Mastering Embedded System Online Diploma

<https://www.learn-in-depth.com/onlinediploma>

Frist Term (Final Project 1)

Ahmed Shawky Abo-muslam

My Profile:

<https://www.learn-in-depth.com/online-diploma/ahmedshawki773%40gmail.com>

Pressure Controlling System

Specification of the system:

1. The System read the pressure values from a sensor .
2. The system inform the crew of a cabin when he detects pressure exceeds 20 bar in the cabin .
3. The system shall starting alarm for 60 seconds after detecting pressure value greater than threshold=20 bar.

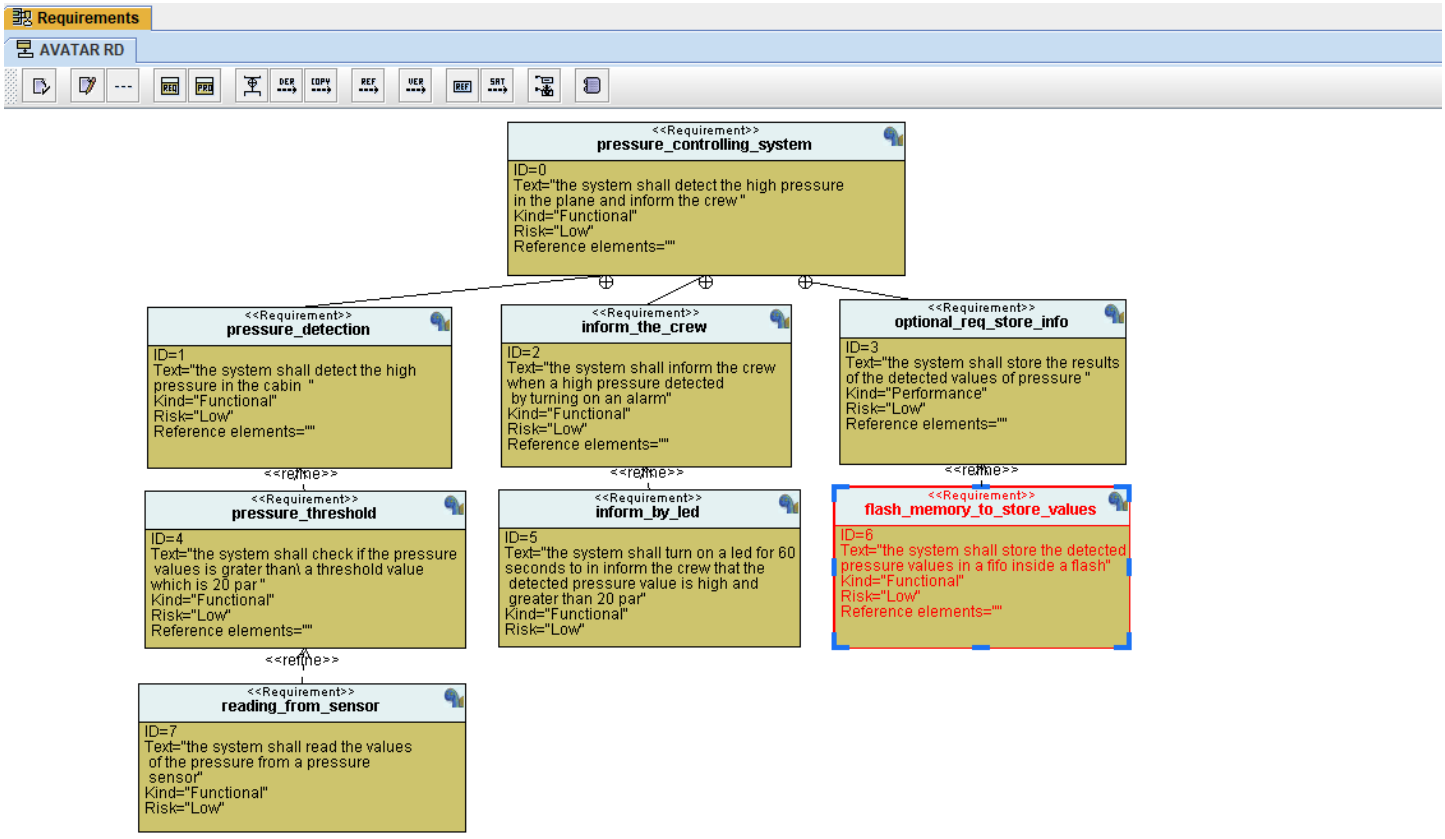
assumptions we should consider:

1.The pressure sensor never fails.

2.The alarm never fails.

3.The controller never faces power cut .

