

# Mastering Embedded System Online Diploma

[www.learn-in-depth.com](http://www.learn-in-depth.com)

First Term (Final Project 1)

Eng. Ebram Edward Fouad Habib

My Profile:

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

## List of Contents:

- Project Description
- Assumptions
- Requirements Diagram
- System Analysis:
  - 1. Use Case Diagram
  - 2. Activity Diagram
  - 3. Sequence Diagram
- System Design (Modules with its own state machines)
- Implementation of each module in C
- .c & .h for each module (An Image for each file.c & file.h with the Corresponding state machine)
- MakeFile
- Startup.c
- Linker\_Script.ld
- SW analysis .map file & symbols table & Section tables
- Proteus Simulation

# Pressure Controller

---

## Project Description:

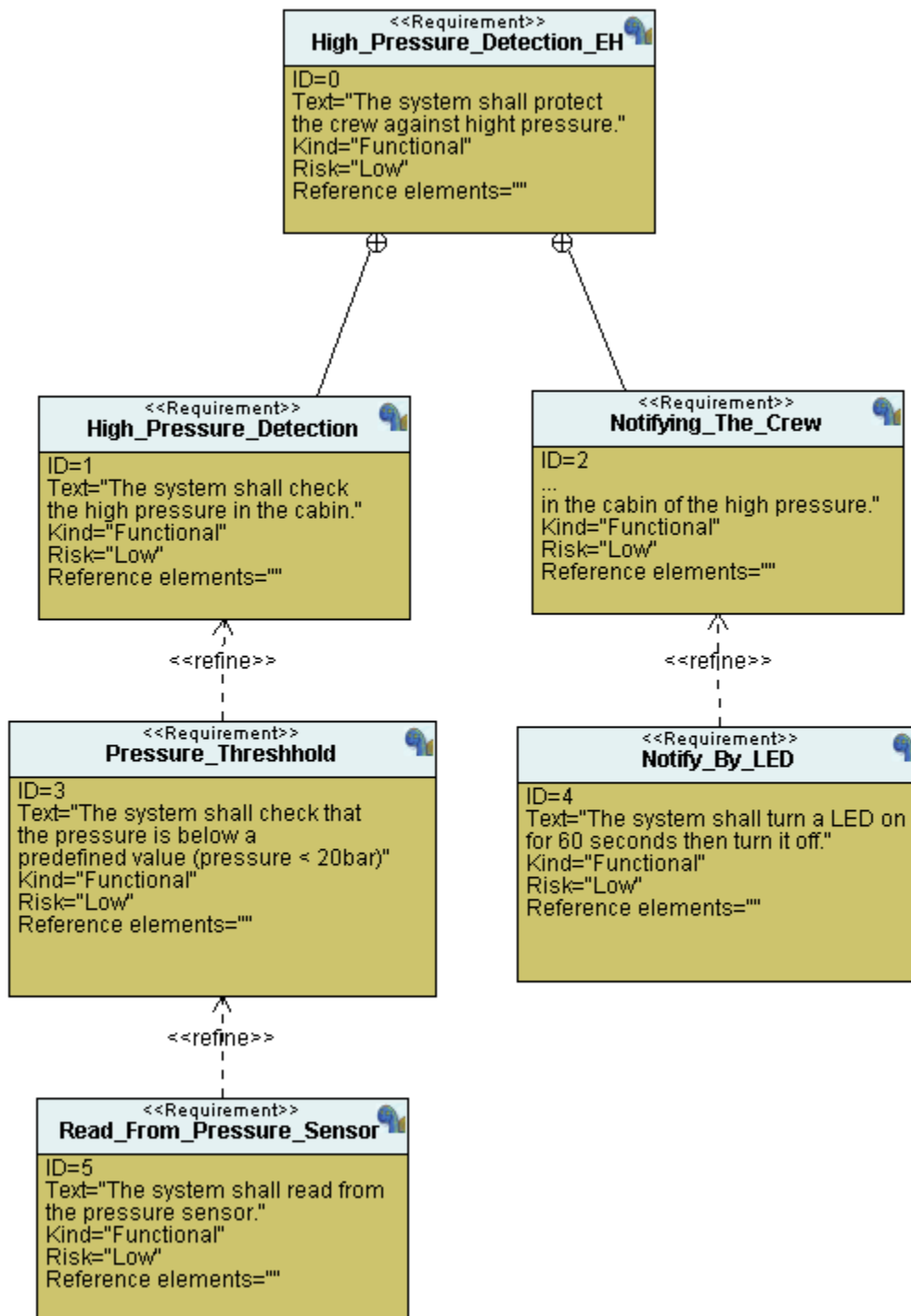
- A pressure controller informs the crew of a cabin with an alarm when the pressure exceeds 20 bars in the cabin.
- The alarm duration equals 60 seconds.

