

**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

1. Case study .....	3
2. Method .....	3
3. Requirments .....	4
4. System analysis .....	5
4.1. Use case diagram .....	5
4.2. Activity diagram .....	5
4.3. Sequence diagram .....	6
5. System design .....	7
6. Implementation code.....	9
6.1. (.c) &(.h) files .....	10
6.2. Makefile .....	14
6.3. Linkerscript.ld.....	14
6.4. Startup.c .....	15
7. Simulation .....	16
8. SW analysis.....	17
8.1. Mapfile.....	17
8.2. Symbols table.....	19

## 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:

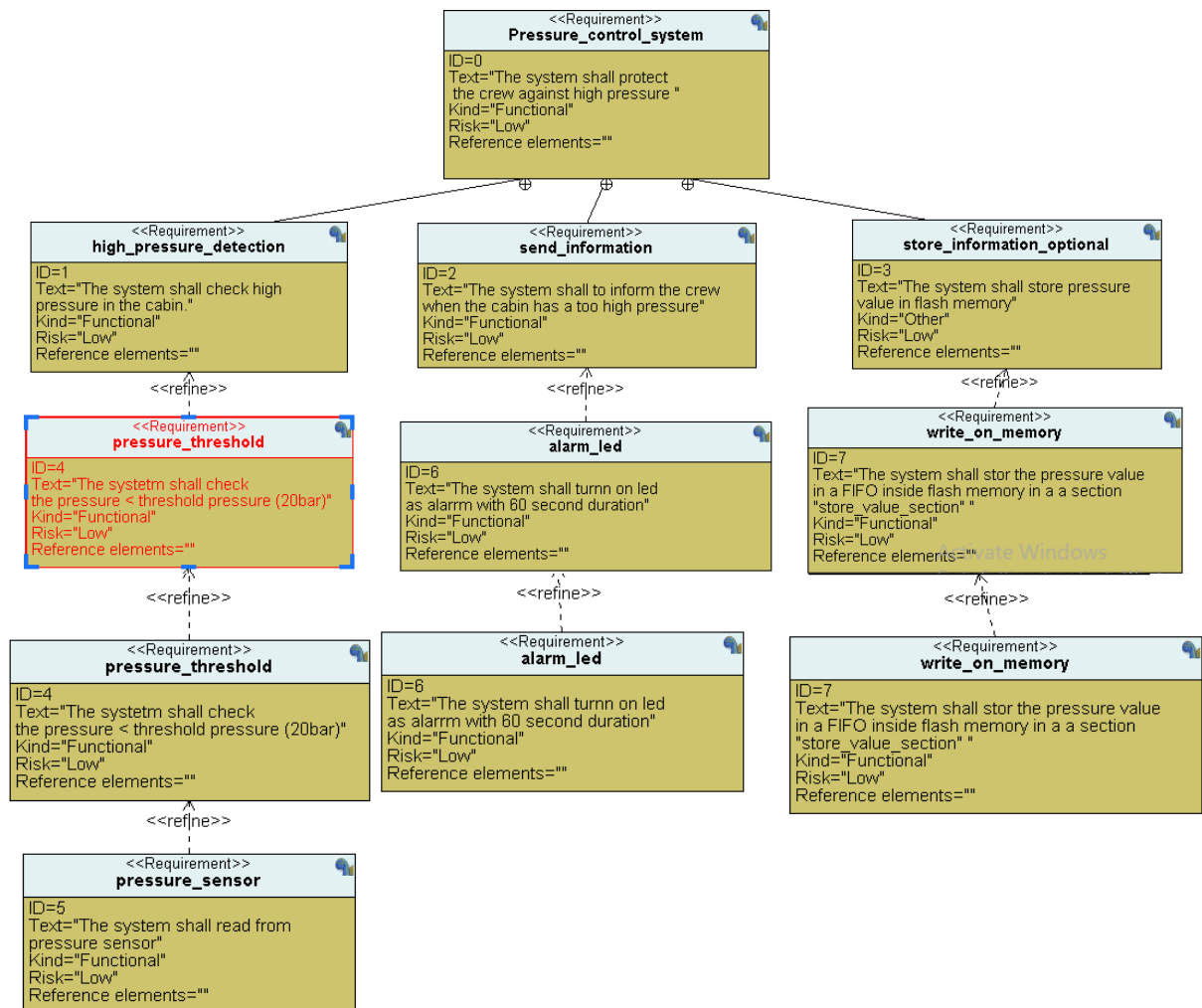
- The keep track of measured value option is not modeled in the first version.

## Method

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

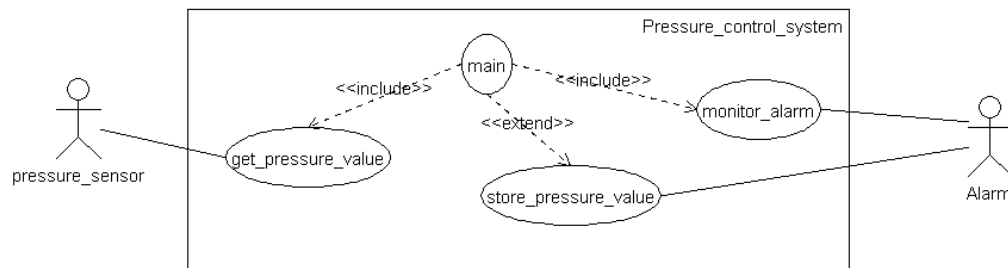
- Easy to set up.
- Productivity improvement.
- Time saving.
- Money saving.
- Improvement of the quality of the delivered product.

