PRESSURE CONTROLLER

Mastering Embedded System Online Diploma

www.learn-in-depth.com

First Term (Final Project 1)

Eng. Reham Nady Abd Elmotaal

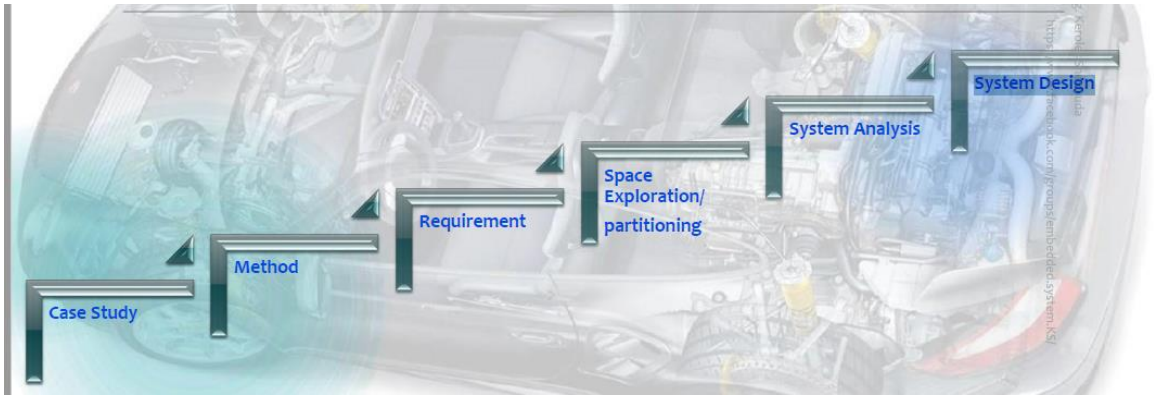
My Profile:
https://www.learn-in-depth.com/online-diploma/reham.nady68@gmail.com

1-Introduction:

Write a bare metal software to control pressure in cabin which informs the crew of a cabin when the pressure exceeds specific value in the cabin with an alarm connected to GPIO port A in Stm32f103CX micro-controller chip. I will use UML to simplify the process of software design. I will build everything from scratch including startup, linker script and source codes, and compile them using arm cross tool chain.

2-System Architecting/Design Sequence:

That contains from 6 steps (case study- Method- Requirement- Space Exploration/partitioning- System Analysis- System Design).



1.Case Study: (a Pressure Controlling System)

❖ Specifications:

- A pressure controller informs the crew of a cabin with an alarm when the pressure exceeds 20 bars in the cabin.
- The Alarm Duration =60s.
- keeps track of the measured values.

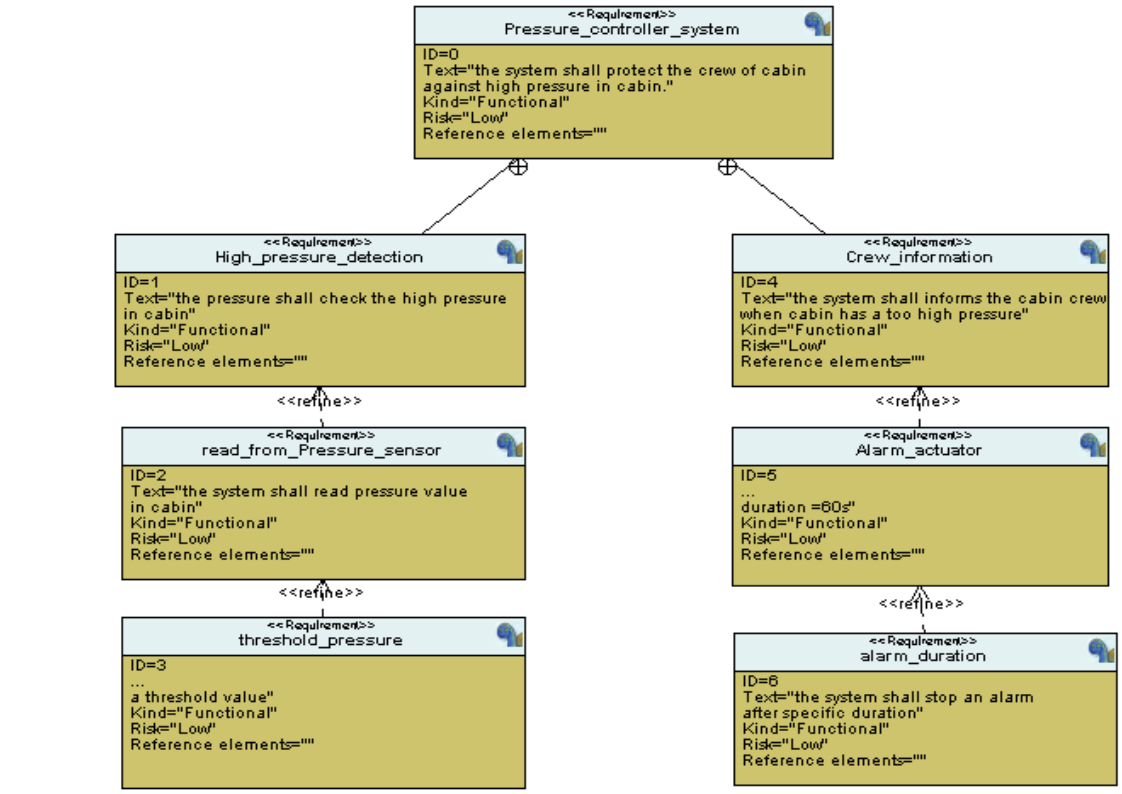
❖ Assumptions:

- 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.

2-Method:

Using V-Model SDLC.

3- System Requirements:



4- System Analysis:

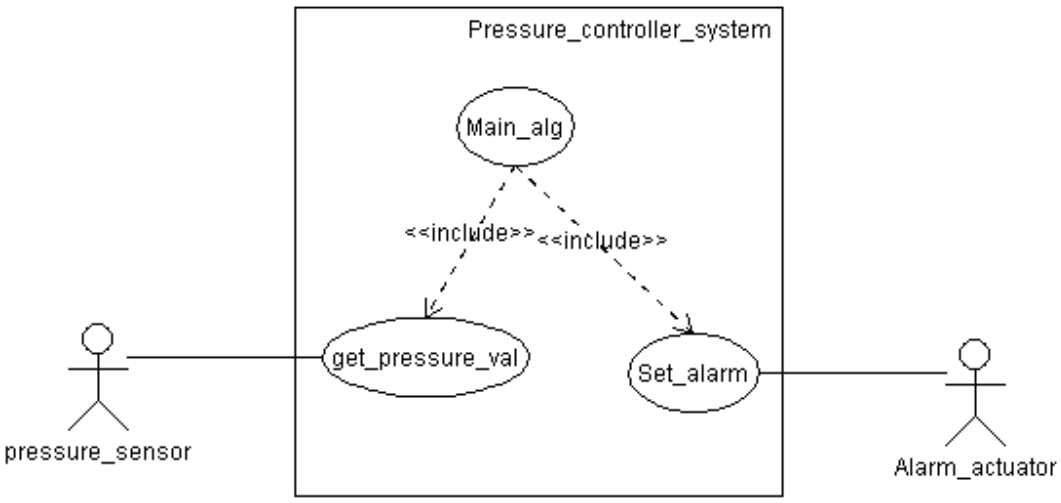
Analysis Method:

1.System boundary and main functions →Use Case Diagram

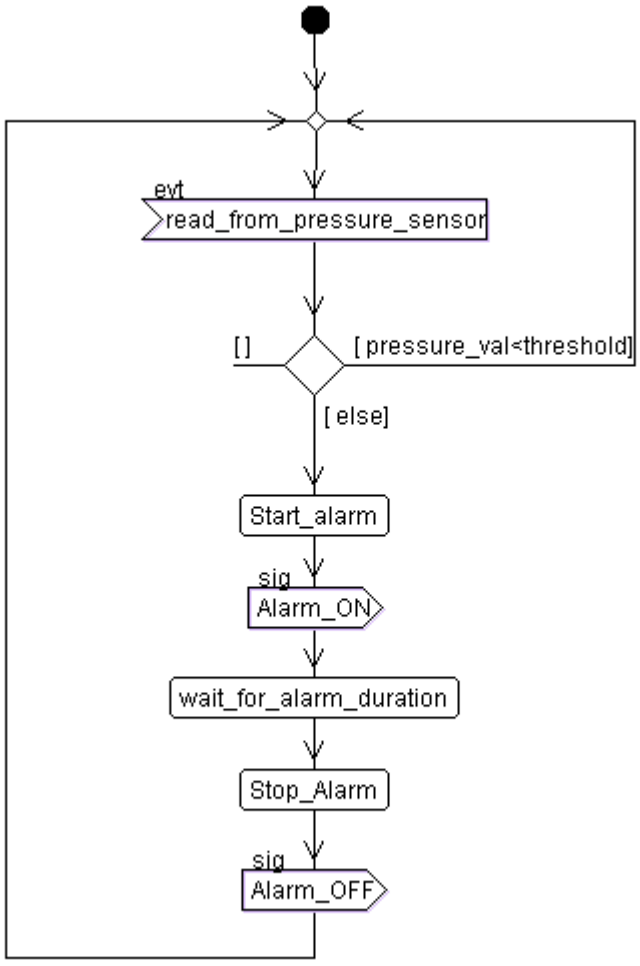
2.Relations between main functions →Activity Diagram

3.Communications between main system entities and actors →Sequence Diagram.

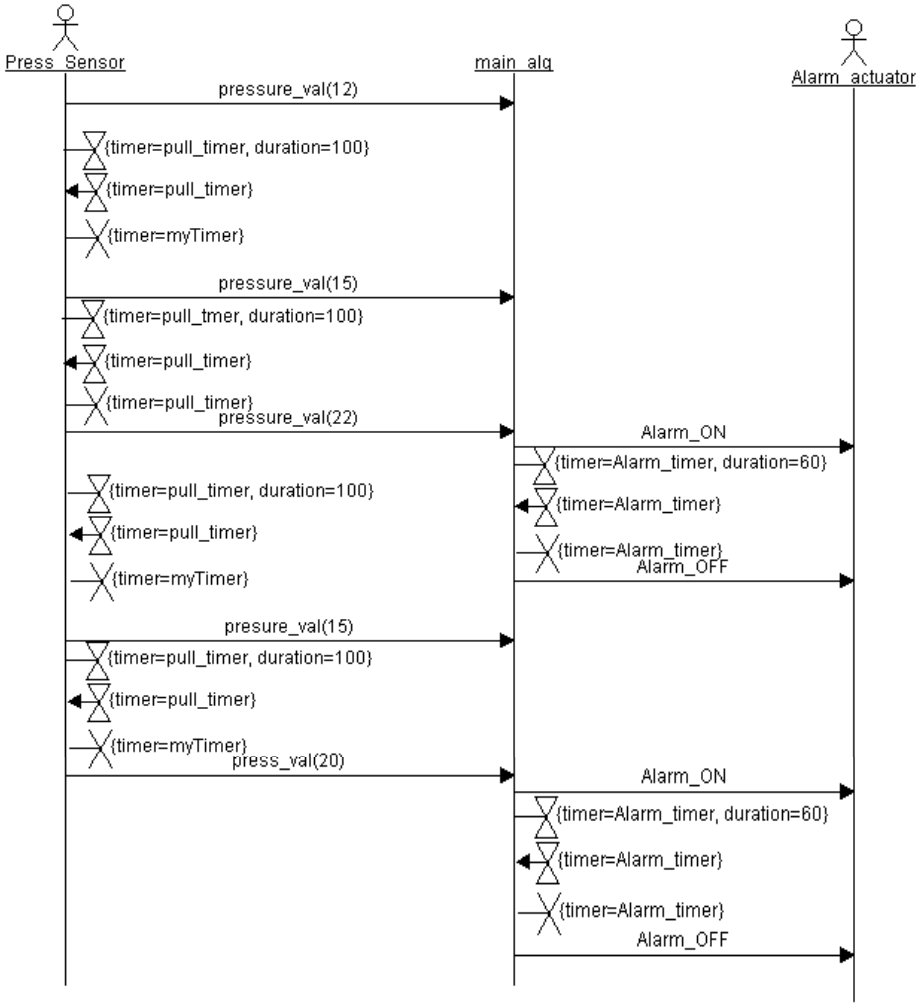
❖ Use Case Diagram:



❖ Activity Diagram:



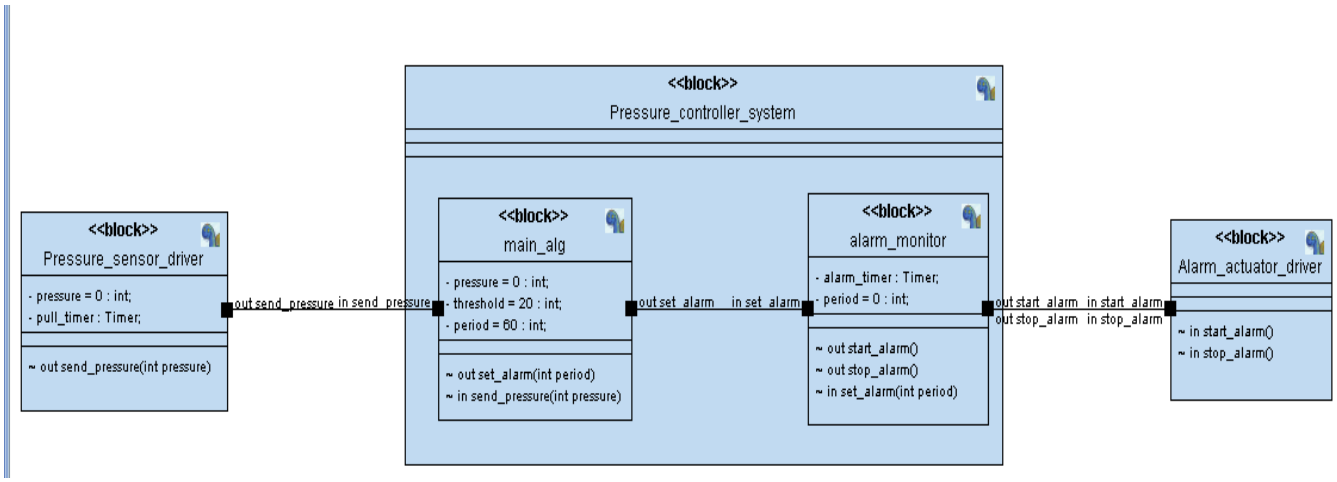
❖ Sequence Diagram:



5- System Design:

❖ System architecture:

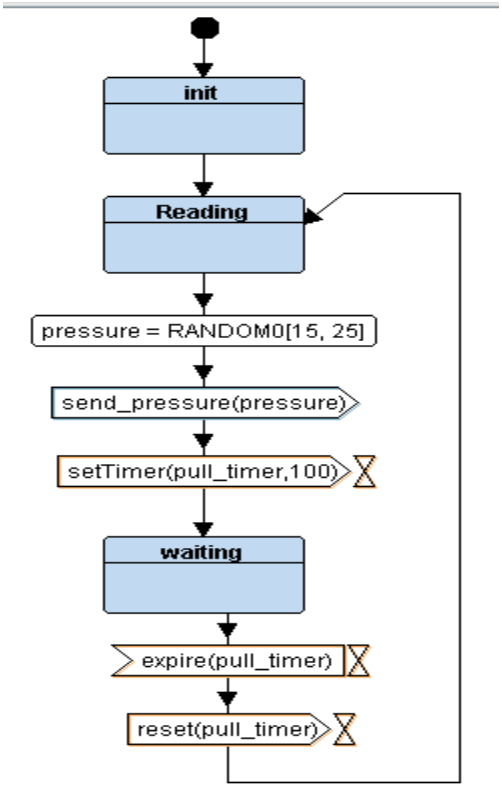
Block Definition Diagram and Internal Block Diagram.