## Assumptions:

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

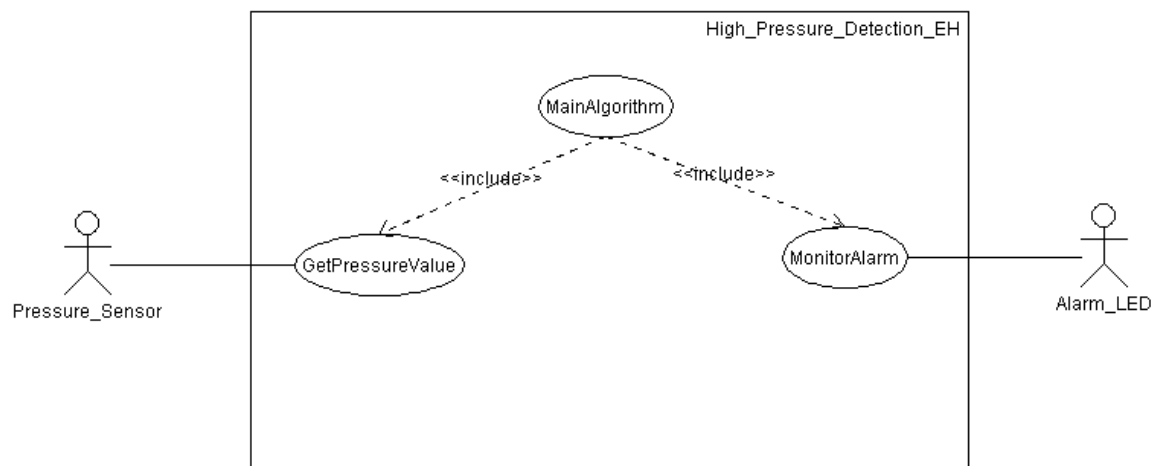


## Requirements Diagram:

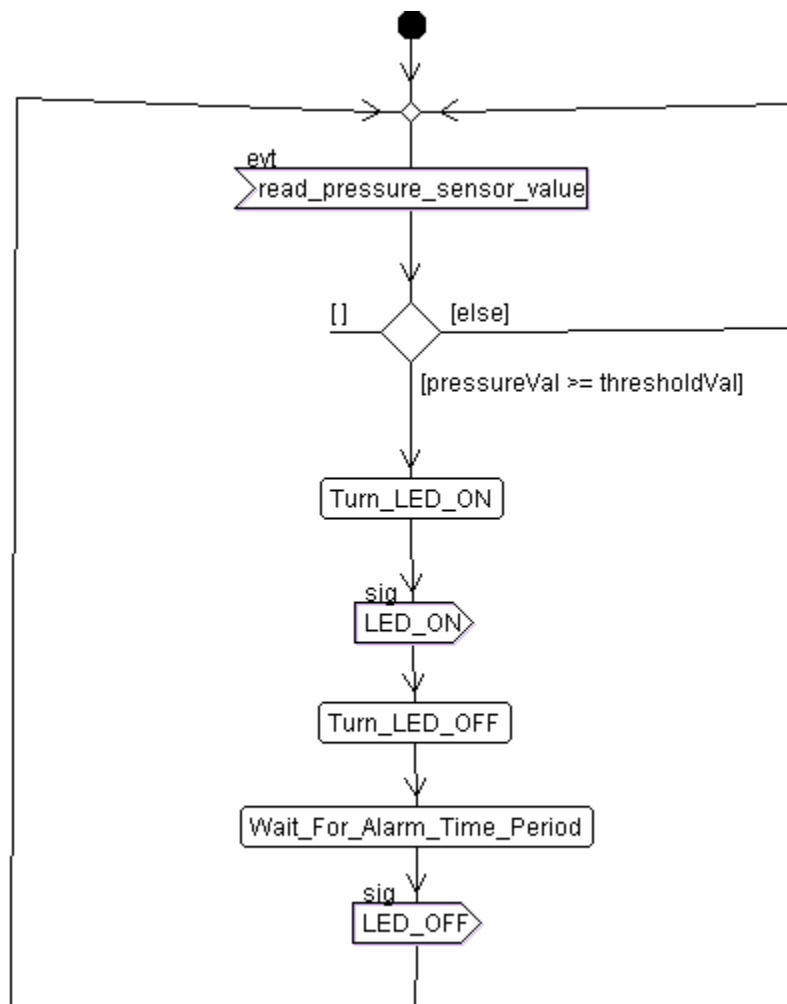


## System Analysis:

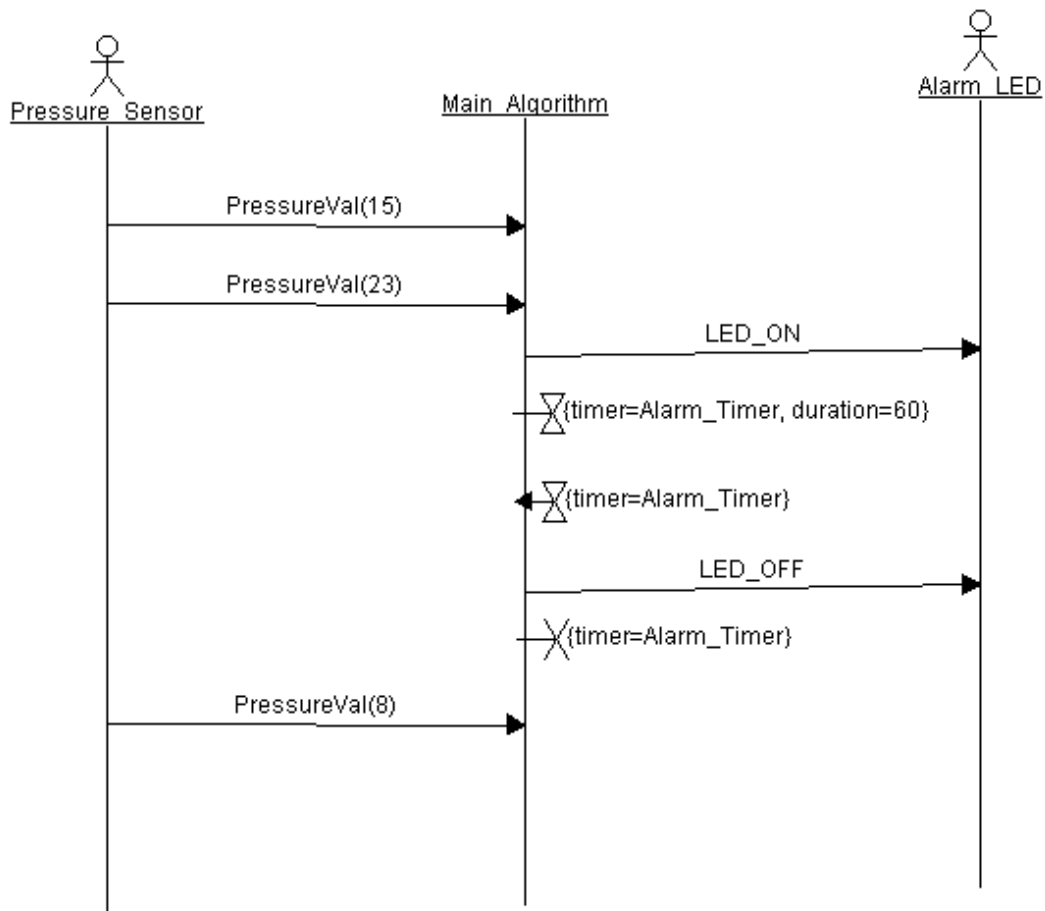
### 1. Use Case Diagram



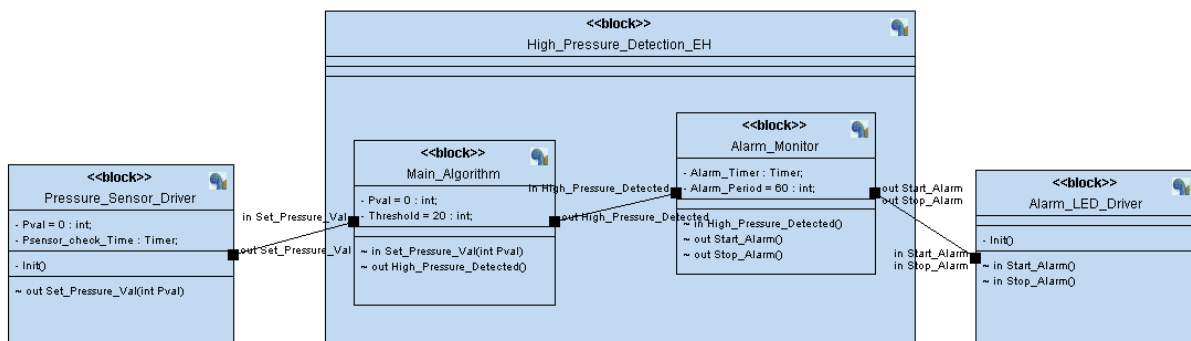
## 2. Activity Diagram



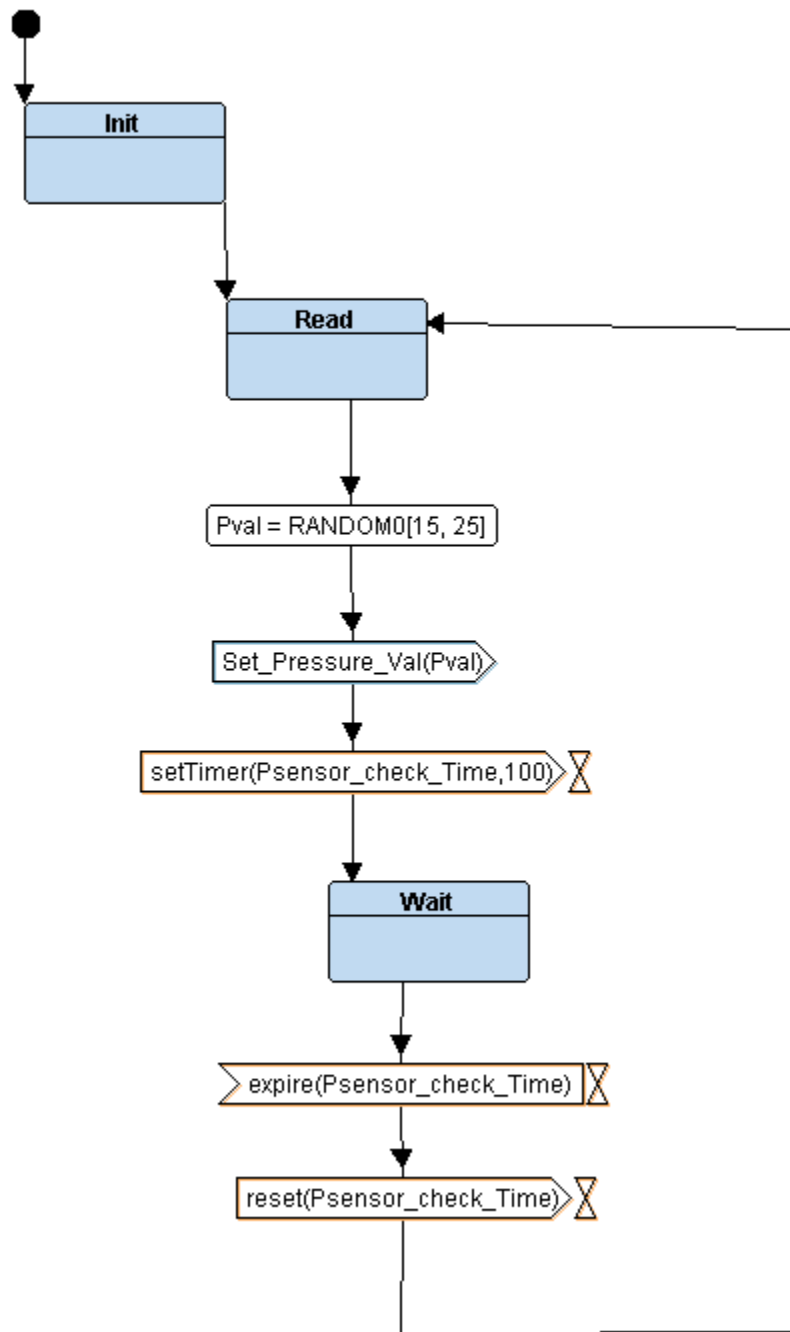
### 3. Sequence Diagram



### System Design (Modules with its own state machines)

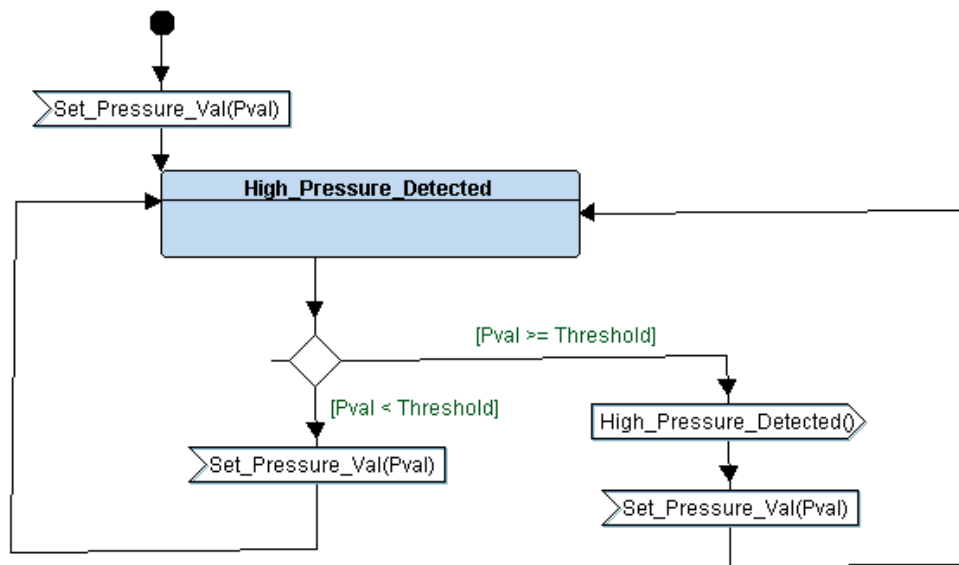


## 1-Pressure Sensor State Diagram:

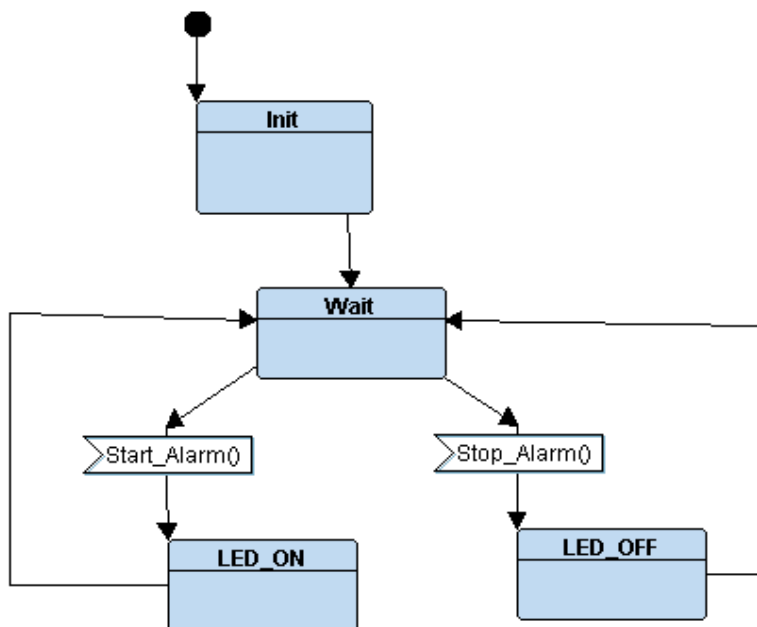