# Requirements

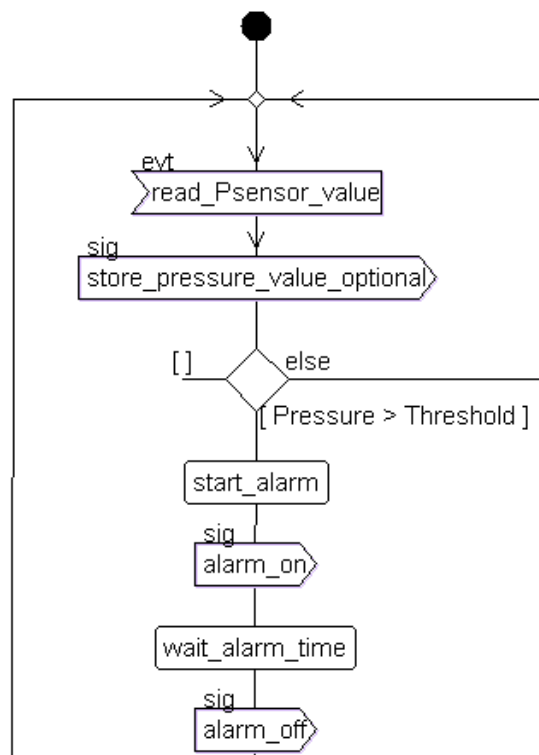


# System analysis

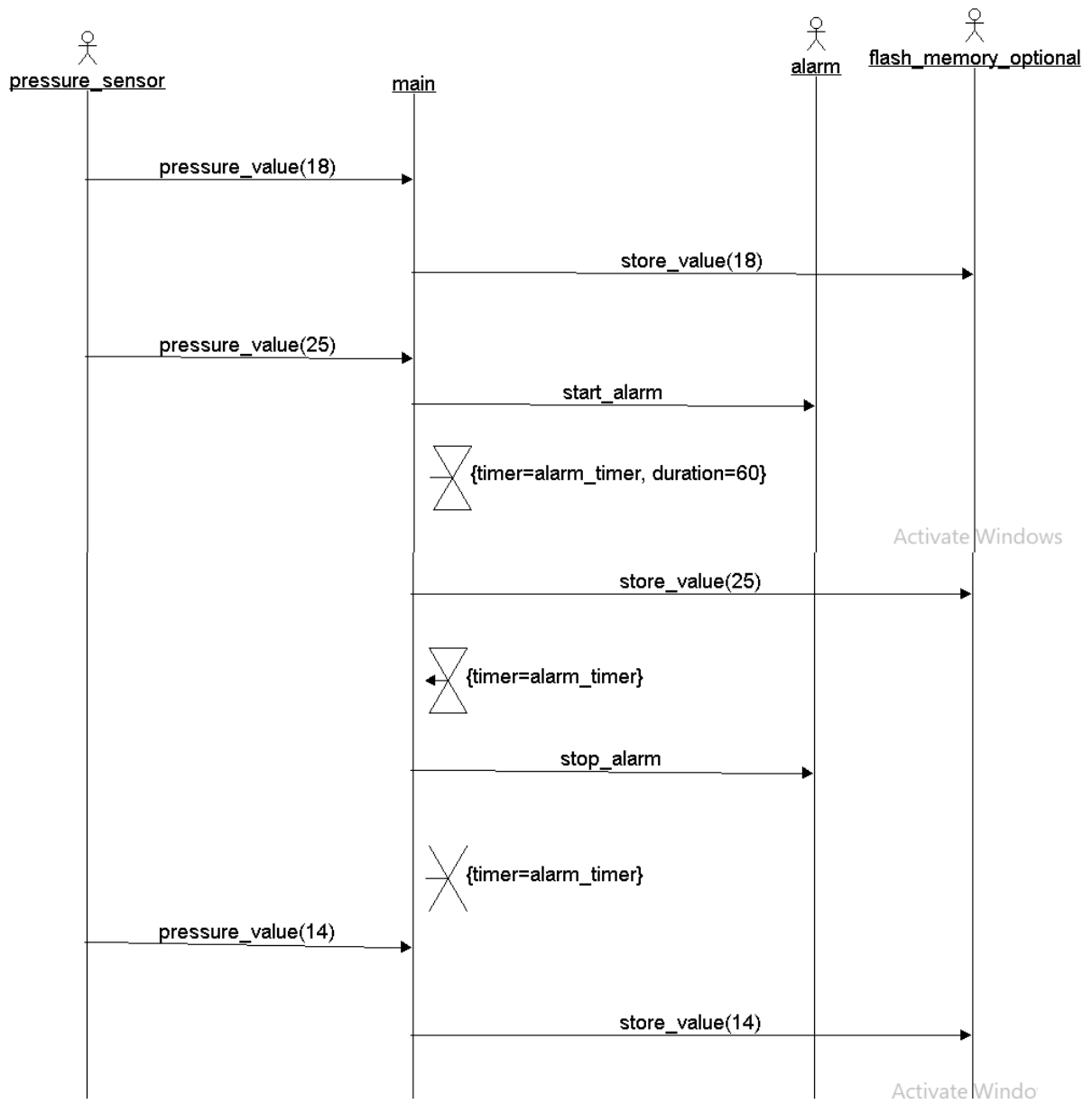
- Use case diagram.



