void ALARM_init();

STATE_def(ALARM_ideal);
STATE_def(ALARM_busy);

extern void (*ALARM_state)();

#endif /* ALARM_H_ */
```

- **driver.c**

```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 GPIO_INITIALIZATION (){
    SET_BIT(APB2ENR, 2);
    GPIOA_CRL &= 0xFF0FFFFF;
    GPIOA_CRL |= 0x00000000;
    GPIOA_CRH &= 0xFF0FFFFF;
    GPIOA_CRH |= 0x22222222;
}
```

- **driver.h**

```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 ();
```

- **main.c**

```c
#include "CP.h"
#include "PS.h"
#include "alarm.h"
void setup(){

    //Initialize Blocks
    PS_init();
    ALARM_init();

    //Set State pointers to their corresponding block
    CP_state = STATE(CP_waiting);
    PS_state = STATE(PS_busy);
    ALARM_state = STATE(ALARM_ideal);
}
void main(){
    GPIO_INITIALIZATION();
    setup();
    while(1){
        //Call state pointer of each block
        PS_state();
        CP_state();
        ALARM_state();

        //Delay loop
        Delay(2000);
    }
}
```

- **state.h**

```c
#ifndef STATE_H_
#define STATE_H_
#include "driver.h"
#include <stdio.h>
#include <stdlib.h>

//State function generator
#define STATE_def(_stateFun_) void State_##_stateFun_()
#define STATE(_stateFun_) State_##_stateFun_

//Triggered signals interface
void PS_set_pressure(int pressure);
void ALARM_set_led(int led);



#endif /* STATE_H_ */
```

- **Makefile**

```makefile
CC          =arm-none-eabi-
CFLAGS      =-mcpu=cortex-m4 -mthumb -gdwarf-2 -g
INCS        =-I .
LIBS        =
SRC         =$(wildcard *.c)
OBJ         =$(SRC:.c=.o)
ASM         =$(wildcard *.s)
ASMOBJ      =$(ASM:.s=.o)
LINKER      =$(wildcard *.ld)
Project_Name = pressure_control_system

all: $(Project_Name).bin
    @echo "====mission completed======"

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

$(Project_Name).elf : $(OBJ) $(ASMOBJ)
    $(CC)ld.exe -T $(LINKER) $(OBJ) $(ASMOBJ) $(LIBS) -Map=Map_File.map -o $@ -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