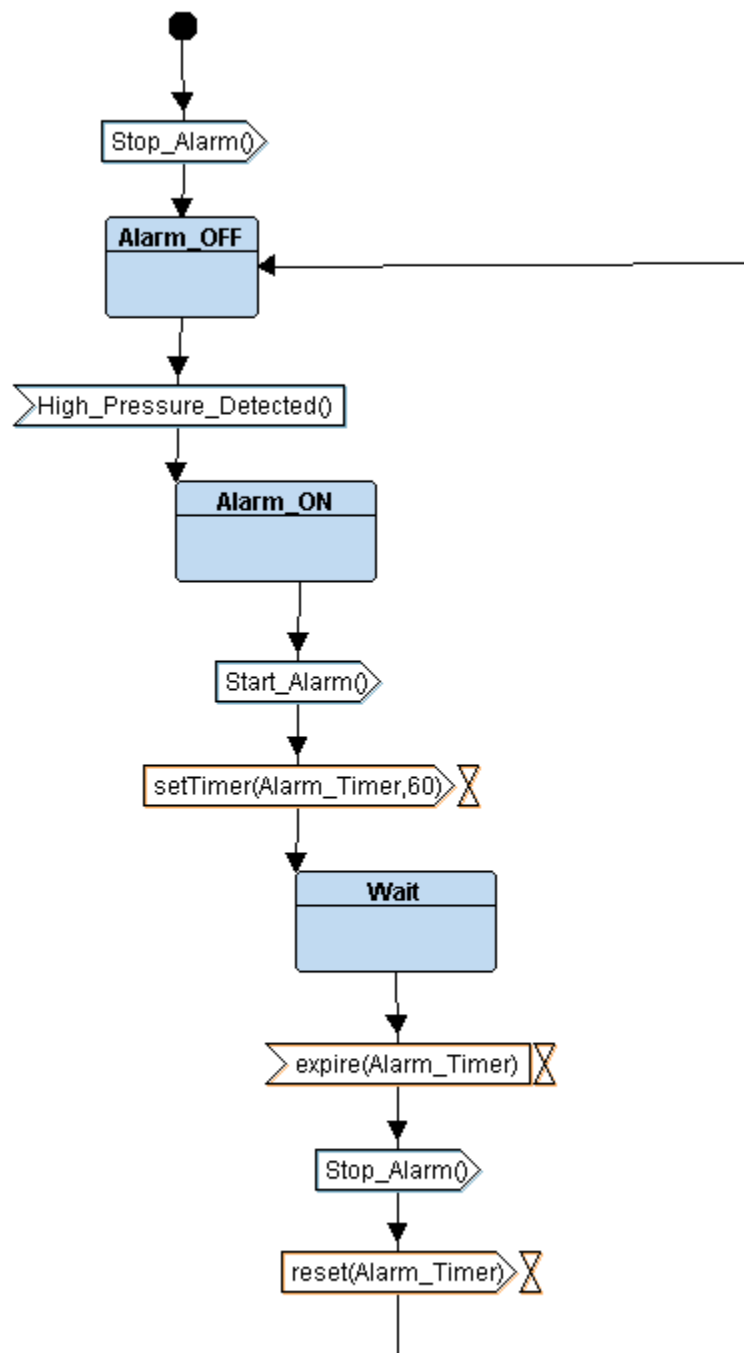
### 2-Alarm LED State Diagram:



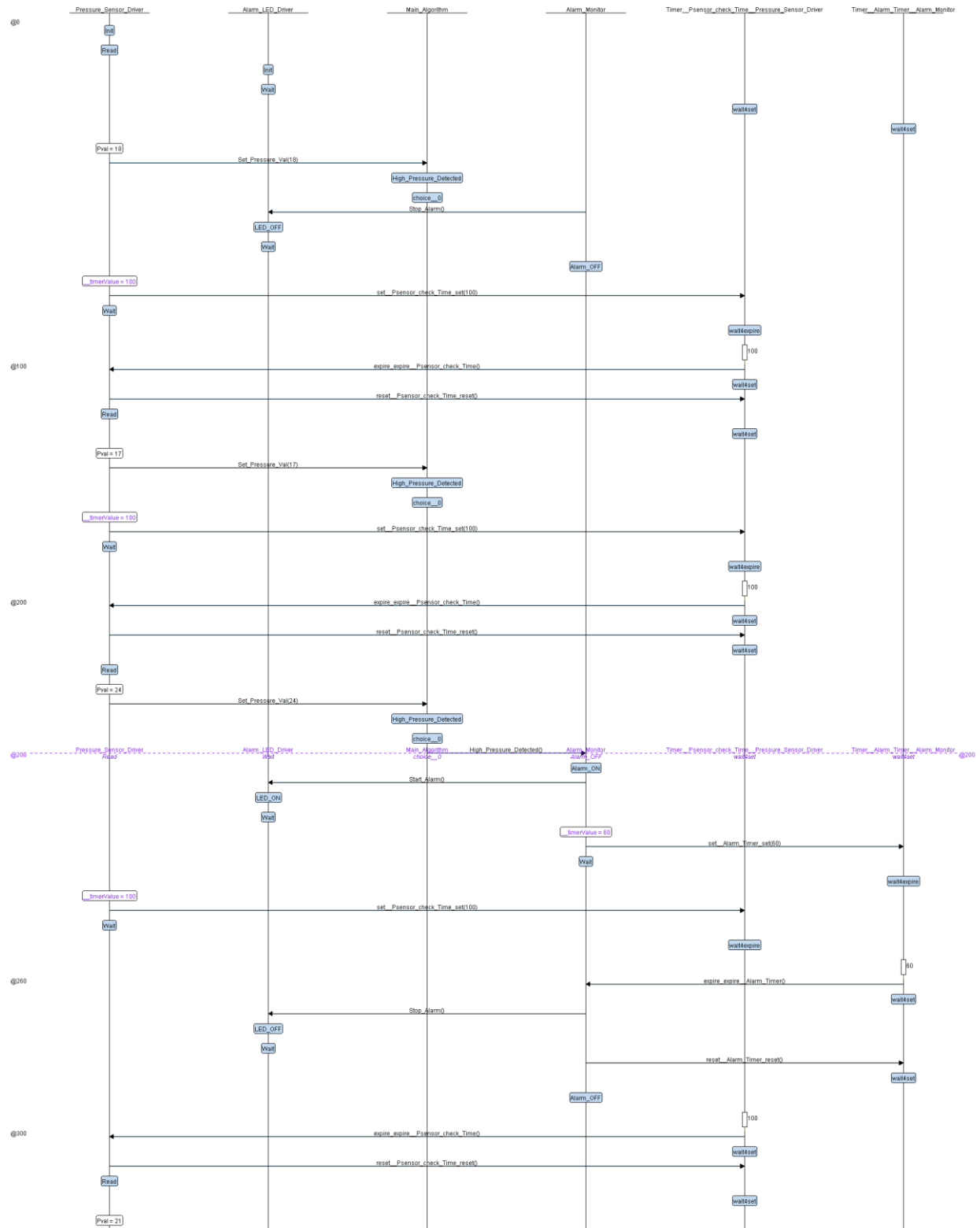
### 3-Alarm LED State Diagram:



#### 4- Alarm Monitor State Diagram:



## - System Design Simulation:



## Implementation of each module in C:

(An Image for each file.c & file.h with the Corresponding state machine)

### main.c

```
1  /*
2  * main.c
3  *
4  * Created on: Nov 26, 2022
5  * Author: Ebram Habib
6  */
7
8  #include "Platform_Types.h"
9  #include "Util.h"
10
11 #include "GPIO_Driver.h"
12 #include "Pressure_Sensor_Driver.h"
13 #include "Alarm_LED_Driver.h"
14 #include "Alarm_Monitor.h"
15 #include "Main_Algorithm.h"
16
17 void (*PS_state)() = STATE(PS_INIT);
18 void (*ALARM_LED_state)() = STATE(ALARM_LED_INIT);
19 void (*ALARM_MONITOR_state)() = STATE(ALARM_MONITOR_ALARM_OFF);
20 void (*MA_state)() = STATE(MA_HIGH_PRESSURE);
21
22 int main(void)
23 {
24     // System Initialization
25     GPIO_Init();
26
27     // Run The Program Forever
28     while (1)
29     {
30         PS_state();
31         ALARM_LED_state();
32         ALARM_MONITOR_state();
33         MA_state();
34     }
35
36     return 0;
37 }
38
```