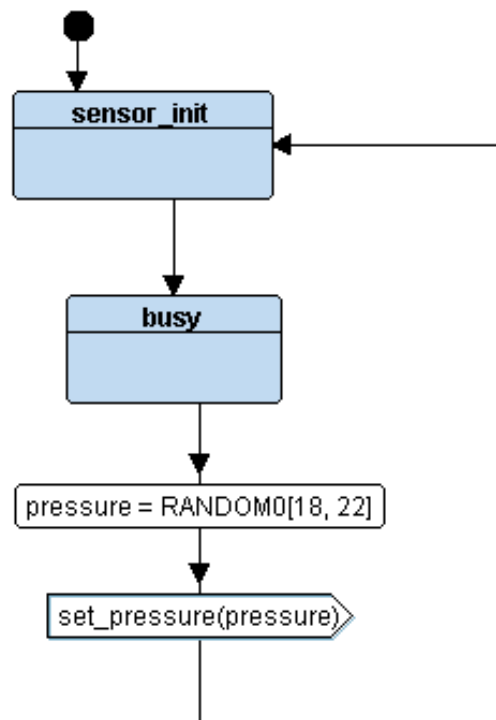
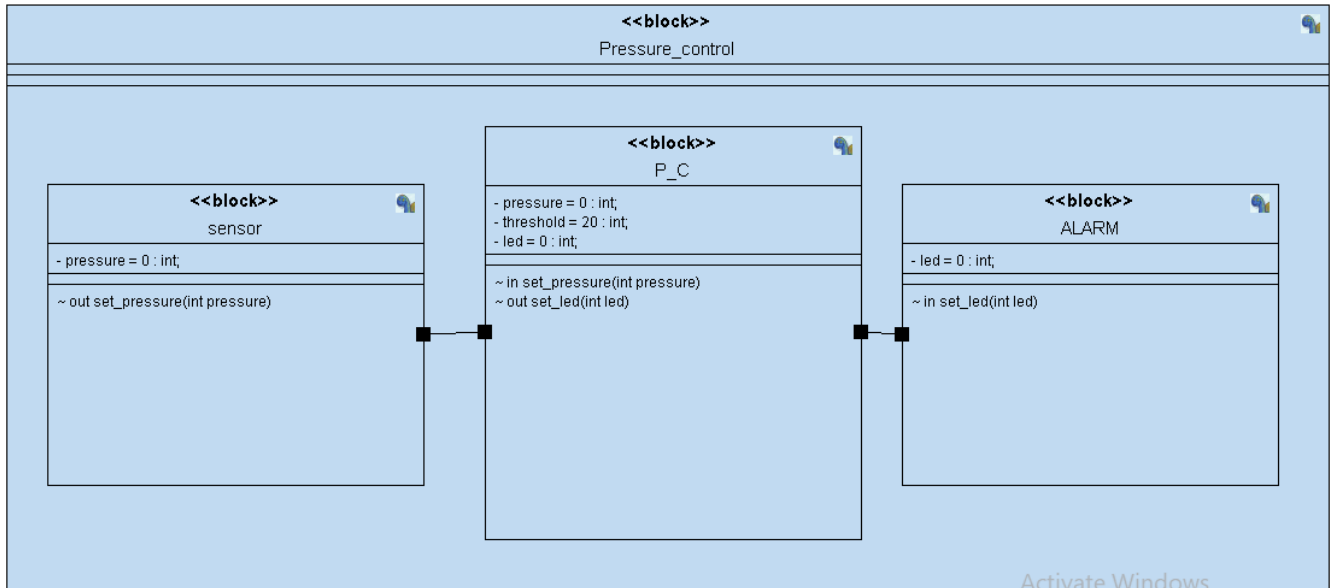
- Activity diagram.