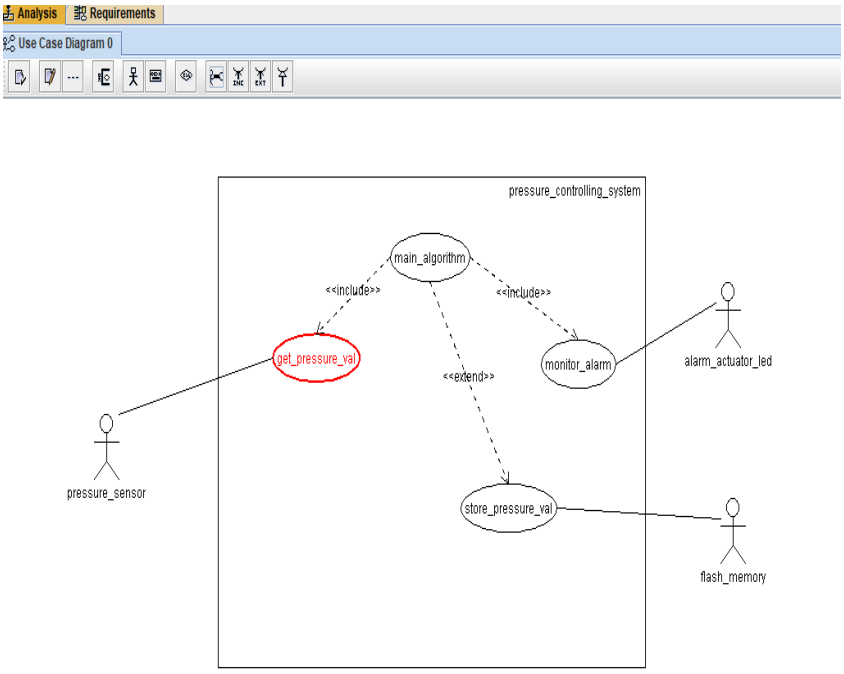
Requirements diagram :



Pressure system will need pressure sensor and led to monitor alarm on it .

System analysis:understanding what a client wants

uml use case diagram:



Use case diagram: describe system

Boundaries and main functions .

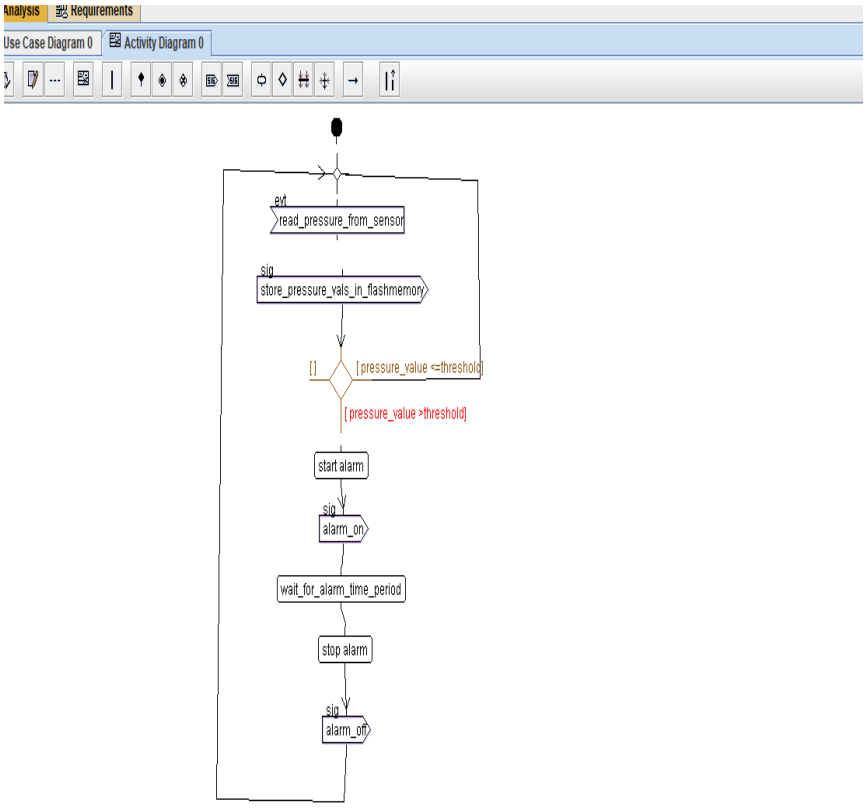
1-get value from a pressure sensor

2-store pressure values in flash memory

2-turn on a led for 60 seconds

If the pressure value exceeds 20 bar

uml activity diagram:



Activity diagram: describe relations between main functions .