## State.h

```
1  /*
2  * State.h
3  *
4  * Created on: Nov 26, 2022
5  * Author: Ebram Habib
6  */
7
8  #ifndef STATE_H_
9  #define STATE_H_
10
11 #include "GPIO_Driver.h"
12
13 // State function generation
14 #define STATE_DEFINE(_statFUN_) void ST_##_statFUN_()
15 #define STATE(_statFUN_) ST_##_statFUN_
16
17 //States Connections
18 // Pressure Sensor =====> Main Algorithm
19 uint32_t PS_get_pressure_value(void);
20
21 // Alarm LED =====> Alarm Monitor
22 void ALARM_LED_start_alarm(void);
23
24 // Alarm LED =====> Alarm Monitor
25 void ALARM_LED_stop_alarm(void);
26
27 // Main Algorithm =====> Alarm Monitor
28 uint32_t MA_high_pressure_detected(void);
29
30 #endif /* STATE_H_ */
31
```

## Util.h

```
1  /*
2  * Util.h
3  *
4  * Created on: Nov 26, 2022
5  * Author: Ebram Habib
6  */
7
8  #ifndef UTIL_H_
9  #define UTIL_H_
10
11 #define GET_BIT(reg,n) (((reg)>>(n))&1)
12 #define SET_BIT(reg,n) (reg|=(1<<n))
13 #define CLR_BIT(reg,n) (reg&=~(1<<n))
14 #define TOGGLE_BIT(reg,n) (reg^=(1<<n))
15
16 #endif /* UTIL_H_ */
17
```

## GPIO Driver.h

```
1  /*
2  * GPIO_Driver.h
3  *
4  * Created on: Nov 26, 2022
5  * Author: Ebram Habib
6  */
7
8  #ifndef GPIO_DRIVER_H_
9  #define GPIO_DRIVER_H_
10
11  #include "Platform_Types.h"
12  #include "Util.h"
13
14  #define GPIO_PORTA 0x40010800
15  #define BASE_RCC 0x40021000
16
17  #define APB2ENR (*(vuint32_t *) (BASE_RCC + 0x18))
18
19  #define GPIOA_CRL (*(vuint32_t *) (GPIO_PORTA + 0x00))
20  #define GPIOA_CRH (*(vuint32_t *) (GPIO_PORTA + 0x04))
21  #define GPIOA_IDR (*(vuint32_t *) (GPIO_PORTA + 0x08))
22  #define GPIOA_ODR (*(vuint32_t *) (GPIO_PORTA + 0x0C))
23
24  void GPIO_Init (void);
25  void GPIO_Delay(uint32_t nCount);
26  uint32_t GPIO_Get_Pressure_Value(void);
27  void GPIO_Set_Alarm_LED(uint32_t i);
28
29  #endif /* GPIO_DRIVER_H_ */
30
```

## GPIO Driver.c

```
1  /*
2   * GPIO_Driver.c
3   *
4   * Created on: Nov 26, 2022
5   * Author: Ebram Habib
6   */
7
8
9  #include "GPIO_Driver.h"
10 #include "Util.h"
11
12 void GPIO_Delay(uint32_t nCount)
13 {
14     for(; nCount != 0; nCount--);
15 }
16
17 uint32_t GPIO_Get_Pressure_Value(void)
18 {
19     return (GPIOA_IDR & 0xFF);
20 }
21
22 void GPIO_Set_Alarm_LED(uint32_t i)
23 {
24     if (i == 1)
25     {
26         SET_BIT(GPIOA_ODR, 13);
27     }
28     else if (i == 0)
29     {
30         CLR_BIT(GPIOA_ODR, 13);
31     }
32 }
33
34 void GPIO_Init (void)
35 {
36     SET_BIT(APB2ENR, 2);
37
38     GPIOA_CRL &= 0xFF0FFFFF;
39     GPIOA_CRL |= 0x00000000;
40     GPIOA_CRH &= 0xFF0FFFFF;
41     GPIOA_CRH |= 0x22222222;
42 }
43
```

## Startup.c

```
1  /*
2  * Startup.c
3  *
4  * Created on: Nov 26, 2022
5  * Author: Ebram Habib
6  */
7
8  #include "Platform_Types.h"
9
10 uint32_t _STACK_TOP;
11
12 extern int main(void);
13
14 void Reset_Hundler(void);
15
16 void Default_Hundler()
17 {
18     Reset_Hundler();
19 }
20
21 void NMI_Handler(void)          __attribute__((weak, alias("Default_Hundler")));
22 void H_Fault_Handler(void)      __attribute__((weak, alias("Default_Hundler")));
23 void MM_Fault_Handler(void)     __attribute__((weak, alias("Default_Hundler")));
24 void Bus_Fault(void)            __attribute__((weak, alias("Default_Hundler")));
25 void Usage_Fault_Handler(void)  __attribute__((weak, alias("Default_Hundler")));
26
27 uint32_t vectors[] __attribute__((section(".vectors"))) = {
28     (uint32_t) &STACK_TOP,
29     (uint32_t) &Reset_Hundler,
30     (uint32_t) &NMI_Handler,
31     (uint32_t) &H_Fault_Handler,
32     (uint32_t) &MM_Fault_Handler,
33     (uint32_t) &Bus_Fault,
34     (uint32_t) &Usage_Fault_Handler
35 };
36
37 extern uint32_t _E_TEXT ; // End of text section
38 extern uint32_t _S_DATA ; // Start of data section
39 extern uint32_t _E_DATA ; // End of data section
40 extern uint32_t _S_BSS ; // Start of bss section
41 extern uint32_t _E_BSS ; // End of bss section
42
43 void Reset_Hundler (void)
44 {
45     //copy data from ROM to RAM
46     uint32_t DATA_Size = (uint8_t*)&_E_DATA - (uint8_t*)&_S_DATA;
47     uint8_t* P_src = (uint8_t*)&_E_TEXT ;
48     uint8_t* P_dst = (uint8_t*)&_S_DATA ;
49
50     for (uint32_t i = 0; i < DATA_Size; ++i)
51     {
52         *((uint8_t*)P_dst++) = *((uint8_t*)P_src++);
53     }
54
55     // init the .bss with zero
56     uint32_t BSS_Size = (uint8_t*)&_E_BSS - (uint8_t*)&_S_BSS;
57     P_dst = (uint8_t*)&_S_BSS;
58
59     for (uint32_t i = 0; i < BSS_Size; ++i)
60     {
61         *((uint8_t*)P_dst++) = (uint8_t)0;
62     }
63
64     //jump to main
65     main();
66 }
67
```



## MakeFile

```
1  #Prepared by Ebram Habib
2
3  CC=arm-none-eabi-
4  CFLAGS=-mcpu=cortex-m3 -gdwarf-2
5  INCS=-I .
6  LIBS=
7  SRC= $(wildcard *.c)
8  OBJ= $(SRC:.c=.o)
9  AS= $(wildcard *.s)
10 AsOBJ= $(AS:.s=.o)
11 Project_name= First_Term_First_Project_High_Pressure_System
12
13 all: $(Project_name).bin
14 |   @echo "=====Build is Done======"
15
16 %.o: %.c
17 |   $(CC)gcc.exe $(CFLAGS) $(INCS) -c $< -o $@
18
19 $(Project_name).elf: $(OBJ) $(AsOBJ)
20 |   $(CC)ld.exe -T Linker_Script.ld $(LIBS) $(OBJ) $(AsOBJ) -o $(Project_name).elf -Map=Map_file.map
21 |   cp $(Project_name).elf $(Project_name).axf
22
23 $(Project_name).bin: $(Project_name).elf
24 |   $(CC)objcopy.exe -O binary $< $@
25
26 clean_all:
27 |   rm *.o *.elf *.bin *.map *.axf *.elf.asm
28
29 clean:
30 |   rm *.elf *.bin
```