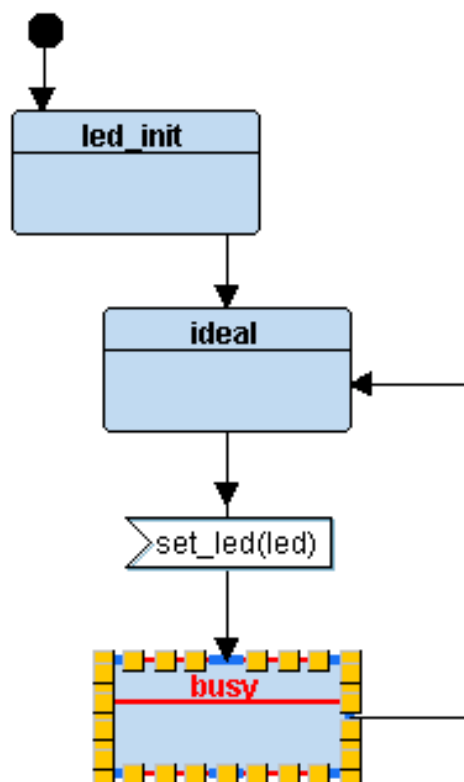
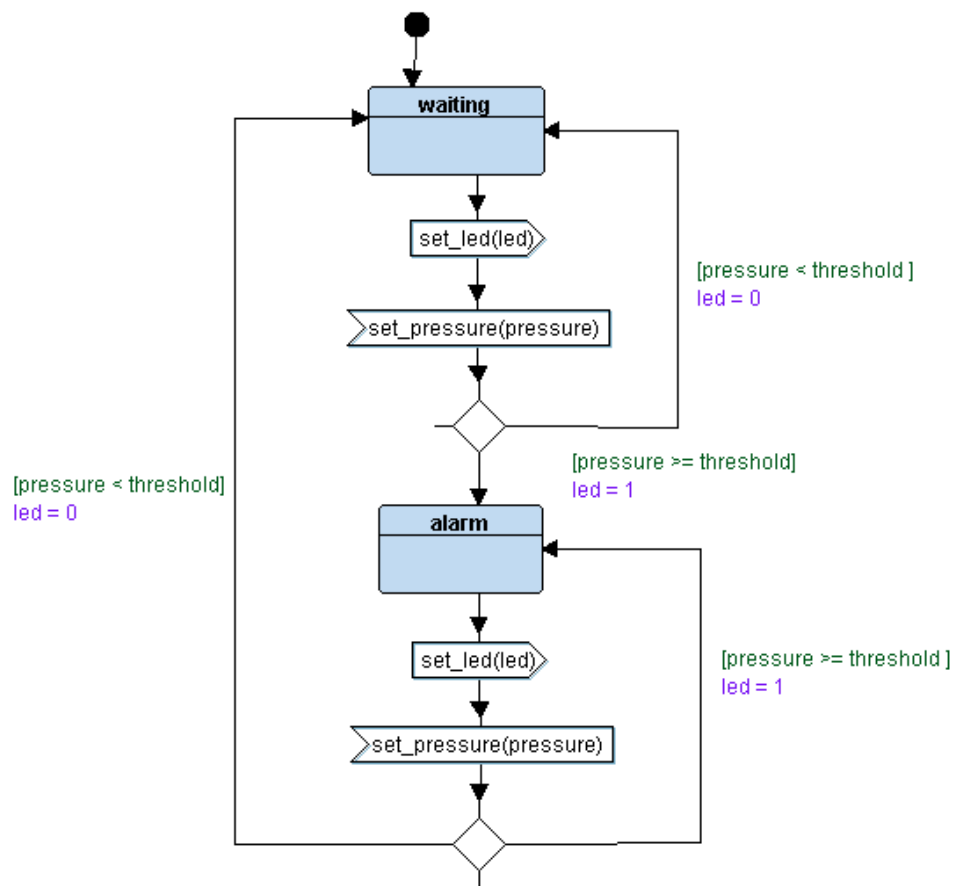


- Sequence diagram



# System design







## Implementation code

- (.c) & (.h) files

### - PS.c

```
6  #include "PS.h"
7
8  //variables
9  int PS_pressure = 0;
10 PS_enum PS_state_id;
11
12 void (*PS_state)();
13
14 void PS_init(){
15     //printf("====PS_INIT====\n");
16 }
17
18 STATE_def(PS_busy){
19     //State Name
20     PS_state_id = PS_busy;
21
22     //State Action
23     PS_pressure = getPressureVal();
24     //printf("PS_busy state: pressure = %d \n", PS_pressure);
25     PS_set_pressure(PS_pressure);
26     PS_state = STATE(PS_busy);
27 }
28
29
```

### - PS.h

```
6
7  #ifndef PS_H_
8  #define PS_H_
9  #include "state.h"
10
11 //define state
12 typedef enum{
13     PS_busy
14 } PS_enum;
15 extern PS_enum PS_state_id;
16
17 void PS_init();
18
19 STATE_def(PS_busy);
20
21 extern void (*PS_state)();
22
23 #endif /* PS_H_ */
24
```

## - CP.c

```

6
7 #include "CP.h"
8 //variables
9 int CP_led = 0;
10 int CP_pressure = 0;
11 int CP_threshold = 20;
12
13 CP_enum CP_state_id;
14 void (*CP_state)();
15
16 void PS_set_pressure(int pressure){
17     CP_pressure = pressure;
18     (CP_pressure <= CP_threshold)? (CP_state = STATE(CP_waiting)) : (CP_state = STATE(CP_driving));
19 }
20
21 STATE_def(CP_waiting){
22     //State Name
23     CP_state_id = CP_waiting;
24     //State Action
25     CP_led = 1;
26     Set_Alarm_actuator(CP_led);
27 }
28 STATE_def(CP_driving){
29     //State Name
30     CP_state_id = CP_driving;
31     //State Action
32     CP_led = 0;
33     //ALARM_set_Led(CP_led);
34     Set_Alarm_actuator(CP_led);
35 }