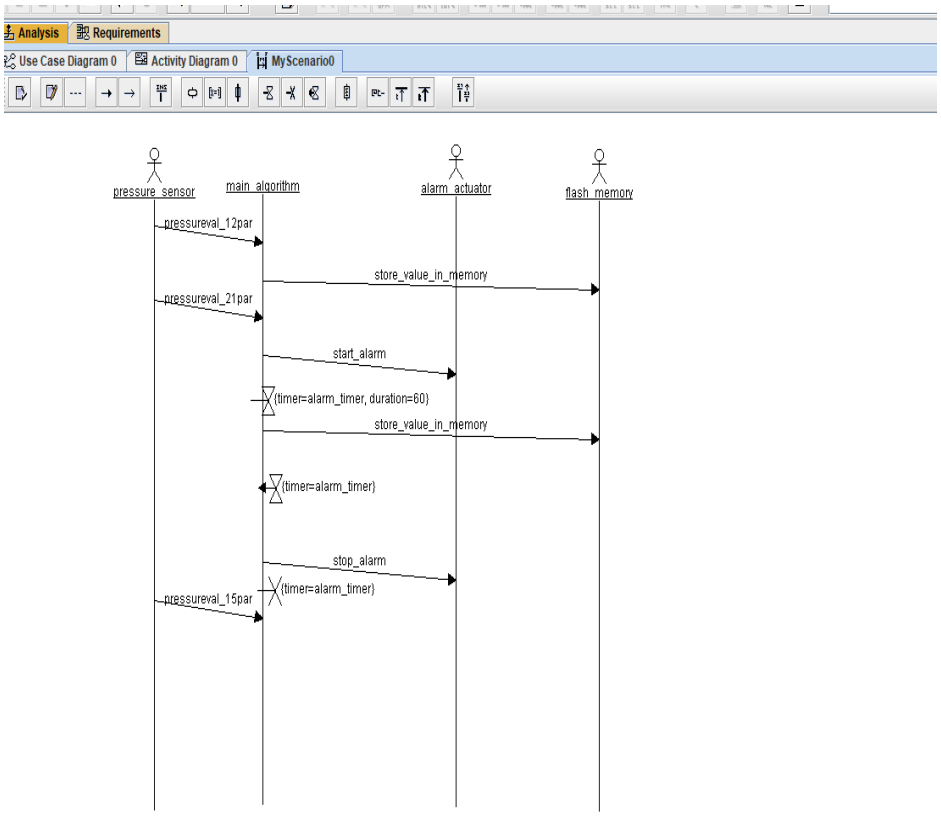
1-read pressure value from sensor

2-store pressure values in flash memory

3-if the pressure values is less than 20 bar The system reads the value again from The pressure sensor .

4- if the pressure values is greater than 20 bar The system start alarm for 60 seconds and after the time is expired reads the pressure value again.

Sequence diagram :

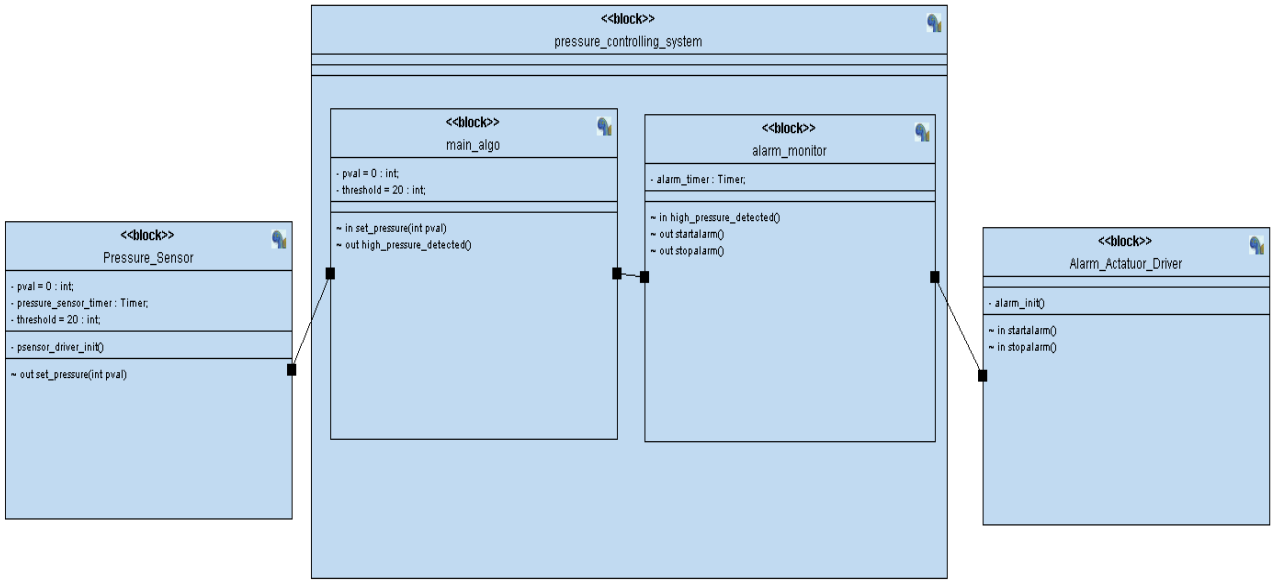


Sequence diagram: describe communications between main system functions and actors.

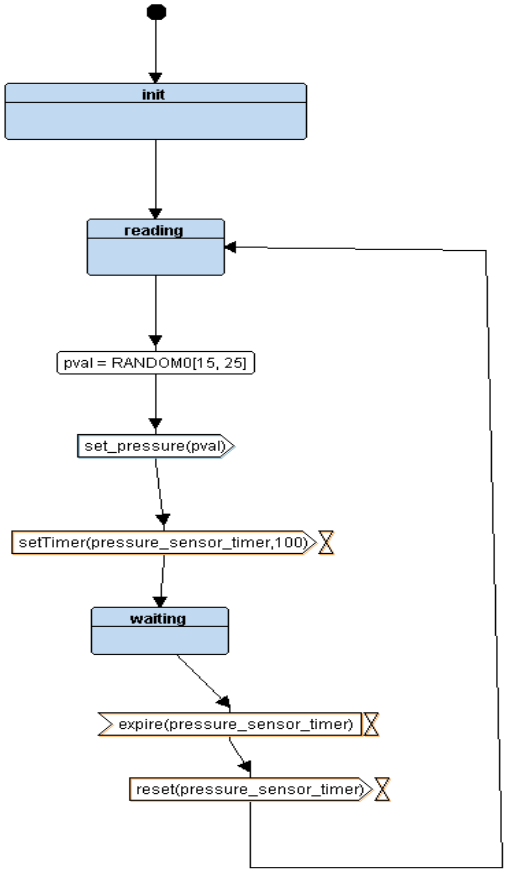
1-if the pressure sensor reads value 12 bar , stores it in flash and nothing will happen .