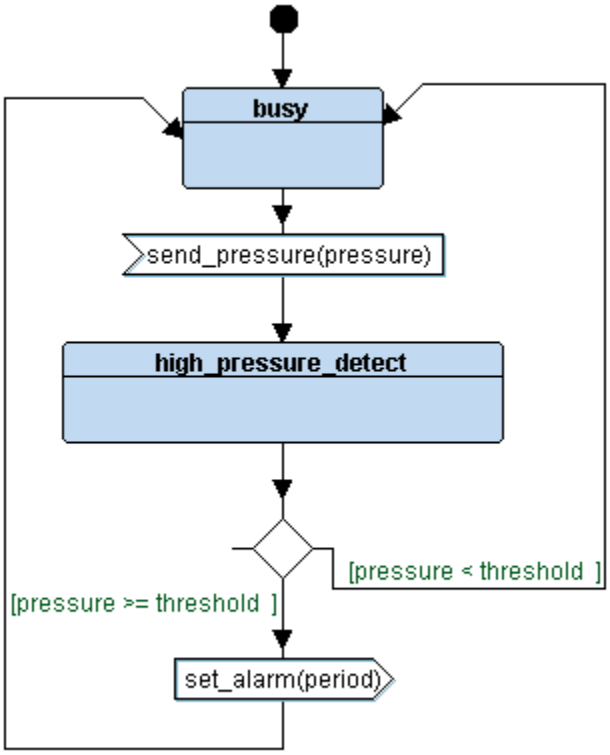
❖ Behavior of the system:

State Machine Diagram.

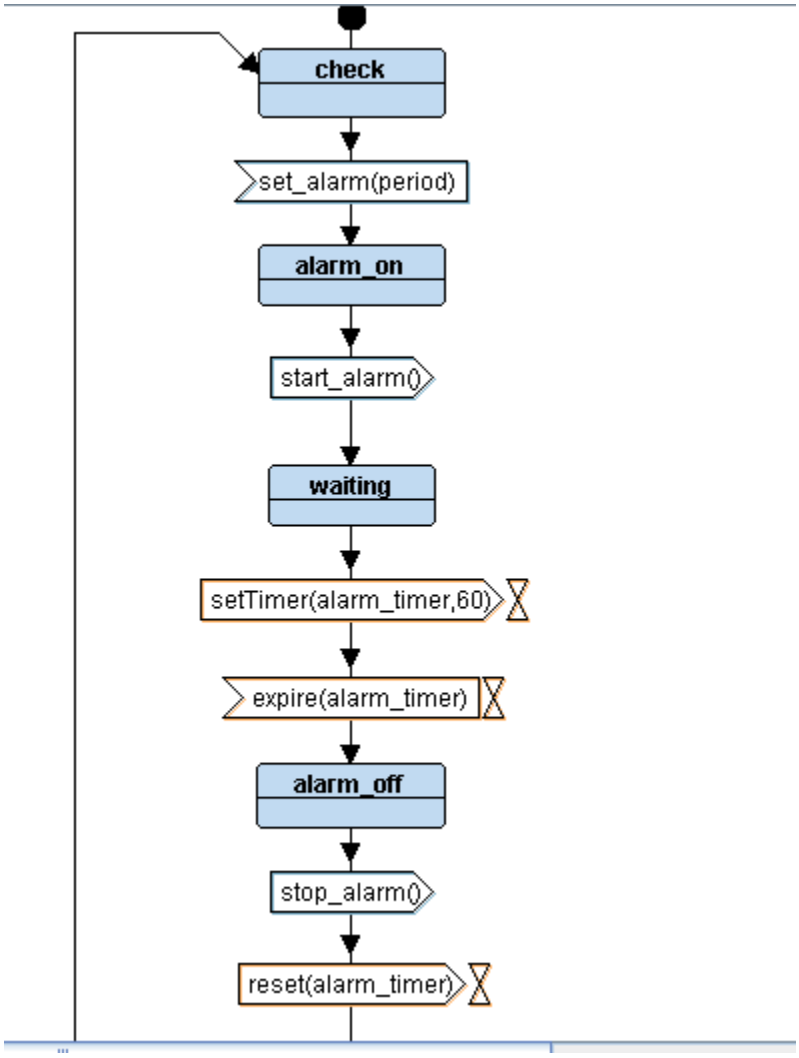
➢ Pressure-Sensor-driver:



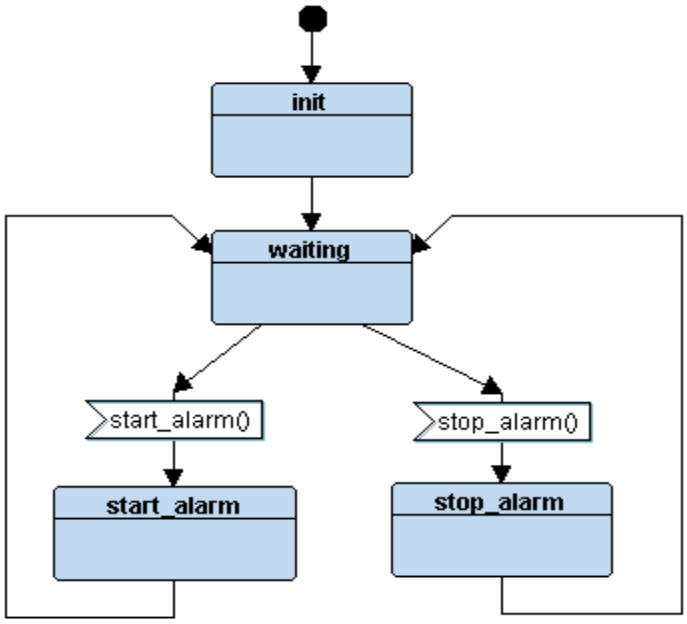
- **High-pressure-detection:**



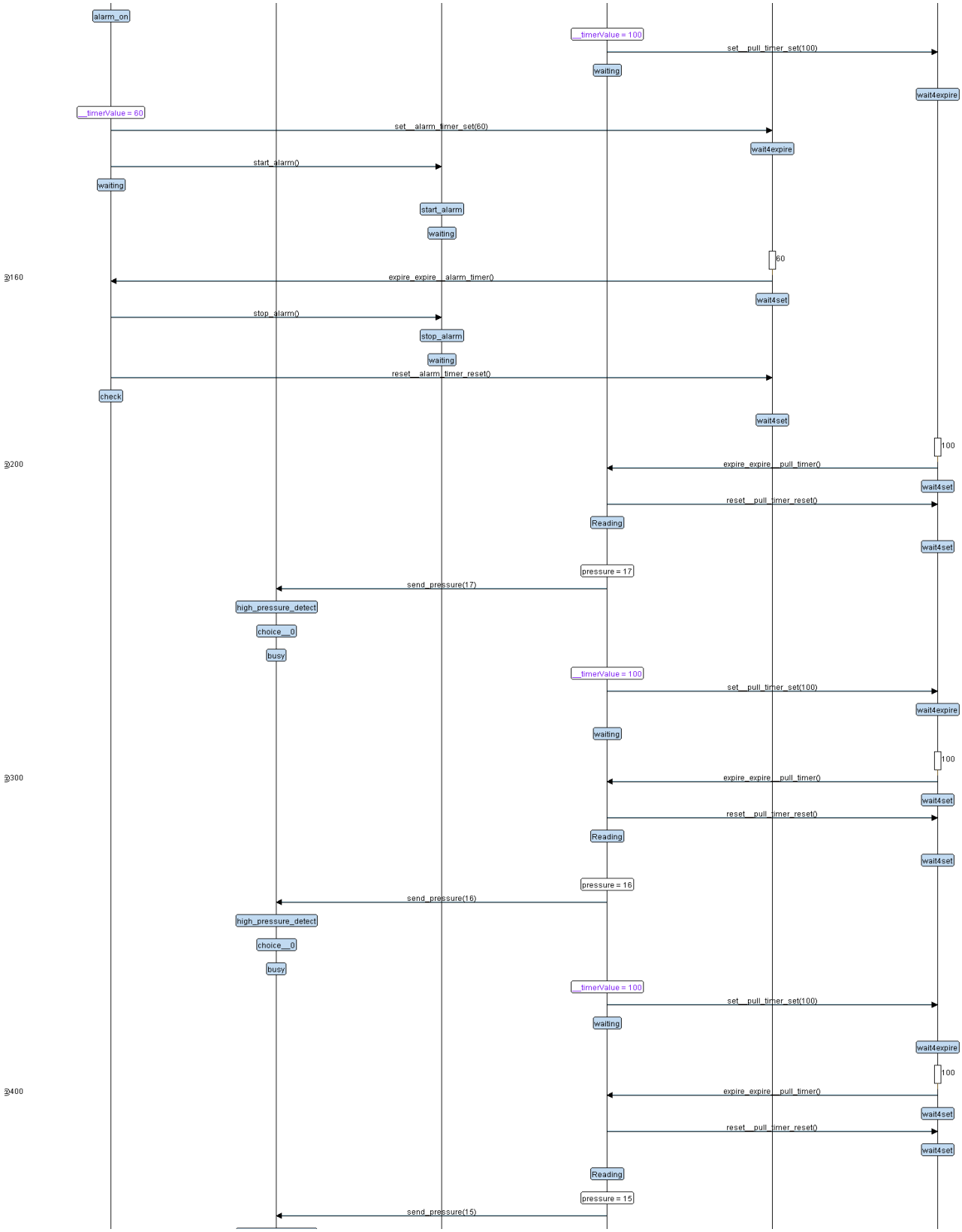
- **Alarm-monitor:**

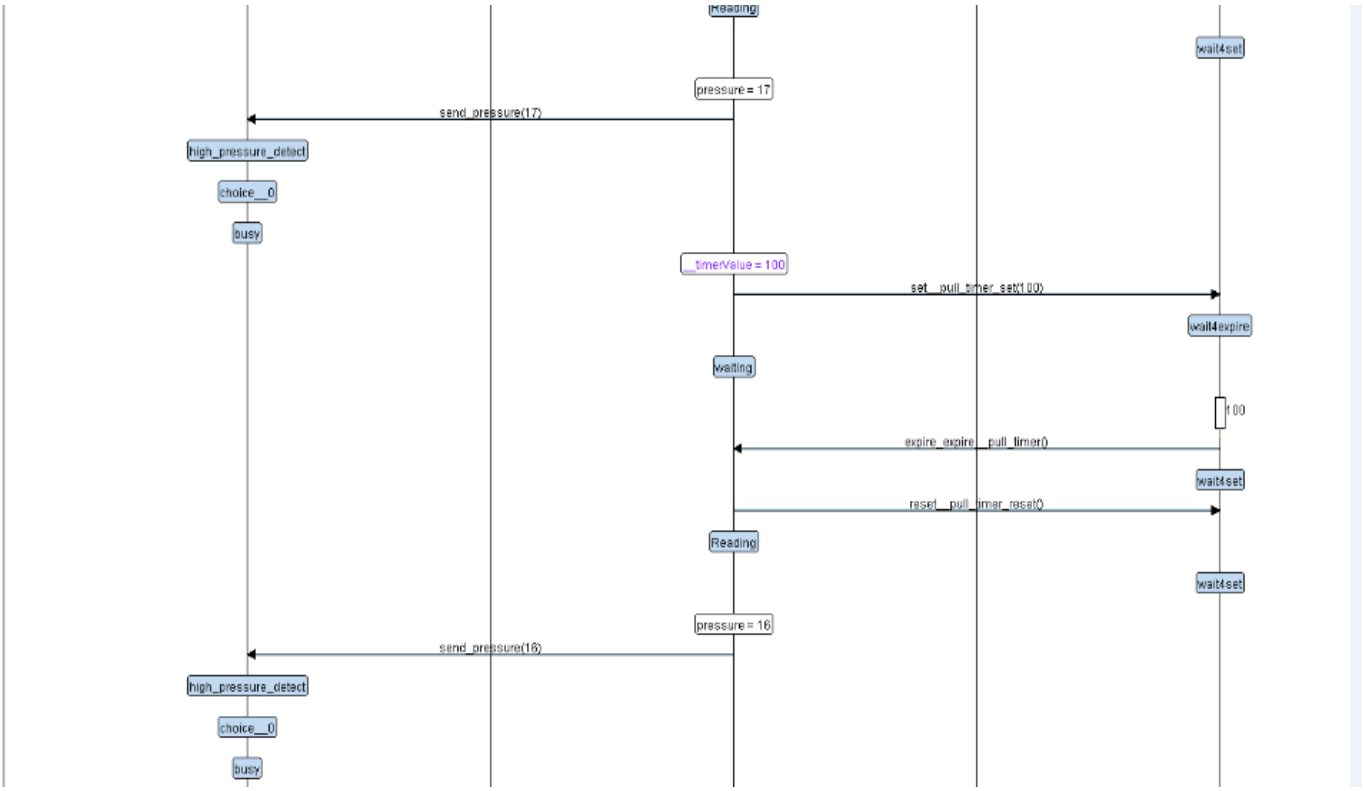


- **Alarm-actuator-driver:**



Different scenario:




Reading
wait4set
pressure = 17
send_pressure(17)
high_pressure_detect
choice__0
busy
_timerValue = 100
set__pull_timer_set(100)
wait4expire
waiting
100
expire_expire__pull_timer()
wait4set
reset__pull_timer_reset()
Reading
wait4set
pressure = 16
send_pressure(16)
high_pressure_detect
choice__0
busy

3- Source codes:

We have 4 modules, so we have 4 source codes &header file.

•Pressure-Sensor-driver:

To initialize pressure sensor and read pressure value.

```c
#include"P_sensor.h"

int pressure;
void(*PSensor_state)();
void PSensor_init(){

}
State_define(PSensor_Waiting){
  PSensor_id=PSensor_Waiting;
  Delay(100);
  PSensor_state=State(Reading);
}
State_define(Reading){
  PSensor_id=Reading;
  pressure=getPressureVal();
  send_pressure(pressure);
  PSensor_state=State(PSensor_Waiting);
}
```

p-sensor .c

```c
#ifndef P_SENSOR_H_
#define P_SENSOR_H_
#include"state.h"
extern void(*PSensor_state)();
enum {
  Reading,
  PSensor_Waiting
}PSensor_id;

void PSensor_init();
State_define(PSensor_Waiting);
State_define(Reading);


#endif

```

p-sensor .h

•High-Pressure-detect:

Get pressure value from pressure sensor and detect pressure value .if pressure value exceeds threshold value will send signal to alarm monitor module.

```c
#include"Pressure_detect.h"

void(*Pdetect_state)();
int pressure_val;
int threshold = 20;
int alarm_period=60;
void send_pressure(int pressure){
  pressure_val=pressure;
  Pdetect_state=State(High_P_detect);
}
State_define(Busy){
  Pressure_detect_id=Busy;
}
State_define(High_P_detect){
  Pressure_detect_id=High_P_detect;
  if(pressure_val>=threshold){
    set_alarm(alarm_period);
  }
  Pdetect_state=State(Busy);


}
```