```

## - CP.h

```

3 //
4
5
6 #ifndef CP_H_
7 #define CP_H_
8 #include "state.h"
9
10 typedef enum{
11     CP_waiting,
12     CP_driving
13 } CP_enum;
14 extern CP_enum CP_state_id;
15
16 STATE_def(CP_waiting);
17 STATE_def(CP_driving);
18
19 extern void (*CP_state)();
20
21 #endif /* CP_H_ */

```

- alarm.c

```
6 | #include "alarm.h"
7 | int ALARM_led = 0;
8 | ALARM_enum ALARM_state_id;
9 | void (*ALARM_state)();
10 | void ALARM_init(){
11 |     //printf("====ALARM_INIT==== \n");
12 | }
13 | void Set_Alarm_actuator(int led){
14 |     ALARM_led = led;
15 |     ALARM_state = STATE(ALARM_busy);
16 |     if (led == 1){
17 |         SET_BIT(GPIOA_ODR,13);
18 |     }
19 |     else if (led == 0){
20 |         RESET_BIT(GPIOA_ODR,13);
21 |     }
22 | }
23 | STATE_def(ALARM_ideal){
24 |     //State Name
25 |     ALARM_state_id = ALARM_ideal;
26 |     //State Action
27 | }
28 | STATE_def(ALARM_busy){
29 |     //State Name
30 |     ALARM_state_id = ALARM_busy;
31 |     //State Action
32 |     ALARM_state = STATE(ALARM_ideal);
33 | }
```

- alarm.h

```
8 | #ifndef ALARM_H_
9 | #define ALARM_H_
10 |
11 | #include "state.h"
12 |
13 | typedef enum{
14 |     ALARM_ideal,
15 |     ALARM_busy
16 | } ALARM_enum;
17 | extern ALARM_enum ALARM_state_id;
18 |
19 | void ALARM_init();
20 |
21 | STATE_def(ALARM_ideal);
22 | STATE_def(ALARM_busy);
23 |
24 | extern void (*ALARM_state)();
25 |
26 | #endif /* ALARM_H_ */
27 |
```

- driver.c

```
2
3  #include "driver.h"
4  #include <stdint.h>
5  #include <stdio.h>
6  void Delay(int nCount)
7  {
8      for(; nCount != 0; nCount--);
9  }
10
11 int getPressureVal(){
12     return (GPIOA_IDR & 0xFF);
13 }
14
15
16 void GPIO_INITIALIZATION (){
17     SET_BIT(APB2ENR, 2);
18     GPIOA_CRL &= 0xFF0FFFFFFF;
19     GPIOA_CRL |= 0x00000000;
20     GPIOA_CRH &= 0xFF0FFFFFFF;
21     GPIOA_CRH |= 0x22222222;
22 }
```

- driver.h

```
2
3  #include <stdint.h>
4  #include <stdio.h>
5
6  #define SET_BIT(ADDRESS,BIT)  ADDRESS |= (1<<BIT)
7  #define RESET_BIT(ADDRESS,BIT) ADDRESS &= ~(1<<BIT)
8  #define TOGGLE_BIT(ADDRESS,BIT) ADDRESS ^= (1<<BIT)
9  #define READ_BIT(ADDRESS,BIT) ((ADDRESS) & (1<<(BIT)))
10
11
12 #define GPIO_PORTA 0x40010800
13 #define BASE_RCC   0x40021000
14
15 #define APB2ENR    *(volatile uint32_t *) (BASE_RCC + 0x18)
16
17 #define GPIOA_CRL  *(volatile uint32_t *) (GPIO_PORTA + 0x00)
18 #define GPIOA_CRH  *(volatile uint32_t *) (GPIO_PORTA + 0x04)
19 #define GPIOA_IDR  *(volatile uint32_t *) (GPIO_PORTA + 0x08)
20 #define GPIOA_ODR  *(volatile uint32_t *) (GPIO_PORTA + 0x0C)
21
22 void Delay(int nCount);
23 int getPressureVal();
24 void Set_Alarm_actuator(int i);
25 void GPIO_INITIALIZATION ();
26
```

- main.c

```
5 //
6 #include "CP.h"
7 #include "PS.h"
8 #include "alarm.h"
9 void setup(){
10
11     //Initialize Blocks
12     PS_init();
13     ALARM_init();
14
15     //Set State pointers to their corresponding block
16     CP_state = STATE(CP_waiting);
17     PS_state = STATE(PS_busy);
18     ALARM_state = STATE(ALARM_ideal);
19 }
20 void main(){
21     GPIO_INITIALIZATION();
22     setup();
23     while(1){
24         //Call state pointer of each block
25         PS_state();
26         CP_state();
27         ALARM_state();
28
29         //Delay Loop
30         Delay(2000);
31     }
32 }
33
```