2- if the pressure sensor reads value 21 bar , which is greater than the threshold value which is 20 bar the system start alarm and initiate timer for 60 seconds and stores value in flash memory.

Design diagram (Block diagram):



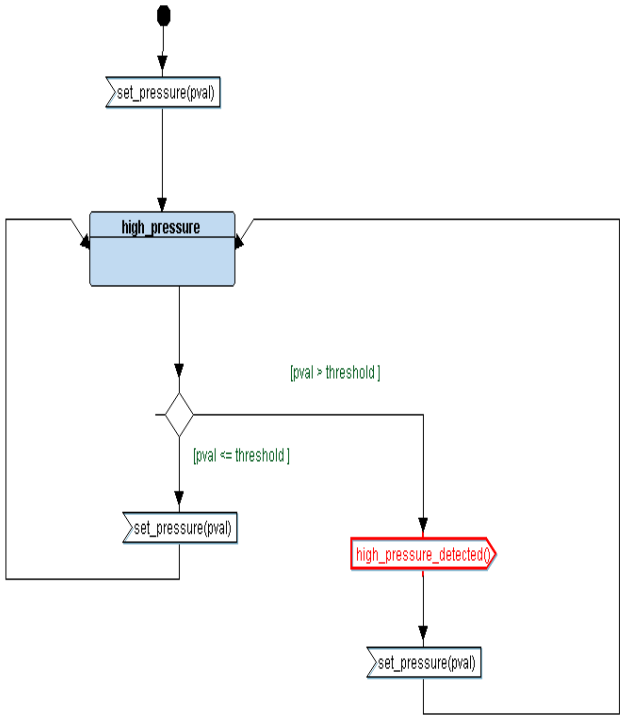
State diagram of Pressure sensor driver Block:



1. Pressure sensor driver is initiated at the beginning of the block .
2. Reads the value from a sensor and send it microcontroller to apply an algorithm on it .
3. Set timer for 100 second and wait until the timer expired then reads the values again

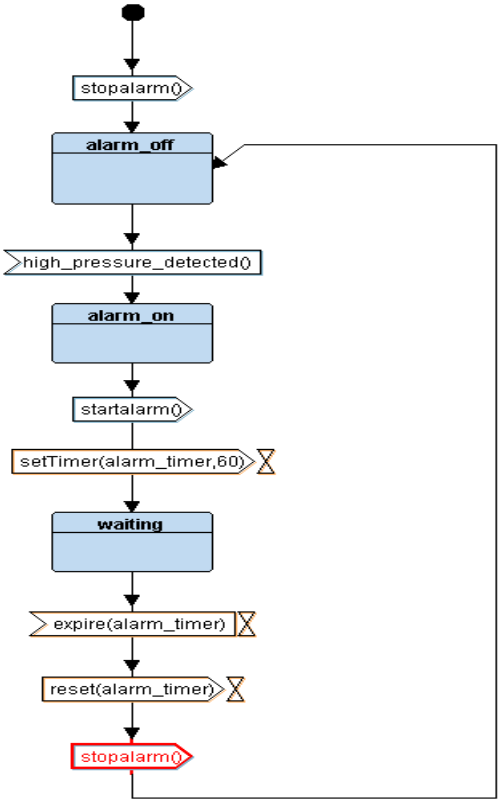
---

State diagram of Main algorithm Block:



1. the microcontroller receives the values of pressure from sensor and apply the algorithm on it.
2. If the pressure values less than the threshold (20 par)it reads the value again .
3. If the pressure values greater than the threshold (20 par), the micrconttroller send signal to alarm monitor module to start alarm for 60 seconds then reads the value again.

State diagram of Alarm monitor Block:

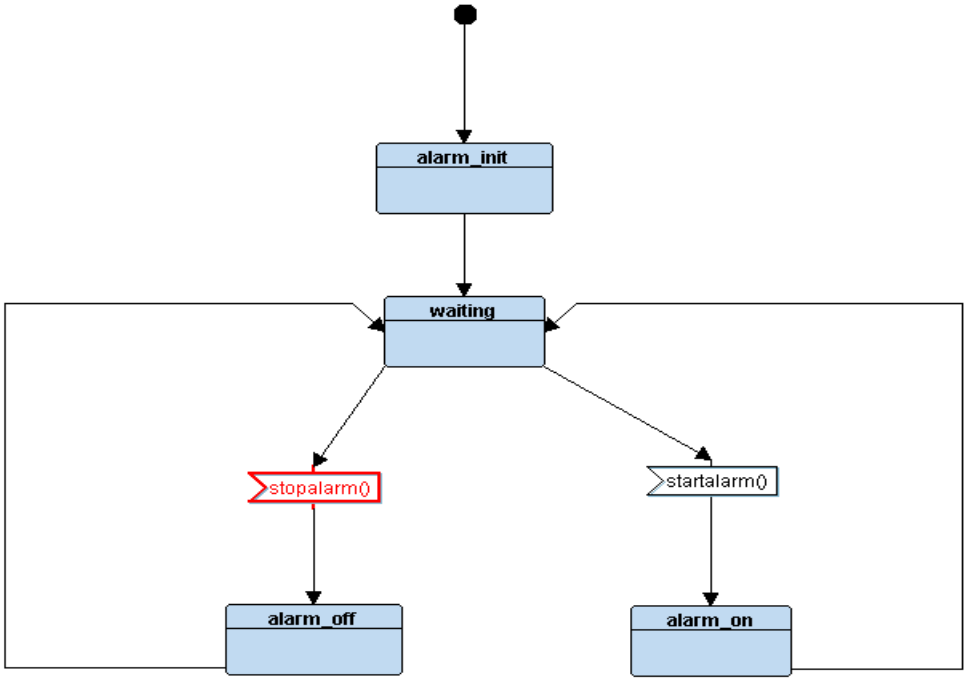


1. The alarm monitor sends signal to make alarm off and stays in alarm off state until it detects a high pressure value then start alarm
2. The alarm stays on for 60 seconds and then stop alarm until the alarm monitor detects a high pressure values
.