Pressure-detect .c

```c
#ifndef PRESSURE_DETECT_H_
#define PRESSURE_DETECT_H_
#include"state.h"
extern void(*Pdetect_state)();
enum {
  Busy,
  High_P_detect
}Pressure_detect_id;

State_define(Busy);
State_define(High_P_detect);


#endif
```

Pressure-detect .h

•Alarm-Monitor:

To monitor signal from high pressure-detect and control alarm actuator.

```c
#include"Alarm_monitor.h"
void(*MONITOR_alarm_state)();
int period_alarm=0;

void set_alarm(int period){
  period_alarm=period;
  MONITOR_alarm_state=State(Alarm_ON);
}

State_define(Check){
  MONITOR_alarm_id=Check;
}

State_define(Alarm_Monitor_Waiting){
  MONITOR_alarm_id=Alarm_Monitor_Waiting;
  Delay(period_alarm);
  MONITOR_alarm_state=State(Alarm_OFF);
}
State_define(Alarm_ON){
  MONITOR_alarm_id=Alarm_ON;
  start_alarm();
  MONITOR_alarm_state=State(Alarm_Monitor_Waiting);
}
State_define(Alarm_OFF){
  MONITOR_alarm_id=Alarm_OFF;
  stop_alarm();
  MONITOR_alarm_state=State(Check);
}
```

Alarm-Monitor .c

```c
#ifndef MONITOR_alarm_MONITOR_H_
#define MONITOR_alarm_MONITOR_H_
#include"state.h"
extern void(*MONITOR_alarm_state)();
enum {
  Check,
  Alarm_ON,
  Alarm_Monitor_Waiting,
  Alarm_OFF
}MONITOR_alarm_id;

State_define(Check);
State_define(Alarm_Monitor_Waiting);
State_define(Alarm_ON);
State_define(Alarm_OFF);

#endif
```

Alarm-Monitor .h

•Alarm-Actuator:

To initialize alarm actuator and control it.

```c
#include"Alarm_actuator.h"
void(*Alarm_state)() ;
void stop_alarm(){
  Alarm_state=State(Alarm_Stop);
}
void start_alarm(){
  Alarm_state=State(Alarm_Start);
}
void Alarm_init(){
  Set_Alarm_actuator(1);
}
State_define(Alarm_Waiting){
  Alarm_id = Alarm_Waiting;
}
State_define(Alarm_Start){
  Alarm_id = Alarm_Start;
  Set_Alarm_actuator(0);
  Alarm_state=State(Alarm_Waiting);
}
State_define(Alarm_Stop){
  Alarm_id = Alarm_Stop;
  Set_Alarm_actuator(1);
  Alarm_state=State(Alarm_Waiting);
}
```

Alarm-actuator .c

```c
#ifndef ALARM_ACTUATOR_H_
#define ALARM_ACTUATOR_H_

#include"state.h"

extern void(*Alarm_state)();
enum {
  Alarm_Waiting,
  Alarm_Start,
  Alarm_Stop
}Alarm_id;

void Alarm_init();
State_define(Alarm_Waiting);
State_define(Alarm_Start);
State_define(Alarm_Stop);

#endif
```

Alarm-actuator .h

4- Startup:

as (CortexM3) can initialize the SP with the first 4 bytes, so we can write startup by C code.

the startup code will include a few instructions to:

- ❖ Write vector handler functions.
- ❖ Copy data section from FLASH to SRAM
- ❖ Initialize. bss section by zero in SRAM
- ❖ Call main function.

```
uint32_t vectors[] __attribute__((section(".vectors"))) ={
      (uint32_t) &_stack_top,
      (uint32_t) &Reset_Handler,
      (uint32_t) &NMI_Handler,
      (uint32_t) &H_fault_Handler,
      (uint32_t) &MM_fault_Handler,
      (uint32_t) &BUS_Fault,
      (uint32_t) &Usage_Fault_Handler,
};

void Reset_Handler(){
  int i=0;
  // copy . data section from flash to sram
  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;
  for(i=0;i<Data_size;i++){*p_dst++ = *p_src++;}
  // initilaize . bss section
  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++ = 0;}
  // branch to main
  main();
}
void Default_Handler(){main();}
```

5- Linker script:

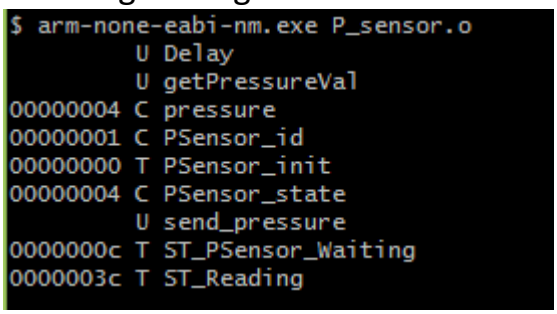
In this linker script, we define memory boundaries, in this app we have two memories. the last section in linker used to divide my code in all files and organize it to burn it on the micro controller.

```
MEMORY
{
  FLASH (rx) : ORIGIN = 0x08000000, LENGTH = 128k
  SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 20k
}

SECTIONS
{
  .text : {
    *(.vectors*)
    *(.text*)
    *(.rodata*)
    _E_text = . ;
  } > FLASH

  .data : {
    _S_data = . ;
    *(.data*)
    _E_data = . ;
  } > SRAM AT> FLASH

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

  . = . + 0x1000 ;
  _stack_top = . ;

}
```

6- Symbols:

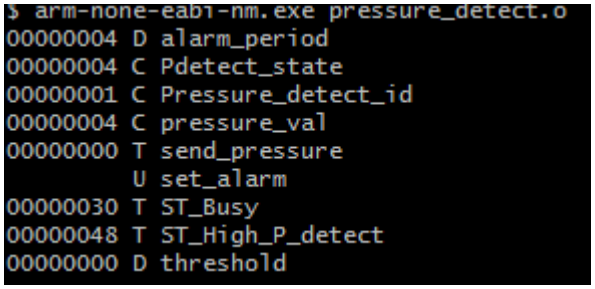
• Pressure-sensor. o:

1- Delay: unresolved symbol and will be resolved during Linking process.
2- Getpressureval: unresolved symbol and will be resolved during Linking process.
3- Pressure: which in code section.
4- Psensor-id: which in code section.
5- Psensor-init: which in text section.
6- Psensor-state: which in code section.
7- Send-pressure: unresolved symbol and will be resolved during Linking process.
8- St-psensor-waiting: which in text section.
9- St-reading: which in text section.

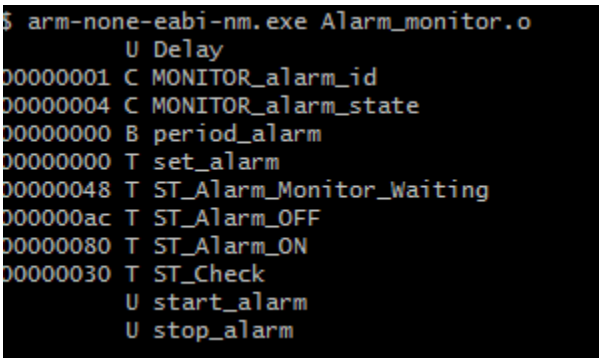


• High-Pressure-detect. o:

1- alarm-period: which in data section.
2- Pdetect-state: which in code section.
3- Pressure-detect-id: which in code sec.
4- Pressure-val: which in code section.
5- Send-pressure: which in text section.
6- Set-alarm: unresolved symbol and will be resolved during Linking process.
7- St-busy: which in text section.
8- ST-high-p-detect: which in text section.
9- Threshold: which in data section.