- state.h

```
6
7 #ifndef STATE_H_
8 #define STATE_H_
9 #include "driver.h"
10 #include <stdio.h>
11 #include <stdlib.h>
12
13 //State function generator
14 #define STATE_def(_stateFun_) void State_##_stateFun_()
15 #define STATE(_stateFun_) State_##_stateFun_
16
17 //Triggered signals interface
18 void PS_set_pressure(int pressure);
19 void ALARM_set_led(int led);
20
21
22
23 #endif /* STATE_H_ */
```

- **Makefile**

```

1 CC      =arm-none-eabi-
2 CFLAGS  =-mcpu=cortex-m4 -mthumb -gdwarf-2 -g
3 INCS    =-I .
4 LIBS    =
5 SRC     =$(wildcard *.c)
6 OBJ     =$(SRC:.c=.o)
7 ASM     =$(wildcard *.s)
8 ASMOBJ  =$(ASM:.s=.o)
9 LINKER  =$(wildcard *.ld)
10 Project_Name = pressure_control_system
11
12 all: $(Project_Name).bin
13     @echo "====mission completed===="
14
15 %.o: %.c
16     $(CC)gcc.exe $(CFLAGS) $(INCS) -c $< -o $@
17
18 $(Project_Name).elf : $(OBJ) $(ASMOBJ)
19     $(CC)ld.exe -T $(LINKER) $(OBJ) $(ASMOBJ) $(LIBS) -Map=Map_File.map -o $@ -Map=Map_file.map
20     cp $(Project_Name).elf $(Project_Name).axf
21
22 $(Project_Name).bin : $(Project_Name).elf
23     $(CC)objcopy.exe -O binary $< $@
24
25 clean_all:
26     rm *.o *.elf *.bin *.map
27
28 clean:
29     rm *.bin *.elf

```

- **Linkerscript.ld**

```

2
3 MEMORY {
4     Flash(RX) : ORIGIN = 0x00000000, LENGTH = 512M
5     SRAM(RWX) : ORIGIN = 0x20000000, LENGTH = 512M
6 }
7 SECTIONS {
8     .text : {
9         *(.vectors*)
10        . = ALIGN(4) ;
11        *(.text*)
12        . = ALIGN(4) ;
13        *(.rodata*)
14        . = ALIGN(4) ;
15        _E_TEXT = .;
16    }> Flash
17
18    .data : {
19        _S_DATA = .;
20        *(.data*)
21        . = ALIGN(4) ;
22        _E_DATA = .;
23    }> SRAM AT> Flash
24
25    .bss : {
26        _S_bss = .;
27        *(.bss*)
28        . = ALIGN(4) ;
29        _E_bss = .;
30    }> SRAM
31 }