State diagram of alarm actuator driver Block:

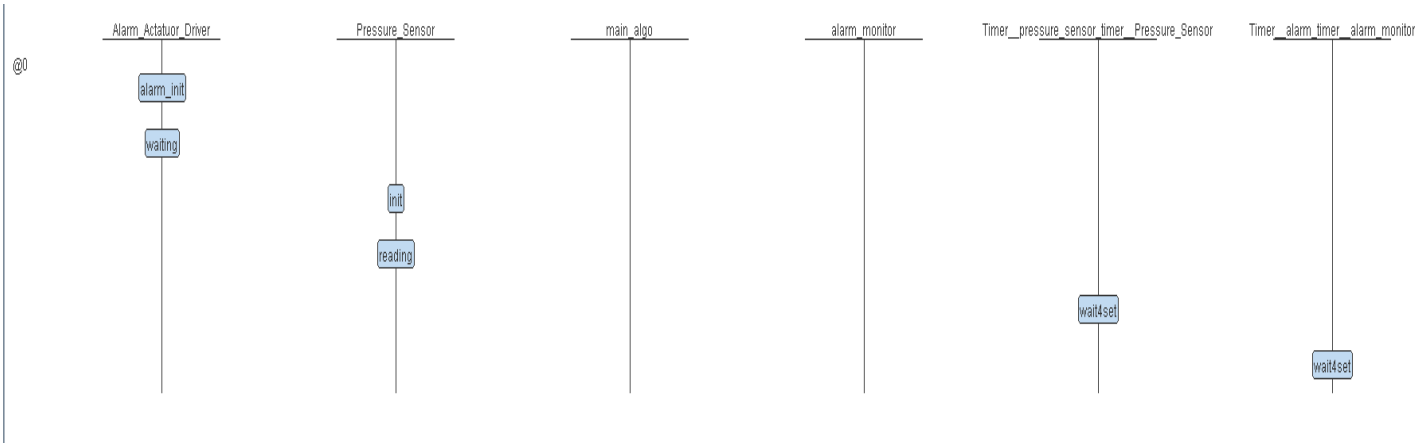
1. The alarm actuator block initiate the driver at the beginning and stays in waiting state until it receives signals
2. If the actuator receives a stop alarm signal it goes to alarm off state then go to waiting state and waits for another signal
3. If the actuator receives a start alarm signal it goes to alarm on state for 60 seconds then go to waiting state and waits for another signal