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

- **Linkerscript.ld**

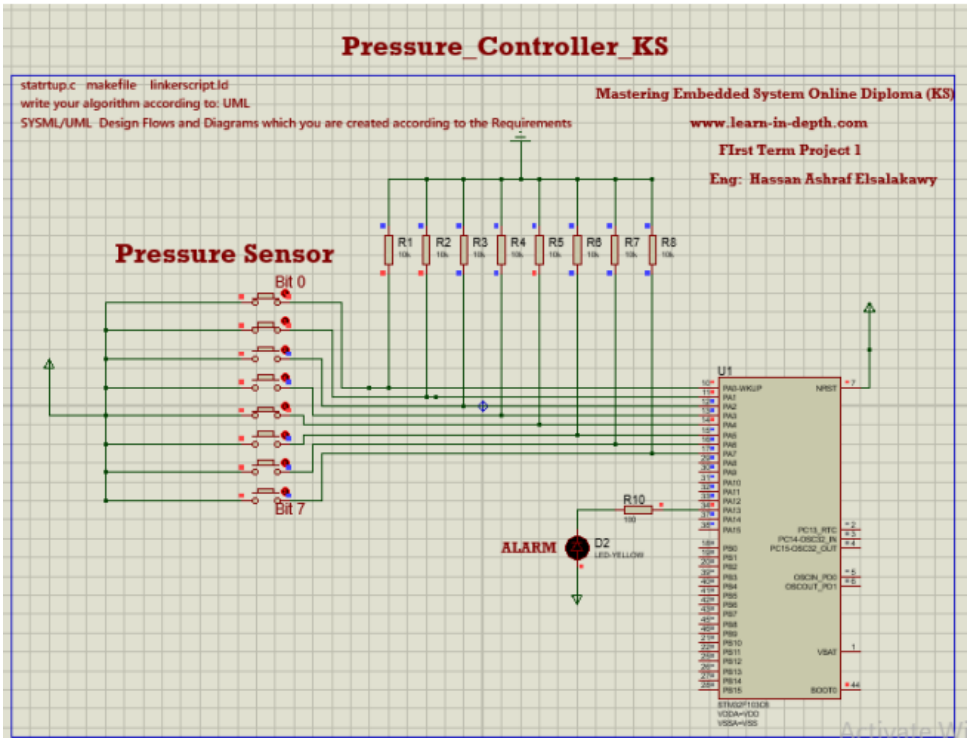
```linker script
MEMORY {
    Flash(RX) : ORIGIN = 0x00000000, LENGTH = 512M
    SRAM(RWX) : ORIGIN = 0x20000000, LENGTH = 512M
}
SECTIONS {
    .text : {
        *(.vectors*)
        . = ALIGN(4) ;
        *(.text*)
        . = ALIGN(4) ;
        *(.rodata*)
        . = ALIGN(4) ;
        _E_TEXT = .;
    }> Flash

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

    .bss : {
        _S_bss = .;
        *(.bss*)
        . = ALIGN(4) ;
        _E_bss = .;
    }> SRAM
}
```

- **Startup.c**

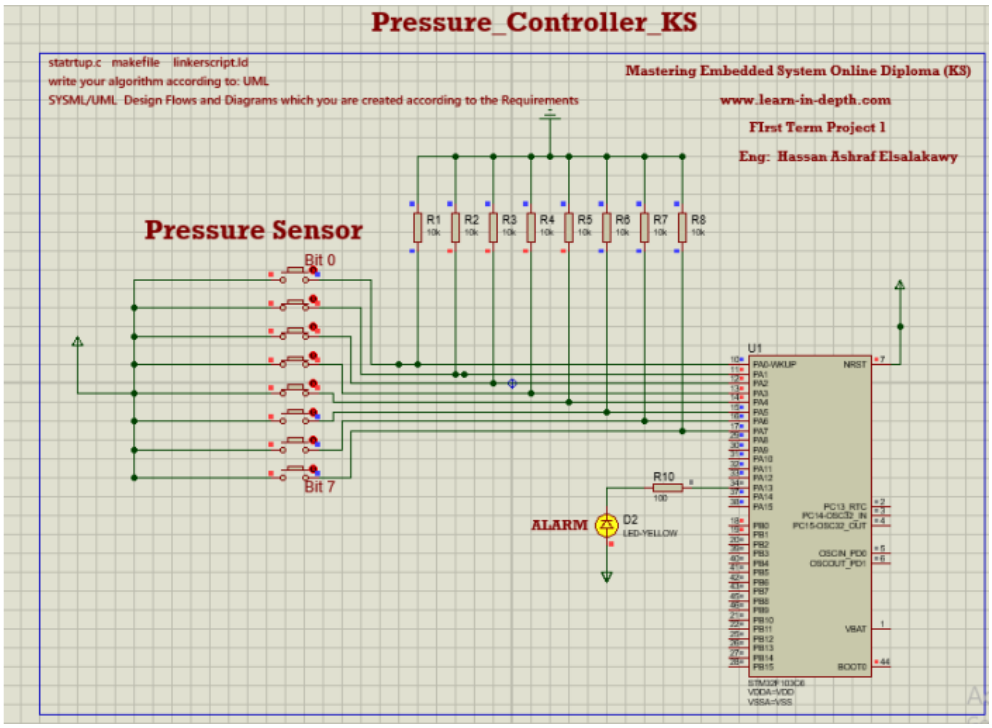
```c
#include  <stdint.h>
extern int main (void);
extern unsigned int  _E_TEXT;
extern unsigned int  _S_DATA;
extern unsigned int  _E_DATA;
extern unsigned int  _S_bss;
extern unsigned int  _E_bss;

void  reset_handler(){
    //copy .data  from flash to  ram
    unsigned int Data_size = (unsigned char*)&_E_DATA - (unsigned char*)&_S_DATA;
    unsigned char* P_Src = (unsigned char*)&_E_TEXT;
    unsigned char* P_Dst = (unsigned char*)&_S_DATA;
    int i;
    for(i=0;i<Data_size;i++) {
        *P_Dst = *P_Src;
        P_Dst++;
        P_Src++;
    }
    // Init bss section in Ram with 0
    unsigned int bss_size = (unsigned char*)&_E_bss - (unsigned char*)&_S_bss;
    P_Dst = (unsigned char*)&_S_bss;

    for(i=0;i<bss_size;i++) {
        *P_Dst = (unsigned char)0;
        P_Dst++;
    }
    main();
}


void default_handler (){
    reset_handler();
}

void NMI_handler ()  __attribute__((weak,alias("default_handler")));;
void H_fault_handler ()  __attribute__((weak,alias("default_handler")));;



static unsigned long  stack_top[256]; //  1024byte=256*4

void  (* const fun_to_vectors []) () __attribute__((section(".vectors"))) =
{
    (void (*)()) ((unsigned long)stack_top + sizeof(stack_top)),
     &reset_handler,
     &NMI_handler,
     &H_fault_handler
};
```