```

- Startup.c

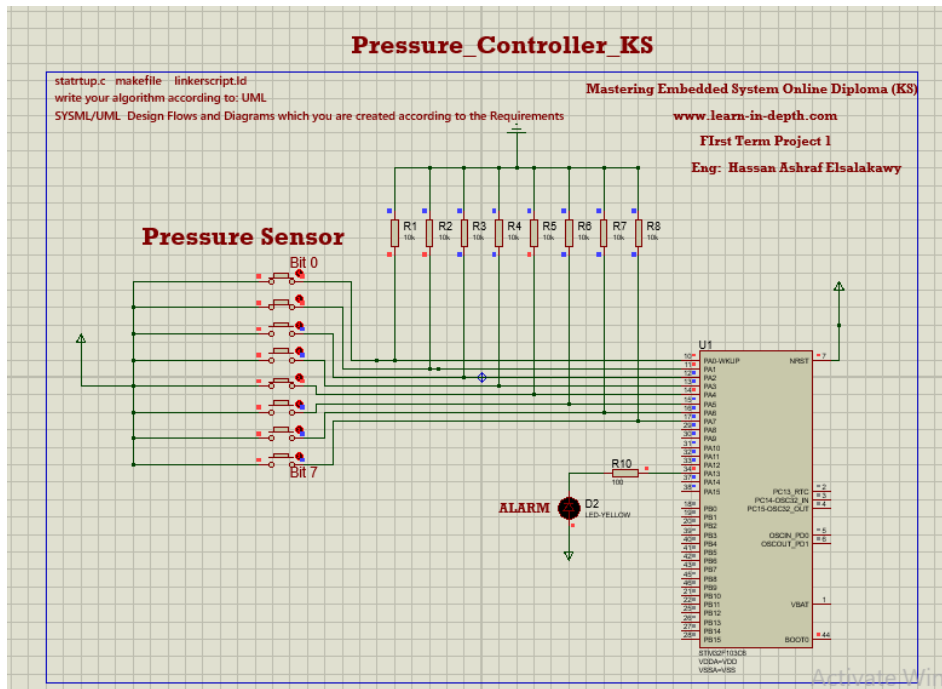
```

3 | #include <stdint.h>
4 | extern int main (void);
5 | extern unsigned int _E_TEXT;
6 | extern unsigned int _S_DATA;
7 | extern unsigned int _E_DATA;
8 | extern unsigned int _S_bss;
9 | extern unsigned int _E_bss;
10 |
11 | void reset_handler(){
12 |     //copy .data from flash to ram
13 |     unsigned int Data_size = (unsigned char*)&_E_DATA - (unsigned char*)&_S_DATA;
14 |     unsigned char* P_Src = (unsigned char*)&_E_TEXT;
15 |     unsigned char* P_Dst = (unsigned char*)&_S_DATA;
16 |     int i;
17 |     for(i=0;i<Data_size;i++) {
18 |         *P_Dst = *P_Src;
19 |         P_Dst++;
20 |         P_Src++;
21 |     }
22 |     // Init bss section in Ram with 0
23 |     unsigned int bss_size = (unsigned char*)&_E_bss - (unsigned char*)&_S_bss;
24 |     P_Dst = (unsigned char*)&_S_bss;
25 |
26 |     for(i=0;i<bss_size;i++) {
27 |         *P_Dst = (unsigned char)0;
28 |         P_Dst++;
29 |     }
30 |     main();
31 | }
32 |
33 |
34 | void default_handler (){
35 |     reset_handler();
36 | }
37 |
38 | void NMI_handler () __attribute__((weak,alias("default_handler")));;
39 | void H_fault_handler () __attribute__((weak,alias("default_handler")));;
40 |
41 |
42 |
43 | static unsigned long stack_top[256]; // 1024byte=256*4
44 |
45 | void (* const fun_to_vectors []) () __attribute__((section(".vectors"))) =
46 | {
47 |     (void (*)()) ((unsigned long)stack_top + sizeof(stack_top)),
48 |     &reset_handler,
49 |     &NMI_handler,
50 |     &H_fault_handler
51 | };
52 |
53 |

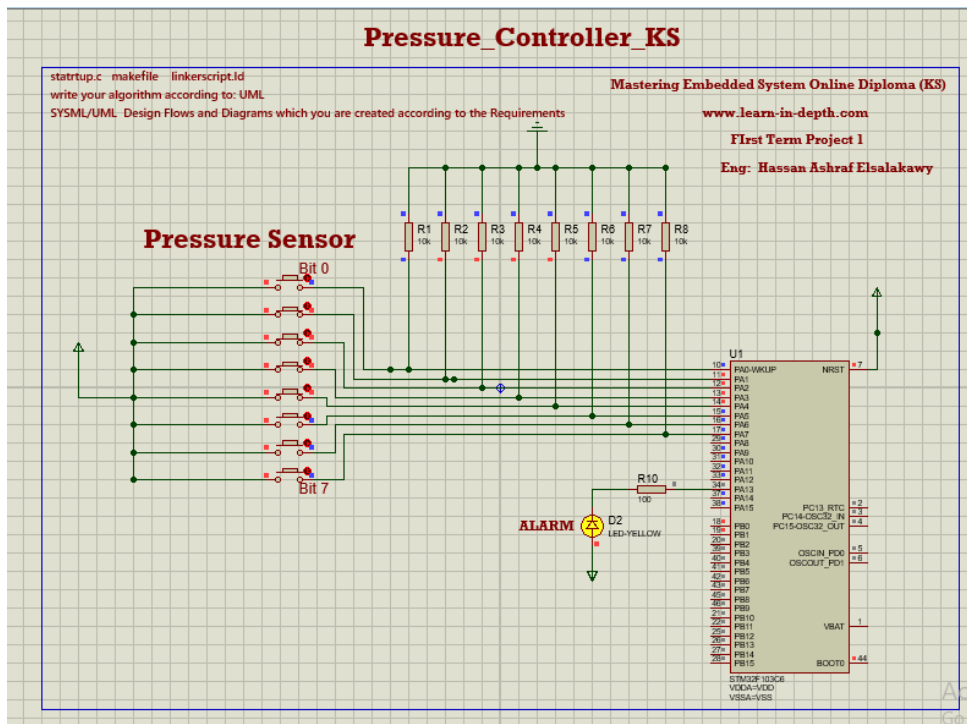
```

# Simulation

- Pressure = 19 bar < threshold (20 bar)



- Pressure = 30 bar > threshold (20 bar)