## Linker\_Script.ld

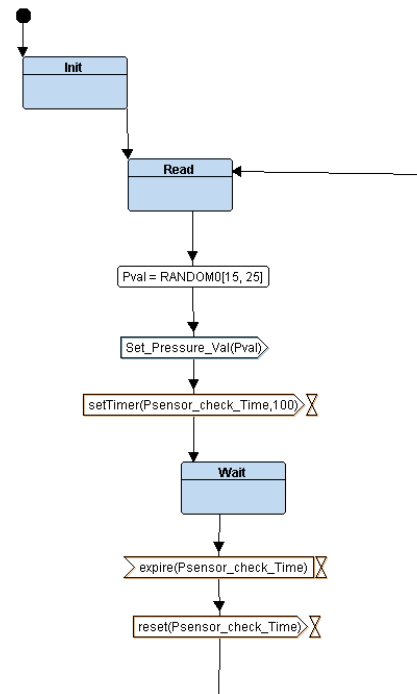
```
1  /* learn-in-depth diploma
2  First_Term_First_Project_High_Pressure_System
3  Eng: Ebram Habib
4  */
5
6  MEMORY
7  {
8      flash(RX)    : ORIGIN = 0X08000000, LENGTH = 512M
9      sram(RWX)    : ORIGIN = 0X20000000, LENGTH = 512M
10 }
11
12 SECTIONS
13 {
14     .text :
15     {
16         *(.vectors*)
17         *(.text*)
18         *(.rodata*)
19         _E_TEXT = . ; /* End of .text section*/
20     }>flash
21
22     .data :
23     {
24         _S_DATA = . ;
25         *(.data*)
26         . = ALIGN(4);
27         _E_DATA = . ;
28     }>sram AT>flash
29
30     .bss :
31     {
32         _S_BSS = . ;
33         *(.bss*)
34         . = ALIGN(4);
35         _E_BSS = . ;
36
37         . = ALIGN(4);
38         . = . + 0x1000 ;
39         _STACK_TOP = . ;
40     }>sram
41 }
```

## Pressure\_Sensor\_Driver

```

1  /*
2  * Pressure_Sensor_Driver.h
3  *
4  * Created on: Nov 26, 2022
5  * Author: Ebram Habib
6  */
7
8  #ifndef PRESSURE_SENSOR_DRIVER_H_
9  #define PRESSURE_SENSOR_DRIVER_H_
10
11 #include "State.h"
12
13 //Declare State Functions of Pressure Sensor
14 STATE_DEFINE(PRESSURE_SENSOR_DRIVER_H_);
15 STATE_DEFINE(PRESSURE_SENSOR_DRIVER_H_);
16 STATE_DEFINE(PRESSURE_SENSOR_DRIVER_H_);
17
18 //State Pointer to function
19 extern void (*PS_state)();
20
21 #endif /* PRESSURE_SENSOR_DRIVER_H_ */
22
23
24 1  /*
25  * Pressure_Sensor_Driver.c
26  *
27  * Created on: Nov 26, 2022
28  * Author: Ebram Habib
29  */
30
31 #include "Pressure_Sensor_Driver.h"
32
33 // Defining the States
34 enum {
35     PS_INIT,
36     PS_READING,
37     PS_WAITING
38 }PS_Status;
39
40 static uint32_t PS_pressure_value ;
41
42 STATE_DEFINE(PRESSURE_SENSOR_DRIVER_H_);
43 {
44     // Initialize the pressure sensor
45     // Call the pressure sensor driver functions
46     // State Action
47     PS_Status = PS_INIT;
48
49     // Check event and update state
50     PS_state = STATE(PRESSURE_SENSOR_DRIVER_H_);
51 }
52
53 STATE_DEFINE(PRESSURE_SENSOR_DRIVER_H_);
54 {
55     // State Action
56     PS_Status = PS_READING;
57
58     // Read form pressure sensor
59     PS_pressure_value = GPIO_Get_Pressure_Value();
60
61     // Check event and update state
62     PS_state = STATE(PRESSURE_SENSOR_DRIVER_H_);
63 }
64
65

```



```

42
43 STATE_DEFINE(PRESSURE_SENSOR_DRIVER_H_);
44 {
45     // State Action
46     PS_Status = PS_WAITING;
47
48     // Wait for data
49     GPIO_Delay(1000);
50
51     // Check event and update state
52     PS_state = STATE(PRESSURE_SENSOR_DRIVER_H_);
53 }
54
55 // Set pressure in Main Algorithm
56 uint32_t PS_get_pressure_value()
57 {
58     return PS_pressure_value;
59 }
60
61

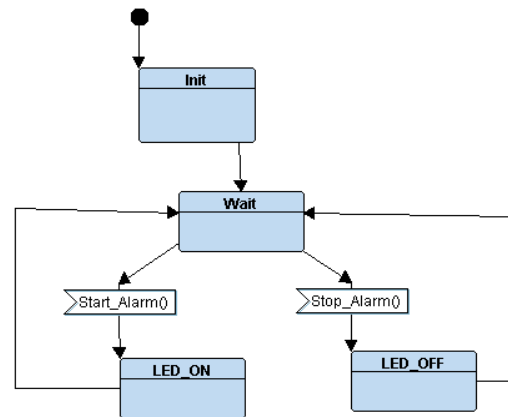
```

## Alarm\_LED\_Driver

```

1  /*
2   * Alarm_LED_Driver.h
3   *
4   * Created on: Nov 26, 2022
5   * Author: Ebram Habib
6   */
7
8  #ifndef ALARM_LED_DRIVER_H_
9  #define ALARM_LED_DRIVER_H_
10
11 #include "State.h"
12
13 //Declare State Functions of Alarm LED
14 STATE_DEFINE(ALARM_LED_INIT);
15 STATE_DEFINE(ALARM_LED_WAITING);
16 STATE_DEFINE(ALARM_LED_ON);
17 STATE_DEFINE(ALARM_LED_OFF);
18
19 //State Pointer to function
20 extern void (*ALARM_LED_state)();
21
22 #endif /* ALARM_LED_DRIVER_H_ */
23

```



```

1  /*
2   * Alarm_LED_Driver.c
3   *
4   * Created on: Nov 26, 2022
5   * Author: Ebram Habib
6   */
7
8  #include "Alarm_LED_Driver.h"
9
10 // Defining the States
11 enum {
12     ALARM_LED_INIT,
13     ALARM_LED_WAITING,
14     ALARM_LED_ON,
15     ALARM_LED_OFF,
16 }ALARM_LED_Status;
17
18 STATE_DEFINE(ALARM_LED_INIT)
19 {
20     // Initialize the alarm LED driver
21     // Call the alarm LED driver functions
22     // State Action
23     ALARM_LED_Status = ALARM_LED_INIT;
24
25     // Check event and update state
26     ALARM_LED_state = STATE(ALARM_LED_WAITING);
27 }
28
29 STATE_DEFINE(ALARM_LED_WAITING)
30 {
31     // State Action
32     ALARM_LED_Status = ALARM_LED_WAITING;
33 }
34