4

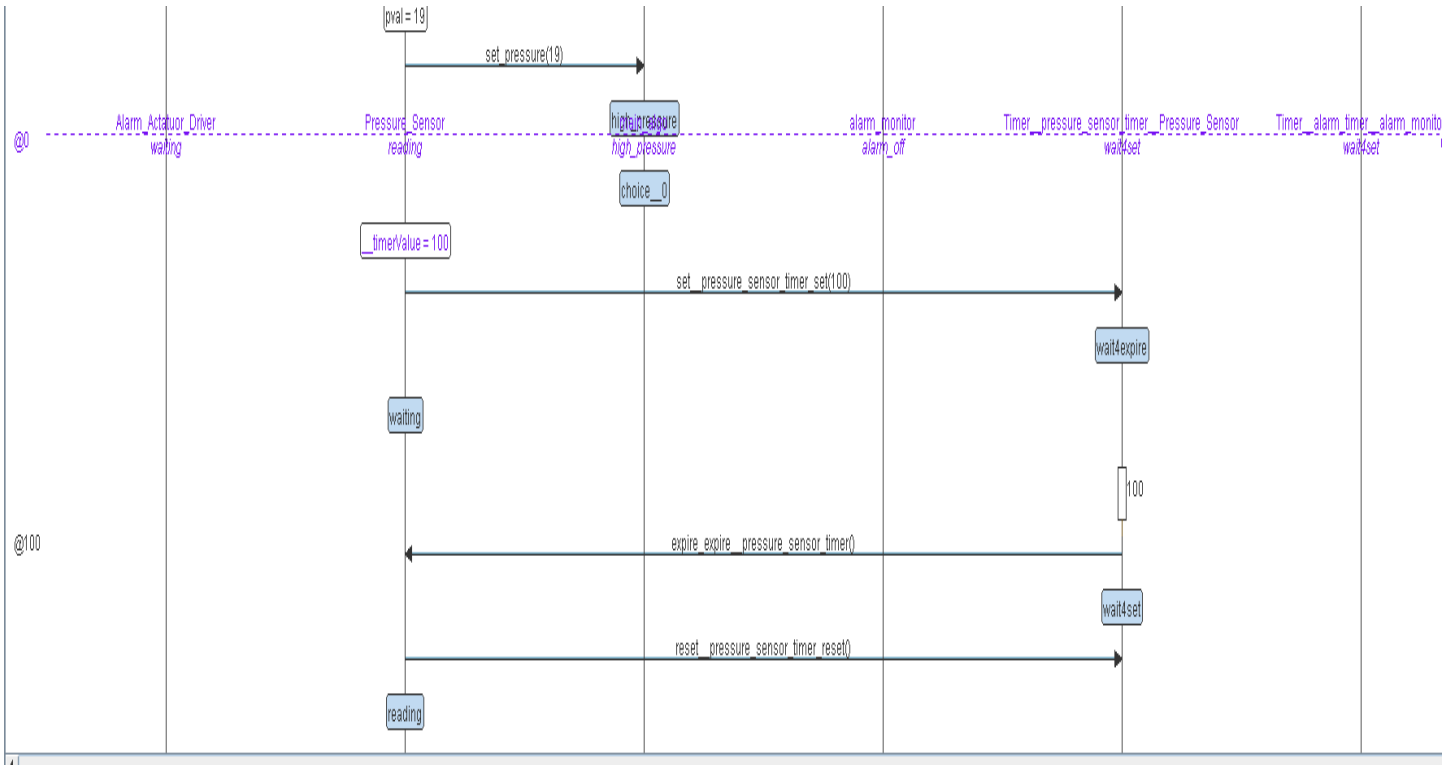


The output of simulation project :

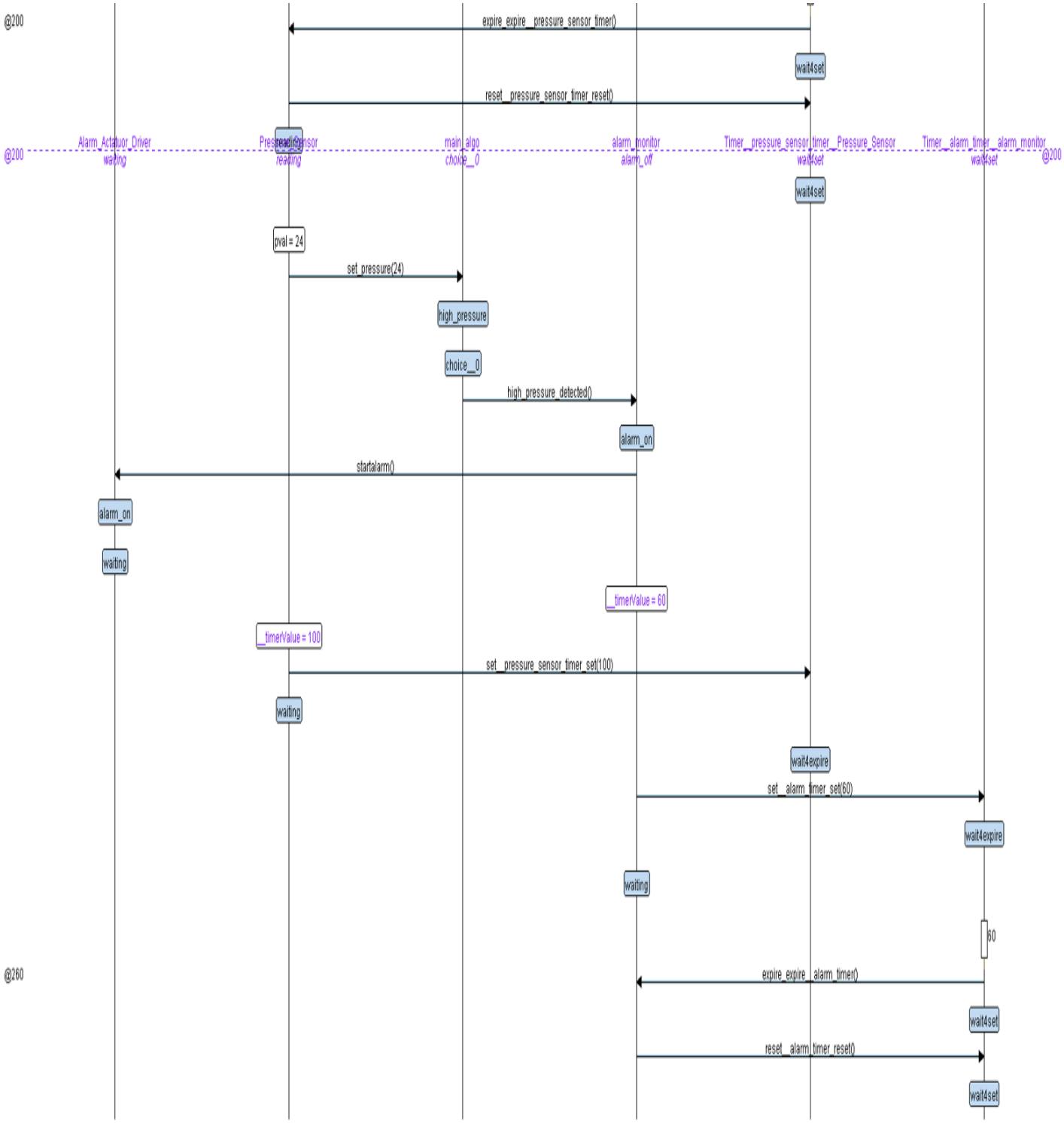
At the start of simulation:



When pressure sensor reads value less than the threshold:



When pressure sensor reads value less than the threshold:


@200
expire_expire__pressure_sensor_timer()
wait4set
reset__pressure_sensor_timer_reset()
Alarm_Actuator_Driver
waiting
Pressure_Sensor
reading
main_algo
choice__0
alarm_monitor
alarm_off
Timer__pressure_sensor_timer__Pressure_Sensor
wait4set
Timer__alarm_timer__alarm_monitor
wait4set
@200
wait4set
pval = 24
set_pressure(24)
high_pressure
choice__0
high_pressure_detected()
alarm_on
startalarm()
alarm_on
waiting
__timerValue = 60
__timerValue = 100
set__pressure_sensor_timer_set(100)
waiting
wait4expire
set__alarm_timer_set(60)
wait4expire
waiting
60
@260
expire_expire__alarm_timer()
wait4set
reset__alarm_timer_reset()
wait4set

Symbol for GPIO_Driver:

```
$ arm-none-eabi-nm.exe driver.o
00000000 T Delay
00000020 T getPressureVal
00000074 T GPIO_INITIALIZATION
00000038 T Set_Alarm_actuator
```

symbol_for_pressure_sensor

```
$ arm-none-eabi-nm.exe pressure_sensor.o
         U getPressureVal
00000000 T pressure_sensor_init
00000004 C PS_state
00000001 C PS_state_id
00000000 B pval
         U set_pressure
0000000c T st_PS_reading
```

symbol_for_main_controller:

```
$ arm-none-eabi-nm.exe main_algo.o
         U high_pressure_detected
00000004 C MA_state
00000001 C MA_state_id
00000000 B pressurevalue
00000040 T set_pressure
00000000 T st_MA_high_pressure
00000000 D threshold
```

symbol_for_alarm monitor:

```
arm-none-eabi-nm.exe alarm_monitor.o
00000004 C AM_state
00000001 C AM_state_id
         U Delay
00000050 T high_pressure_detected
         U Set_Alarm_actuator
00000000 T st_AM_alarmoff
00000018 T st_AM_alarmon
```

symbol_for_application:

```
$ arm-none-eabi-nm.exe main.o
00000004 C AM_state
00000001 C AM_state_id
         U GPIO_INITIALIZATION
00000004 C MA_state
00000001 C MA_state_id
00000038 T main
00000004 C PS_state
00000001 C PS_state_id
00000000 T setup
         U st_AM_alarmoff
         U st_MA_high_pressure
         U st_PS_reading
```

symbol for exectuable file at the physical addresses on board:

```
$ arm-none-eabi-nm.exe  pressure_controller.elf
08000288 t _reset
2000000c B AM_state
20000008 B AM_state_id
080000c4 T Delay
080000e4 T getPressureVal
08000138 T GPIO_INITIALIZATION
080000a8 T high_pressure_detected
20000010 B MA_state
20000018 B MA_state_id
080001c0 T main
08000244 T pressure_sensor_init
20000000 B pressurevalue
20000014 B PS_state
20000019 B PS_state_id
20000004 B pval
080000fc T Set_Alarm_actuator
08000228 T set_pressure
08000188 T setup
08000058 T st_AM_alarmoff
08000070 T st_AM_alarmon
080001e8 T st_MA_high_pressure
08000250 T st_PS_reading
08000290 D threshold
0800028e t vector_handler
```

section for GPIO Driver:

```
$ arm-none-eabi-objdump.exe -h driver.o

driver.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, READONLY, CODE
  1 .data         00000000  00000000  00000000  000000f8  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000f8  2**0
                  ALLOC
  3 .debug_info   00000a05  00000000  00000000  000000f8  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 000001de  00000000  00000000  00000afd  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    00000140  00000000  00000000  00000cdb  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  00000e1b  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   000001a7  00000000  00000000  00000e3b  2**0
```

section for main controller:

```
$ arm-none-eabi-objdump.exe -h main_algo.o

main_algo.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000005c  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000004  00000000  00000000  00000090  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000004  00000000  00000000  00000094  2**2
                  ALLOC
  3 .debug_info   00000a1c  00000000  00000000  00000094  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 000001d6  00000000  00000000  00000ab0  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    00000088  00000000  00000000  00000c86  2**0
                  CONTENTS, READONLY, DEBUGGING
```

section for alarm monitor:

```
arm-none-eabi-objdump.exe -h alarm_monitor.o

larm_monitor.o:     file format elf32-littlearm

ections:
dx Name          Size      VMA       LMA       File off  Algn
 0 .text         0000006c  00000000  00000000  00000034  2**2
                 CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
 1 .data         00000000  00000000  00000000  000000a0  2**0
                 CONTENTS, ALLOC, LOAD, DATA
 2 .bss          00000000  00000000  00000000  000000a0  2**0
                 ALLOC
 3 .debug_info   00000a05  00000000  00000000  000000a0  2**0
                 CONTENTS, RELOC, READONLY, DEBUGGING
 4 .debug_abbrev 000001c3  00000000  00000000  00000aa5  2**0
                 CONTENTS, READONLY, DEBUGGING
 5 .debug_loc    0000009c  00000000  00000000  00000c68  2**0
                 CONTENTS, READONLY, DEBUGGING
 6 .debug_aranges 00000020  00000000  00000000  00000d04  2**0
                 CONTENTS, RELOC, READONLY, DEBUGGING
 7 .debug_line   0000013a  00000000  00000000  00000d24  2**0
                 CONTENTS, RELOC, READONLY, DEBUGGING
```

section for main application:

```
$ arm-none-eabi-objdump.exe -h main.o

main.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Alg
  0 .text         00000060  00000000  00000000  00000034  2**
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, COD
  1 .data         00000000  00000000  00000000  00000094  2**
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  00000094  2**
                  ALLOC
  3 .debug_info   00000a66  00000000  00000000  00000094  2**
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 000001ad  00000000  00000000  00000afa  2**
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    00000058  00000000  00000000  00000ca7  2**
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  00000cff  2*
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   00000161  00000000  00000000  00000d1f  2**
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str    000005af  00000000  00000000  00000e80  2**
```