# SW analysis

- Mapfile

```

2 Allocating common symbols
3 Common symbol      size      file
4
5 ALARM_state_id      0x1      alarm.o
6 CP_state             0x4      CP.o
7 CP_state_id          0x1      CP.o
8 ALARM_state          0x4      alarm.o
9 PS_state             0x4      PS.o
10 PS_state_id          0x1      PS.o
11
12 Memory Configuration
13
14 Name              Origin              Length              Attributes
15 Flash              0x00000000          0x20000000          xr
16 SRAM                0x20000000          0x20000000          xrw
17 *default*          0x00000000          0xffffffff
18
19 Linker script and memory map
20
21
22 .text              0x00000000          0x3e0
23 *(.vectors*)
24 .vectors            0x00000000          0x10 startup.o
25                      0x00000000          fun_to_vectors
26                      0x00000010          . = ALIGN (0x4)
27
28 *(.text*)
29 .text              0x00000010          0xc4 CP.o
30                      0x00000010          PS_set_pressure
31                      0x0000006c          State_CP_waiting
32                      0x000000a0          State_CP_driving
33 .text              0x000000d4          0x54 PS.o
34                      0x000000d4          PS_init
35                      0x000000e0          State_PS_busy
36 .text              0x00000128          0xc0 alarm.o
37                      0x00000128          ALARM_init
38                      0x00000134          Set_Alarm_actuator
39                      0x000001a4          State_ALARM_ideal
40                      0x000001bc          State_ALARM_busy
41 .text              0x000001e8          0xbc driver.o
42                      0x000001e8          Delay
43                      0x0000020c          getPressureVal
44                      0x00000224          GPIO_INITIALIZATION
45 .text              0x000002a4          0x80 main.o
46                      0x000002a4          setup
47                      0x000002e8          main
48 .text              0x00000324          0xbc startup.o
49                      0x00000324          reset_handler
50                      0x000003d4          default_handler
51                      0x000003d4          H_fault_handler
52                      0x000003d4          NMI_handler
53                      0x000003e0          . = ALIGN (0x4)
54 *(.rodata*)
55                      0x000003e0          . = ALIGN (0x4)
56                      0x000003e0          _E_TEXT = .

```

```

57 .glue_7      0x000003e0      0x0
58 .glue_7      0x00000000      0x0 linker stubs
59
60 .glue_7t      0x000003e0      0x0
61 .glue_7t      0x00000000      0x0 linker stubs
62
63 .vfp11_veneer 0x000003e0      0x0
64 .vfp11_veneer 0x00000000      0x0 linker stubs
65
66 .v4_bx        0x000003e0      0x0
67 .v4_bx        0x00000000      0x0 linker stubs
68
69 .iplt         0x000003e0      0x0
70 .iplt         0x00000000      0x0 CP.o
71
72 .rel.dyn      0x000003e0      0x0
73 .rel.iplt     0x00000000      0x0 CP.o
74
75 .data         0x20000000      0x4 load address 0x000003e0
76             0x20000000      _S_DATA = .
77 *(.data*)
78 .data         0x20000000      0x4 CP.o
79             0x20000000      CP_threshold
80 .data         0x20000004      0x0 PS.o
81 .data         0x20000004      0x0 alarm.o
82 .data         0x20000004      0x0 driver.o
83 .data         0x20000004      0x0 main.o
84 .data         0x20000004      0x0 startup.o
85             0x20000004      . = ALIGN (0x4)
86             0x20000004      _E_DATA = .
87
88 .igot.plt     0x20000004      0x0 load address 0x000003e4
89 .igot.plt     0x00000000      0x0 CP.o
90
91 .bss          0x20000004      0x428 load address 0x000003e4
92             0x20000004      _S_bss = .
93 *(.bss*)
94 .bss          0x20000004      0x8 CP.o
95             0x20000004      CP_led
96             0x20000008      CP_pressure
97 .bss          0x2000000c      0x4 PS.o
98             0x2000000c      PS_pressure
99 .bss          0x20000010      0x4 alarm.o
100            0x20000010      ALARM_led
101 .bss          0x20000014      0x0 driver.o
102 .bss          0x20000014      0x0 main.o
103 .bss          0x20000014      0x400 startup.o
104            0x20000414      . = ALIGN (0x4)
105            0x20000414      _E_bss = .
106 COMMON        0x20000414      0x5 CP.o
107             0x20000414      CP_state
108             0x20000418      CP_state_id
109 *fill*        0x20000419      0x3
110 COMMON        0x2000041c      0x5 PS.o
111             0x2000041c      PS_state
112             0x20000420      PS_state_id
113 *fill*        0x20000421      0x3
114 COMMON        0x20000424      0x8 alarm.o
115             0x20000424      ALARM_state_id
116             0x20000428      ALARM_state

```

```

117 LOAD CP.o
118 LOAD PS.o
119 LOAD alarm.o
120 LOAD driver.o
121 LOAD main.o
122 LOAD startup.o
123 OUTPUT(pressure_control_system.elf elf32-littlearm)
124 .comment      0x00000000      0x11
125 .comment      0x00000000      0x11 CP.o
126              0x00000000      0x12 (size before relaxing)
127 .comment      0x00000000      0x12 PS.o
128 .comment      0x00000000      0x12 alarm.o
129 .comment      0x00000000      0x12 driver.o
130 .comment      0x00000000      0x12 main.o
131 .comment      0x00000000      0x12 startup.o
132
133 .ARM.attributes
134              0x00000000      0x33
135 .ARM.attributes
136              0x00000000      0x33 CP.o
137 .ARM.attributes
138              0x00000033      0x33 PS.o
139 .ARM.attributes
140              0x00000066      0x33 alarm.o
141 .ARM.attributes
142              0x00000099      0x33 driver.o
143 .ARM.attributes
144              0x000000cc      0x33 main.o
145 .ARM.attributes
146              0x000000ff      0x33 startup.o

```

- Symbols table

```

mit5_final/project_1/source_code (master)
$ 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

```