```

```

34 STATE_DEFINE(ALARM_LED_ON)
35 {
36     // State Action
37     ALARM_LED_Status = ALARM_LED_ON;
38
39     // Start Alarm Actuator
40     GPIO_Set_Alarm_LED(TRUE);
41
42     // Check event and update state
43     ALARM_LED_state = STATE(ALARM_LED_WAITING);
44 }
45
46 STATE_DEFINE(ALARM_LED_OFF)
47 {
48     // State Action
49     ALARM_LED_Status = ALARM_LED_OFF;
50
51     // Start Alarm Actuator
52     GPIO_Set_Alarm_LED(FALSE);
53
54     // Check event and update state
55     ALARM_LED_state = STATE(ALARM_LED_WAITING);
56 }
57
58 void ALARM_LED_start_alarm(void)
59 {
60     // Update State
61     ALARM_LED_state = STATE(ALARM_LED_ON);
62 }
63
64 void ALARM_LED_stop_alarm(void)
65 {
66     // Update State
67     ALARM_LED_state = STATE(ALARM_LED_OFF);
68 }
69

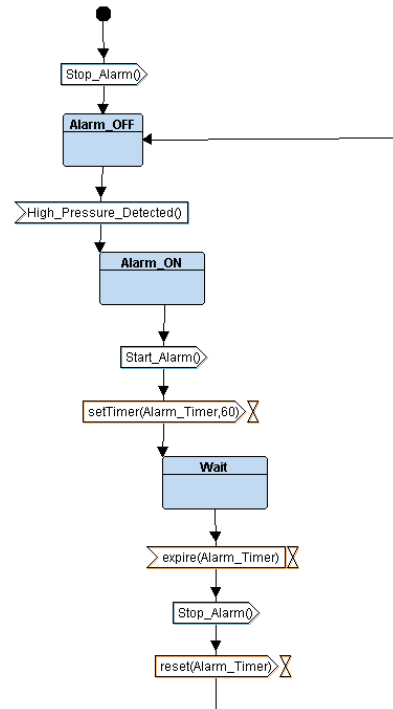
```

# Alarm\_Monitor

```

1  /*
2  * Alarm_Monitor.h
3  *
4  * Created on: Nov 26, 2022
5  * Author: Ebram Habib
6  */
7
8  #ifndef ALARM_MONITOR_H_
9  #define ALARM_MONITOR_H_
10
11 #include "State.h"
12
13 //Declare State Functions of Alarm Monitor
14 STATE_DEFINE(ALARM_MONITOR_ALARM_OFF);
15 STATE_DEFINE(ALARM_MONITOR_ALARM_ON);
16 STATE_DEFINE(ALARM_MONITOR_WAITING);
17
18 //State Pointer to function
19 extern void (*ALARM_MONITOR_state)();
20
21 #endif /* ALARM_MONITOR_H_ */
22

```



```

1  /*
2  * Alarm_Monitor.c
3  *
4  * Created on: Nov 26, 2022
5  * Author: Ebram Habib
6  */
7
8  #include "Alarm_Monitor.h"
9
10 // Defining the states
11 enum {
12     ALARM_MONITOR_ALARM_OFF,
13     ALARM_MONITOR_ALARM_ON,
14     ALARM_MONITOR_WAITING
15 }ALARM_MONITOR_Status;
16
17 STATE_DEFINE(ALARM_MONITOR_ALARM_OFF)
18 {
19     // State Action
20     ALARM_MONITOR_Status = ALARM_MONITOR_ALARM_OFF;
21
22     // Stop alarm LED
23     ALARM_LED_stop_alarm();
24
25     // Check event and update state
26     if(MA_high_pressure_detected() == TRUE)
27     {
28         ALARM_MONITOR_state = STATE(ALARM_MONITOR_ALARM_ON);
29     }
30 }

```

```

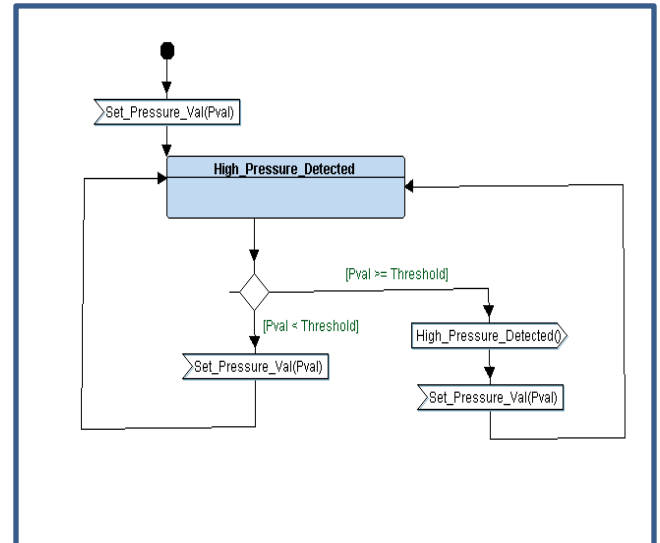
31
32 STATE_DEFINE(ALARM_MONITOR_ALARM_ON)
33 {
34     // State Action
35     ALARM_MONITOR_Status = ALARM_MONITOR_ALARM_ON;
36
37     // Start alarm LED
38     ALARM_LED_start_alarm();
39
40     // Check event and update state
41     ALARM_MONITOR_state = STATE(ALARM_MONITOR_WAITING);
42 }
43
44 STATE_DEFINE(ALARM_MONITOR_WAITING)
45 {
46     // State Action
47     ALARM_MONITOR_Status = ALARM_MONITOR_WAITING;
48
49     GPIO_Delay(1000);
50
51     // Check event and update state
52     ALARM_MONITOR_state = STATE(ALARM_MONITOR_ALARM_OFF);
53 }
54

```

## Main\_Algorithm

```
1  /*
2  * Main_Algorithm.h
3  *
4  * Created on: Nov 26, 2022
5  * Author: Ebram Habib
6  */
7
8  #ifndef MAIN_ALGORITHM_H_
9  #define MAIN_ALGORITHM_H_
10
11 #include "State.h"
12
13 //Declare State Functions of Main Algorithm
14 STATE_DEFINE(MA_HIGH_PRESSURE);
15
16 //State Pointer to function
17 extern void (*MA_state)();
18
19 #endif /* MAIN_ALGORITHM_H_ */
20
```

```
1  /*
2  * Main_Algorithm.c
3  *
4  * Created on: Nov 26, 2022
5  * Author: Ebram Habib
6  */
7
8
9  #include "Main_Algorithm.h"
10
11 // Define the states
12 enum {
13     MA_HIGH_PRESSURE
14 }MA_Status;
15
16 static uint32_t MA_pressure_value;
17 static uint32_t MA_pressure_threshold = 20;
18
19 STATE_DEFINE(MA_HIGH_PRESSURE)
20 {
21     // State Action
22     MA_Status = MA_HIGH_PRESSURE;
23
24     // Read pressure value from pressure sensor
25     MA_pressure_value = PS_get_pressure_value();
26
27     // Check event and update state
28     MA_state = STATE(MA_HIGH_PRESSURE);
29 }
30
31 // Main Program =====> Alarm Monitor
32 uint32_t MA_high_pressure_detected(void)
33 {
34     return (MA_pressure_value >= MA_pressure_threshold);
35 }
36
```