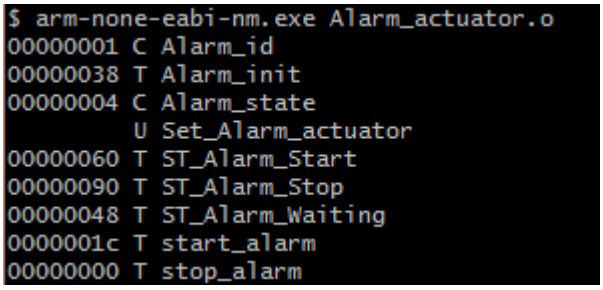


•Alarm-Monitor. o:



1- Delay: unresolved symbol and will be resolved during Linking process.
2- Monitor-alarm-id: which in code section.
3- Monitor-alarm-state: which in code section.
4- Period-alarm: which in bss section.
5- Set-alarm: which in text section.
6- St-alarm-monitor-waiting: which in text section.
7- St-alarm-off: which in text section.
8- St-alarm-on: which in text section.
9- St-check: which in text section.
10- Start-alarm: unresolved symbol and will be resolved during Linking process.
11- Stop-alarm: unresolved symbol and will be resolved during Linking process.

•Alarm-Actuator. o:



1- alarm-id: which in code section.
2- Alarm-init: which which in text section.
3- Alarm-state:which in code section.
4- Set-alarm-actuator: unresolved symbol and will be resolved during Linking process.
5- St-alarm-start:which in text section.
6- St-alarm-stop:which in text section.
7- St-alarm-waiting :which in text section.
8- Start-alarm:which in text section.
9- Stop-alarm:which in text section.

•driver. o:

1- Delay: which in text section.
2- Getpressureval: which in text section.
3- Gpio-initialization: which in text section.
4- Set-alarm-actuator: which in text section.

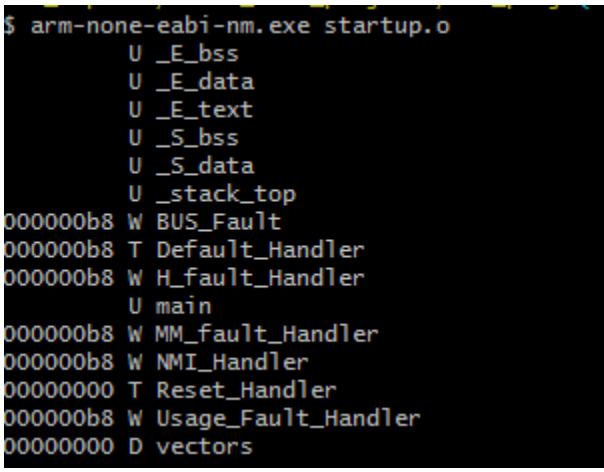
```
$ arm-none-eabi-nm.exe driver.o
00000000 T Delay
00000024 T getPressureVal
0000008c T GPIO_INITIALIZATION
0000003c T Set_Alarm_actuator
```

•main. o:

1- Alarm-id:which in code section.
2- Alarm-init& Alarm-state& Delay& Gpio-initialization: unresolved symbol and will be resolved during Linking process.
3- Main: which in text section.
4- Monitor-alarmstate: unresolved symbol and will be resolved during Linking process.
5- Pressure-detect-id:which in code section.
6- Psensor-id:which in code sedction.
7- Psensor-init & psenssor-state: unresolved symbol and will be resolved during Linking process.
8- Setup:which in text section.
9- St-alarm-waiting& St-busy& st-check&st-reading: unresolved symbol and will be resolved during Linking process.

```
$ arm-none-eabi-nm.exe main.o
00000001 C Alarm_id
         U Alarm_init
         U Alarm_state
         U Delay
         U GPIO_INITIALIZATION
0000005c T main
00000001 C MONITOR_alarm_id
         U MONITOR_alarm_state
         U Pdetect_state
00000001 C Pressure_detect_id
00000001 C PSensor_id
         U PSensor_init
         U PSensor_state
00000000 T setup
         U ST_Alarm_Waiting
         U ST_Busy
         U ST_Check
         U ST_Reading
```

•Startup. o:



1- -E-bss&-E-data&-E-text&-S-bss&-S-data&-stack-top: unresolved symbol and will be resolved during Linking process.
2- Bus-fault:which weak symbol.
3- Default-handler:which in text section.
4- H-fault-handler:weak sympol.
5- Main: unresolved symbol and will be resolved during Linking process.
6- MM-fault-handler &NMI-handler&usage-fault-handler: weak sympol.
7- Reset-handler: which in text section.
8- Vectors:which in data section.

- **Elf image symbols:**

All symbols are resolved.

```
2000000c B _E_bss
20000008 D _E_data
08000540 T _E_text
20000008 B _S_bss
20000000 D _S_data
20001030 B _stack_top
2000000c B Alarm_id
08000054 T Alarm_init
20000004 D alarm_period
20000010 B Alarm_state
08000534 W BUS_Fault
08000534 T Default_Handler
080001b4 T Delay
080001d8 T getPressureVal
08000240 T GPIO_INITIALIZATION
08000534 W H_fault_Handler
0800031c T main
08000534 W MM_fault_Handler
20000014 B MONITOR_alarm_id
20000018 B MONITOR_alarm_state
08000534 W NMI_Handler
2000002c B Pdetect_state
20000008 B period_alarm
20000024 B pressure
2000001c B Pressure_detect_id
20000028 B pressure_val
2000001d B PSensor_id
08000360 T PSensor_init
20000020 B PSensor_state
0800047c T Reset_Handler
080003e4 T send_pressure
080000dc T set_alarm
080001f0 T Set_Alarm_actuator
080002c0 T setup
08000124 T ST_Alarm_Monitor_Waiting
08000188 T ST_Alarm_OFF
0800015c T ST_Alarm_ON
0800007c T ST_Alarm_Start
080000ac T ST_Alarm_Stop
08000064 T ST_Alarm_Waiting
08000414 T ST_Busy
0800010c T ST_Check
0800042c T ST_High_P_detect
0800036c T ST_PSensor_Waiting
0800039c T ST_Reading
08000038 T start_alarm
0800001c T stop_alarm
20000000 D threshold
08000534 W Usage_Fault_Handler
08000000 T vectors
```

7- Sections Headers:

•Pressure-sensor. o:

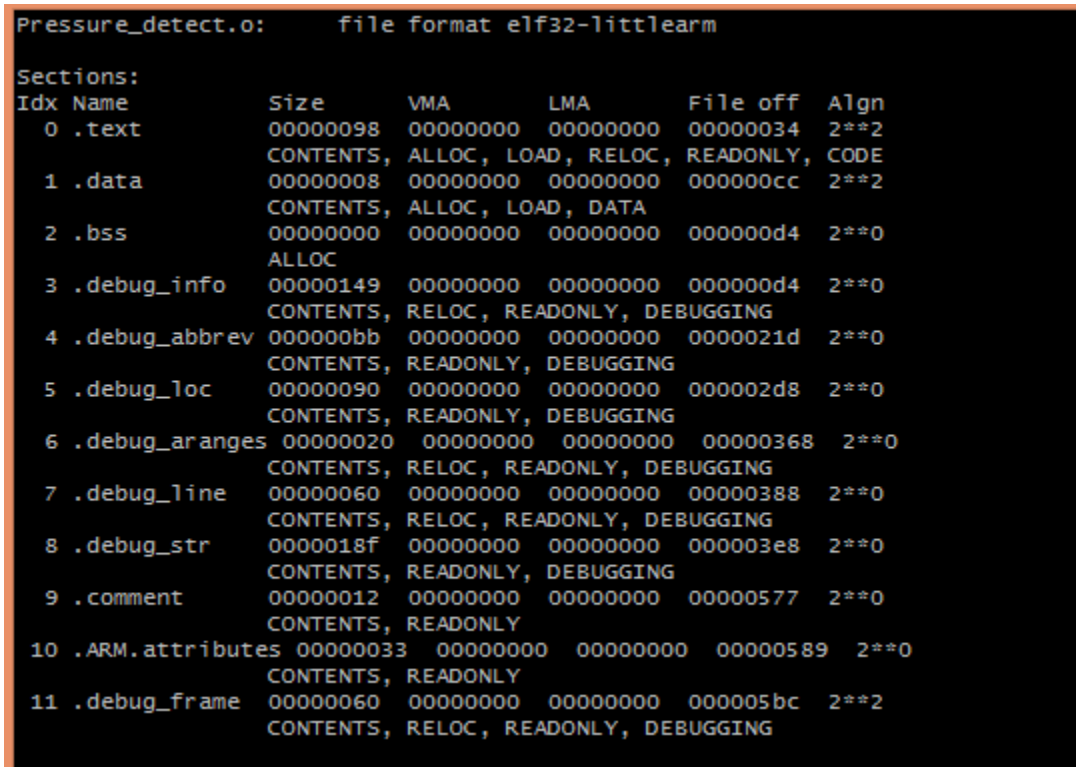
```
P_sensor.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000084  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  000000b8  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000b8  2**0
                  ALLOC
  3 .debug_info   00000111  00000000  00000000  000000b8  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 00000092  00000000  00000000  000001c9  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    00000084  00000000  00000000  0000025b  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  000002df  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   00000052  00000000  00000000  000002ff  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str    00000165  00000000  00000000  00000351  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000012  00000000  00000000  000004b6  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  000004c8  2**0
                  CONTENTS, READONLY
 11 .debug_frame  00000060  00000000  00000000  000004fc  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

1-text section: size of instruction code =0x84.

2-data section: size of initialized global array = 0x0 .

3-bss section: size of uninitialized global =0x0.

- **High-Pressure-detect. o:**



```
Pressure_detect.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000098  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000008  00000000  00000000  000000cc  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000d4  2**0
                  ALLOC
  3 .debug_info   00000149  00000000  00000000  000000d4  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 000000bb  00000000  00000000  0000021d  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    00000090  00000000  00000000  000002d8  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  00000368  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   00000060  00000000  00000000  00000388  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str    0000018f  00000000  00000000  000003e8  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000012  00000000  00000000  00000577  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  00000589  2**0
                  CONTENTS, READONLY
 11 .debug_frame  00000060  00000000  00000000  000005bc  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

1-text section: size of instruction code =0x98.

2-data section: size of initialized global array = 0x08.

3-bss section: size of uninitialized global =0x0.

•Alarm-Monitor. o:

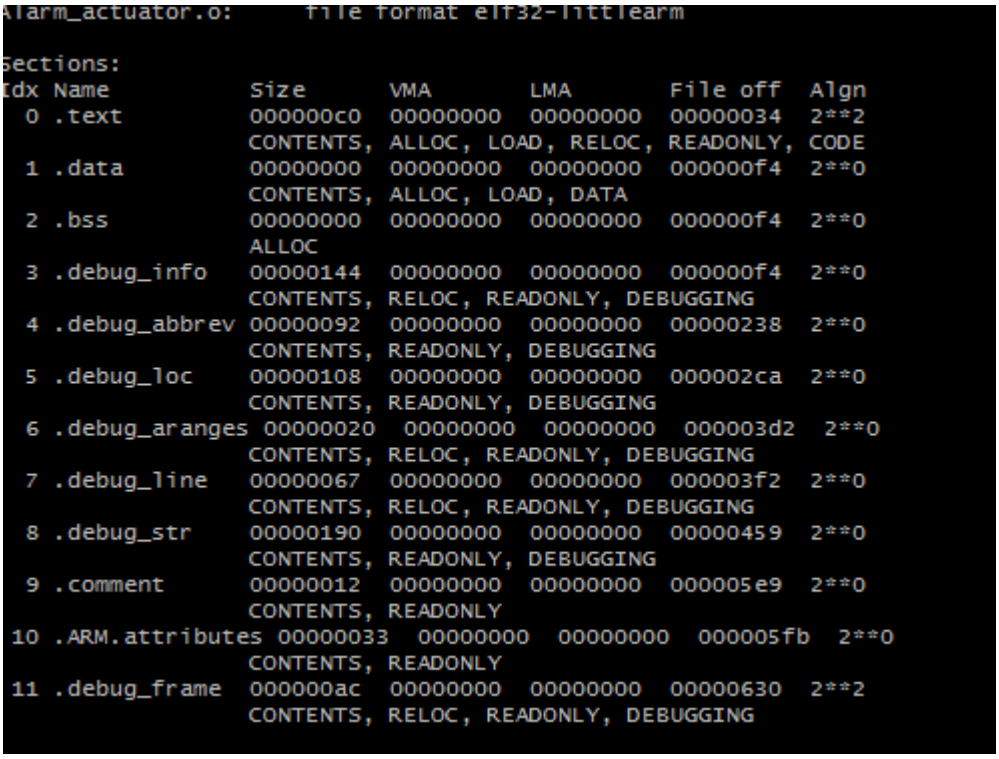
```
Alarm_monitor.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000000d8  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  0000010c  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000004  00000000  00000000  0000010c  2**2
                  ALLOC
  3 .debug_info   0000015b  00000000  00000000  0000010c  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 000000bb  00000000  00000000  00000267  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    000000e8  00000000  00000000  00000322  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  0000040a  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   00000065  00000000  00000000  0000042a  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str    000001b2  00000000  00000000  0000048f  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000012  00000000  00000000  00000641  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  00000653  2**0
                  CONTENTS, READONLY
 11 .debug_frame  00000098  00000000  00000000  00000688  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

1-text section: size of instruction code =0xd8.

2-data section: size of initialized global array = 0x00.

3-bss section: size of uninitialized global =0x04.

- **Alarm-Actuator. o:**



```
Alarm_actuator.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000000c0  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  000000f4  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000f4  2**0
                  ALLOC
  3 .debug_info   00000144  00000000  00000000  000000f4  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 00000092  00000000  00000000  00000238  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    00000108  00000000  00000000  000002ca  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  000003d2  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   00000067  00000000  00000000  000003f2  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str    00000190  00000000  00000000  00000459  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000012  00000000  00000000  000005e9  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  000005fb  2**0
                  CONTENTS, READONLY
 11 .debug_frame  000000ac  00000000  00000000  00000630  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

1-text section: size of instruction code =0xc0.

2-data section: size of initialized global array = 0x00.

3-bss section: size of uninitialized global =0x00.

•driver. o:

```
driver.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000010c  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000000  00000000  00000000  00000140  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  00000140  2**0
                  ALLOC
  3 .debug_info   00000103  00000000  00000000  00000140  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 0000009d  00000000  00000000  00000243  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    000000c8  00000000  00000000  000002e0  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  000003a8  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   00000099  00000000  00000000  000003c8  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str    0000014a  00000000  00000000  00000461  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000012  00000000  00000000  000005ab  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  000005bd  2**0
                  CONTENTS, READONLY
 11 .debug_frame  00000078  00000000  00000000  000005f0  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

1-text section: size of instruction code =0x10c.

2-data section: size of initialized global array = 0x00.

3-bss section: size of uninitialized global =0x00.

- **main. o:**

```
main.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000000a0  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  000000d4  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000d4  2**0
                  ALLOC
  3 .debug_info   00000197  00000000  00000000  000000d4  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 000000a5  00000000  00000000  0000026b  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    00000058  00000000  00000000  00000310  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  00000368  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   000000a6  00000000  00000000  00000388  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str    000001fa  00000000  00000000  0000042e  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000012  00000000  00000000  00000628  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  0000063a  2**0
                  CONTENTS, READONLY
 11 .debug_frame  00000048  00000000  00000000  00000670  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

1-text section: size of instruction code =0xa0.