Simulation

- **Pressure = 19 bar < threshold (20 bar)**



- **Pressure = 19 bar < threshold (20 bar)**



SW analysis

- **Mapfile**

```
Allocating common symbols
Common symbol       size              file

ALARM_state_id      0x1               alarm.o
CP_state            0x4               CP.o
CP_state_id         0x1               CP.o
ALARM_state         0x4               alarm.o
PS_state            0x4               PS.o
PS_state_id         0x1               PS.o

Memory Configuration

Name             Origin             Length             Attribute
Flash            0x00000000         0x20000000         xr
SRAM             0x20000000         0x20000000         xrw
*default*        0x00000000         0xffffffff

Linker script and memory map


.text           0x00000000      0x3e0
 *(.vectors*)
 .vectors       0x00000000       0x10 startup.o
                0x00000000                fun_to_vectors
                0x00000010                . = ALIGN (0x4)
 *(.text*)
 .text          0x00000010       0xc4 CP.o
                0x00000010                PS_set_pressure
                0x0000006c                State_CP_waiting
                0x000000a0                State_CP_driving
 .text          0x000000d4       0x54 PS.o
                0x000000d4                PS_init
                0x000000e0                State_PS_busy
 .text          0x00000128       0xc0 alarm.o
                0x00000128                ALARM_init
                0x00000134                Set_Alarm_actuator
                0x000001a4                State_ALARM_ideal
                0x000001bc                State_ALARM_busy
 .text          0x000001e8       0xbc driver.o
                0x000001e8                Delay
                0x0000020c                getPressureVal
                0x00000224                GPIO_INITIALIZATION
 .text          0x000002a4       0x80 main.o
                0x000002a4                setup
                0x000002e8                main
 .text          0x00000324       0xbc startup.o
                0x00000324                reset_handler
                0x000003d4                default_handler
                0x000003d4                H_fault_handler
                0x000003d4                NMI_handler
                0x000003e0                . = ALIGN (0x4)
```

```
 *(.rodata*)
                0x000003e0                 . = ALIGN (0x4)
                0x000003e0                 _E_TEXT = .

.glue_7         0x000003e0        0x0
 .glue_7        0x00000000        0x0 linker stubs

.glue_7t        0x000003e0        0x0
 .glue_7t       0x00000000        0x0 linker stubs

.vfp11_veneer   0x000003e0        0x0
 .vfp11_veneer  0x00000000        0x0 linker stubs

.v4_bx          0x000003e0        0x0
 .v4_bx         0x00000000        0x0 linker stubs

.iplt           0x000003e0        0x0
 .iplt          0x00000000        0x0 CP.o

.rel.dyn        0x000003e0        0x0
 .rel.iplt      0x00000000        0x0 CP.o

.data           0x20000000        0x4 load address 0x000003e0
                0x20000000                 _S_DATA = .
 *(.data*)
 .data          0x20000000        0x4 CP.o
                0x20000000                 CP_threshold
 .data          0x20000004        0x0 PS.o
 .data          0x20000004        0x0 alarm.o
 .data          0x20000004        0x0 driver.o
 .data          0x20000004        0x0 main.o
 .data          0x20000004        0x0 startup.o
                0x20000004                 . = ALIGN (0x4)
                0x20000004                 _E_DATA = .

.igot.plt       0x20000004        0x0 load address 0x000003e4
 .igot.plt      0x00000000        0x0 CP.o

.bss            0x20000004      0x428 load address 0x000003e4
                0x20000004                 _S_bss = .
 *(.bss*)
 .bss           0x20000004        0x8 CP.o
                0x20000004                 CP_led
                0x20000008                 CP_pressure
 .bss           0x2000000c        0x4 PS.o
                0x2000000c                 PS_pressure
 .bss           0x20000010        0x4 alarm.o
                0x20000010                 ALARM_led
 .bss           0x20000014        0x0 driver.o
 .bss           0x20000014        0x0 main.o
 .bss           0x20000014      0x400 startup.o
                0x20000414                 . = ALIGN (0x4)
                0x20000414                 _E_bss = .
```

```
 COMMON          0x20000414        0x5 CP.o
                 0x20000414                CP_state
                 0x20000418                CP_state_id
 *fill*          0x20000419        0x3
 COMMON          0x2000041c        0x5 PS.o
                 0x2000041c                PS_state
                 0x20000420                PS_state_id
 *fill*          0x20000421        0x3
 COMMON          0x20000424        0x8 alarm.o
                 0x20000424                ALARM_state_id
                 0x20000428                ALARM_state
LOAD CP.o
LOAD PS.o
LOAD alarm.o
LOAD driver.o
LOAD main.o
LOAD startup.o
OUTPUT(pressure_control_system.elf elf32-littlearm)
```

- **Symbols table**

MINGW32:/h/kerlos diploma/master embedded/Assignment/unit5_final/project_1/source code

```
hassa@DESKTOP-OPAITC3 MINGW32 /h/kerlos diploma/master embedded/Assignment/unit5
_final/project_1/source code
$ arm-none-eabi-nm pressure_control_system.elf
20000414 B _E_bss
20000004 D _E_DATA
000003e0 T _E_TEXT
20000004 B _S_bss
20000000 D _S_DATA
00000128 T ALARM_init
20000010 B ALARM_led
20000428 B ALARM_state
20000424 B ALARM_state_id
20000004 B CP_led
20000008 B CP_pressure
20000414 B CP_state
20000418 B CP_state_id
20000000 D CP_threshold
000003d4 T default_handler
000001e8 T Delay
00000000 T fun_to_vectors
0000020c T getPressureVal
00000224 T GPIO_INITIALIZATION
000003d4 W H_fault_handler
000002e8 T main
000003d4 W NMI_handler
000000d4 T PS_init
2000000c B PS_pressure
00000010 T PS_set_pressure
2000041c B PS_state
20000420 B PS_state_id
00000324 T reset_handler
00000134 T Set_Alarm_actuator
000002a4 T setup
20000014 b stack_top
000001bc T State_ALARM_busy
000001a4 T State_ALARM_ideal
000000a0 T State_CP_driving
0000006c T State_CP_waiting
000000e0 T State_PS_busy
```