## SW analysis:

### 1- Map file

```
1
2 Allocating common symbols
3 Common symbol size file
4
5 MA_Status 0x1 Main_Algorithm.o
6 ALARM_LED_Status 0x1 Alarm_LED_Driver.o
7 ALARM_MONITOR_Status 0x1 Alarm_Monitor.o
8 PS_Status 0x1 Pressure_Sensor_Driver.o
9
10
11 Memory Configuration
12
13 Name Origin Length Attributes
14 flash 0x08000000 0x20000000 xrw
15 sram 0x20000000 0x20000000 xrw
16 *default* 0x08000000 0xffffffff
17
18 Linker script and memory map
19
20
21 .text 0x08000000 0x3cc
22 *(.vectors*)
23 .vectors 0x08000000 0x1c Startup.o
24 0x08000000 vectors
25
26 *(.text*)
27 .text 0x0800001c 0xc4 Alarm_LED_Driver.o
28 0x0800001c ST_ALARM_LED_INIT
29 0x08000040 ST_ALARM_LED_WAITING
30 0x08000058 ST_ALARM_LED_ON
31 0x08000080 ST_ALARM_LED_OFF
32 0x080000a8 ALARM_LED_start_alarm
33 0x080000c4 ALARM_LED_stop_alarm
34
35 .text 0x080000e0 0x7c Alarm_Monitor.o
36 0x080000e0 ST_ALARM_MONITOR_ALARM_OFF
37 0x08000110 ST_ALARM_MONITOR_ALARM_ON
38 0x08000134 ST_ALARM_MONITOR_WAITING
39
40 .text 0x0800015c 0xc4 GPIO_Driver.o
41 0x0800015c GPIO_Delay
42 0x0800017c GPIO_Get_Pressure_Value
43 0x08000194 GPIO_Set_Alarm_LED
44 0x080001d0 GPIO_Init
45 0x08000220 0x34 main.o
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81 .data 0x20000000 0x14 load address 0x080003cc
82 0x20000000 _S_DATA = .
83
84 *(.data*)
85 .data 0x20000000 0x0 Alarm_LED_Driver.o
86 .data 0x20000000 0x0 Alarm_Monitor.o
87 .data 0x20000000 0x0 GPIO_Driver.o
88 .data 0x20000000 0x10 main.o
89 0x20000000 PS_state
90 0x20000004 ALARM_LED_state
91 0x20000008 ALARM_MONITOR_state
92 0x2000000c MA_state
93
94 .data 0x20000010 0x4 Main_Algorithm.o
95 .data 0x20000014 0x0 Pressure_Sensor_Driver.o
96 .data 0x20000014 0x0 Startup.o
97
98 . = ALIGN (0x4)
99 _E_DATA = .
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

## 2- Symbols table

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



ET-UNIT-TECHNOLOGY-RESEARCH-PROGRAM-2 / 5/ DOWNLOADS/ EMBEDDED HERE WE GO AGAIN/ KEROLOS

```
$ arm-none-eabi-objdump.exe -h First_Term_First_Project_High_Pressure_System.elf
First_Term_First_Project_High_Pressure_System.elf:      file format elf32-littlearm

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

#### 4- Entry Point Address

```

Entry point address: 0x8000000
Start of program headers: 52 (bytes into file)
Start of section headers: 141672 (bytes into file)
Flags: 0x5000200, Version5 EABI, soft-float ABI
Size of this header: 52 (bytes)
Size of program headers: 32 (bytes)
Number of program headers: 2
Size of section headers: 40 (bytes)
Number of section headers: 16
Section header string table index: 15

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

```

# Proteus Simulation

Proteus 8 Professional - Schematic Capture

View Tool Design Graph Debug Library Template System Help

Base Design

HW\_project\_KIT\_FIRST\_TERM\_project1

Pressure Sensor

Bit 0

Bit 7

ALARM

CM3 Source Code - U1

GPIO\_Driver.c

```

800015C {
800015D   for(; ncount != 0; ncount--);
800015E }
800015F
8000160 uint32_t GPIO_Get_Pressure_Value(void)
8000161 {
8000162   return (GPIOA_IDR & 0xFF);
8000163 }
8000164
8000165 void GPIO_Set_Alarm_LED(uint32_t i)
8000166 {
8000167   if (i == 1)
8000168   {
8000169     SET_BIT(GPIOA_ODR, 13);
8000170   }
8000171   else if (i == 0)
8000172   {
8000173     CLR_BIT(GPIOA_ODR, 13);
8000174   }
8000175 }
8000176
8000177 void GPIO_Init (void)
8000178 {
8000179   SET_BIT(APB2ENR, 2);
8000180 }
8000181
8000182 GPIOA_CRL &= 0xFFFFFFF;
8000183 GPIOA_CRL |= 0x00000000;
8000184 GPIOA_CRH &= 0xFFFFFFF;
8000185 GPIOA_CRH |= 0x22222222;
8000186 }
    
```

CM3 Variables - U1

Name	Address	Value
ALARM_MONITOR_Status	20001010	ALARM_MONITOR_WA...
MA_Status	2000101E	MA_HIGH_PRESSURE (0)
MA_pressure_value	20000014	27
MA_pressure_threshold	20000010	20
PS_Status	2000101F	PS_READING (1)
PS_pressure_value	20000018	27
STACK_TOP	2000103C	16777730
Vectors	08000000	dword[7]
ALARM_LED_Status	2000103C	ALARM_LED_ON (2)
ncount	BP+12 = ...	501

Activate Windows  
Go to Settings to activate Windows.

12:01 AM  
11/27/2022

Proteus 8 Professional - Schematic Capture

View Tool Design Graph Debug Library Template System Help

Base Design

HW\_project\_KIT\_FIRST\_TERM\_project1

Pressure Sensor

Bit 0

Bit 7

ALARM

CM3 Source Code - U1

GPIO\_Driver.c

```

/*
 * GPIO_Driver.c
 * Created on: Nov 26, 2022
 * Author: Ebrahim Habib
 */
#include "GPIO_Driver.h"
#include "util.h"

void GPIO_Delay(uint32_t ncount)
{
800015C {
800015D   for(; ncount != 0; ncount--);
800015E }
800015F
8000160 uint32_t GPIO_Get_Pressure_Value(void)
8000161 {
8000162   return (GPIOA_IDR & 0xFF);
8000163 }
8000164
8000165 void GPIO_Set_Alarm_LED(uint32_t i)
8000166 {
8000167   if (i == 1)
8000168   {
8000169     SET_BIT(GPIOA_ODR, 13);
8000170   }
8000171   else if (i == 0)
8000172   {
8000173     CLR_BIT(GPIOA_ODR, 13);
8000174   }
8000175 }
8000176
8000177 void GPIO_Init (void)
8000178 {
8000179   SET_BIT(APB2ENR, 2);
8000180 }
8000181
8000182 GPIOA_CRL &= 0xFFFFFFF;
8000183 GPIOA_CRL |= 0x00000000;
8000184 GPIOA_CRH &= 0xFFFFFFF;
8000185 GPIOA_CRH |= 0x22222222;
8000186 }
    
```

CM3 Variables - U1

Name	Address	Value
ALARM_MONITOR_Status	20001010	ALARM_MONITOR_AL...
MA_Status	2000101E	MA_HIGH_PRESSURE (0)
MA_pressure_value	20000014	27
MA_pressure_threshold	20000010	20
PS_Status	2000101F	PS_WAITING (2)
PS_pressure_value	20000018	27
STACK_TOP	2000103C	33554691
Vectors	08000000	dword[7]
ALARM_LED_Status	2000103C	ALARM_LED_OFF (3)
ncount	BP+12 = ...	481

Activate Windows  
Go to Settings to activate Windows.

11:54 PM  
11/26/2022