2-data section: size of initialized global array = 0x00.

3-bss section: size of uninitialized global =0x00.

•Startup. o:

```
startup.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000000c4  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  000000f8  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000f8  2**0
                  ALLOC
  3 .vectors      0000001c  00000000  00000000  000000f8  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, DATA
  4 .debug_info   00000167  00000000  00000000  00000114  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  5 .debug_abbrev 000000c0  00000000  00000000  0000027b  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_loc    00000064  00000000  00000000  0000033b  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_aranges 00000020  00000000  00000000  0000039f  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_line   000000ad  00000000  00000000  000003bf  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  9 .debug_str    00000179  00000000  00000000  0000046c  2**0
                  CONTENTS, READONLY, DEBUGGING
 10 .comment      00000012  00000000  00000000  000005e5  2**0
                  CONTENTS, READONLY
 11 .ARM.attributes 00000033  00000000  00000000  000005f7  2**0
                  CONTENTS, READONLY
 12 .debug_frame  0000004c  00000000  00000000  0000062c  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

1-text section: size of instruction code =0xc4.

2-data section: size of initialized global array = 0x00.

3-bss section: size of uninitialized global =0x00.

4-vectors section: size of vector table=0x1c.

- **Elf image Sections Headers:**

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

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000540  08000000  08000000  00008000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000008  20000000  08000540  00010000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000028  20000008  08000548  00010008  2**2
                  ALLOC
  3 .debug_info   000008fa  00000000  00000000  00010008  2**0
                  CONTENTS, READONLY, DEBUGGING
  4 .debug_abbrev 0000049c  00000000  00000000  00010902  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    00000488  00000000  00000000  00010d9e  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 000000e0  00000000  00000000  00011226  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_line   0000036a  00000000  00000000  00011306  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .debug_str    0000036d  00000000  00000000  00011670  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000011  00000000  00000000  000119dd  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  000119ee  2**0
                  CONTENTS, READONLY
 11 .debug_frame  00000310  00000000  00000000  00011a24  2**2
                  CONTENTS, READONLY, DEBUGGING
```

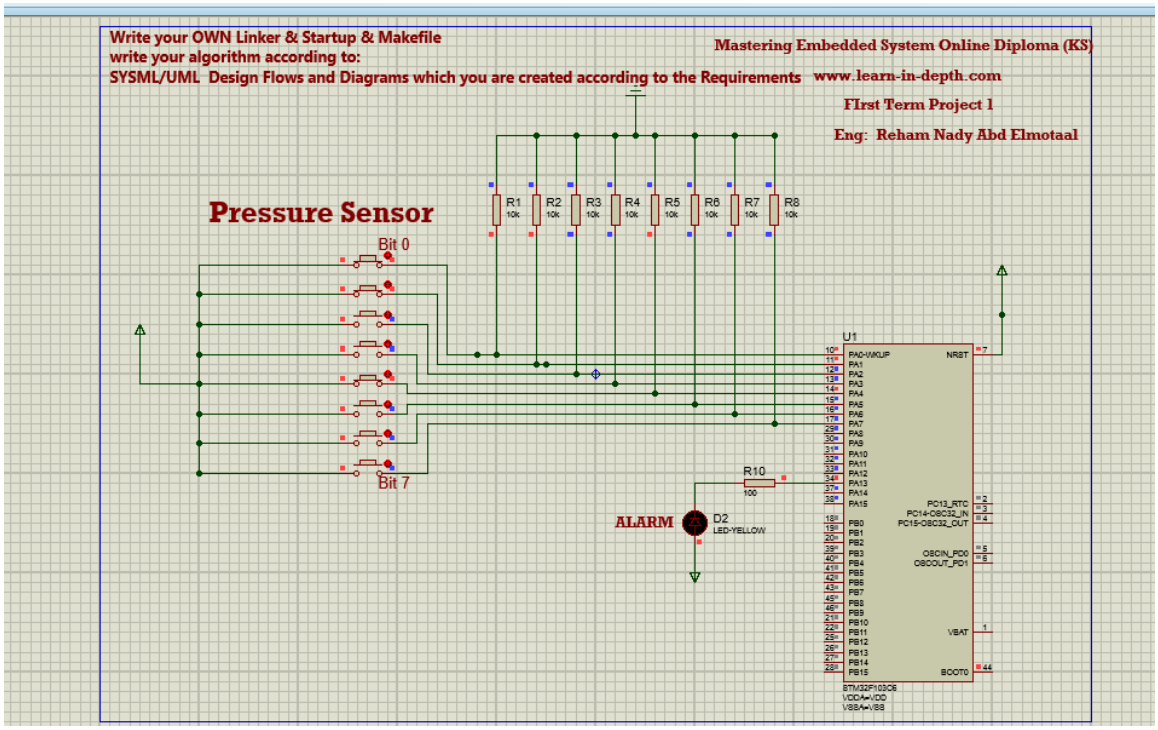
1-text section: size of instruction code =0x540.

2-data section: size of initialized global array = 0x08.

3-bss section: size of uninitialized global =0x28.

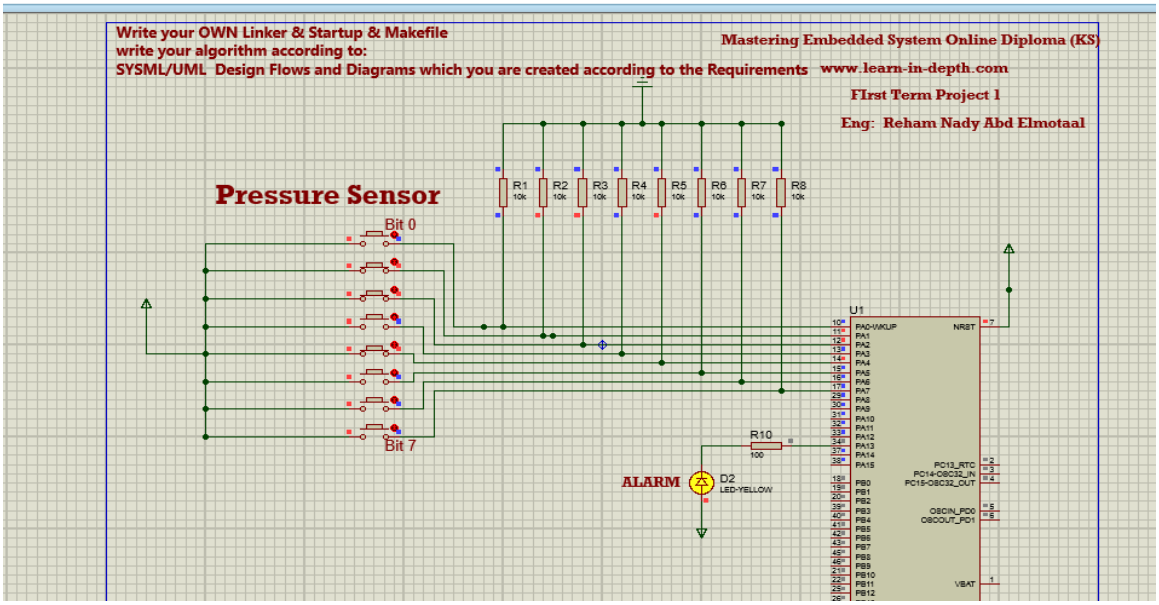
8- Simulation Result on proteus:

- **Pressure < threshold (20 bar):**



pressure = 19.

- **Pressure >threshold (20 bar):**



pressure=22.