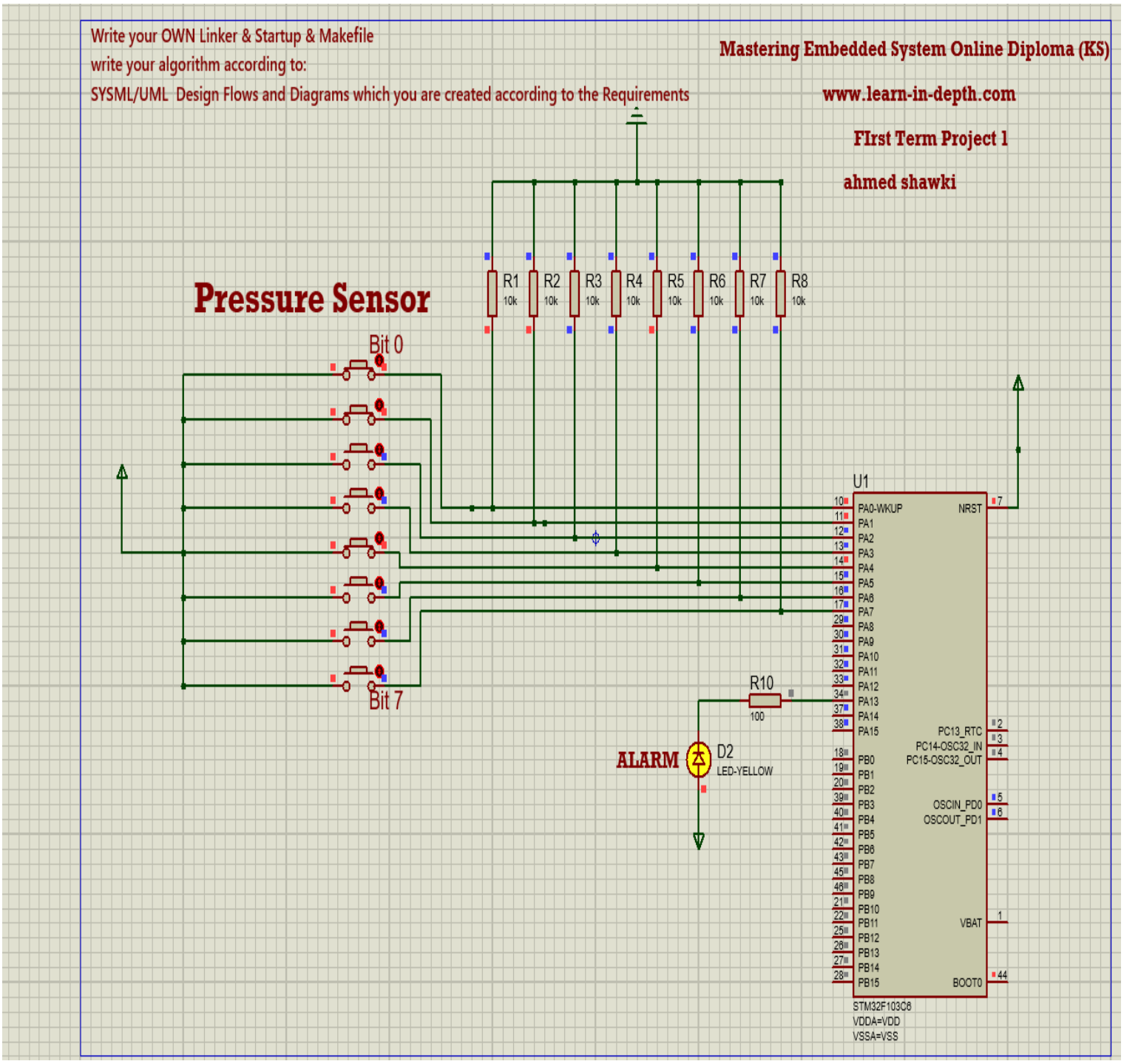
section for pressure controller at the loading address in flash memory and at run time in ram:

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

pressure_controller.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000002e4  08000000  08000000  00010000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000004  20000000  080002e4  00020000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          0000101a  20000004  080002e8  00020004  2**2
                  ALLOC
  3 .debug_info   00003cf5  00000000  00000000  00020004  2**0
                  CONTENTS, READONLY, DEBUGGING
  4 .debug_abbrev 00000ab3  00000000  00000000  00023cf9  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    000003a8  00000000  00000000  000247ac  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 000000c0  00000000  00000000  00024b54  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_line   000007fa  00000000  00000000  00024c14  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .debug_str    000006be  00000000  00000000  0002540e  2**0
```

The output of simulation on protus when pressure(17) less than threshold



The output of simulation on protus when pressure(25) greater than threshold