Mastering Embedded System Online Diploma

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

First Term (Final Project 1)

Eng. Mohamed Abd El-Naby Mohamed

My Profile:

https://www.learn-in-depth.com/online-diploma/mahameda.naby@gmail.com

Table of Contents

LIST OF FIGURES

Pressure Detection System

Description

This System deliver software that detect high pressure and alarm if pressure is high.

System Specifications

1- Pressure detection at 60 bar inform with alarm.
2- Alarm duration 60 second
3- Optional Keeps track the measured values.

System Assumptions:

1- Controller setup and shutdown procedure are not modeled.
2- Controller maintenance is not modeled.
3- Pressure sensor never fails.
4- Alarm never fails.
5- All components never face power cut.

System Architecture

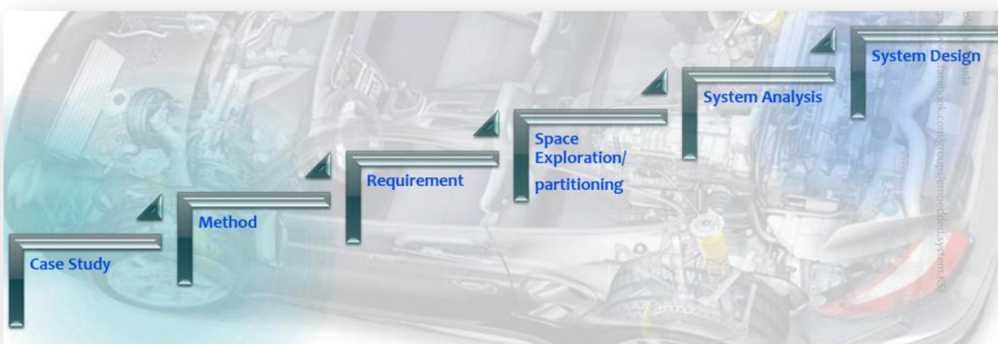


Figure 1:System Architecture

1- Case study

software that detects high pressure and alarm for info.

2- Method

Using V-Model

3- Requirement

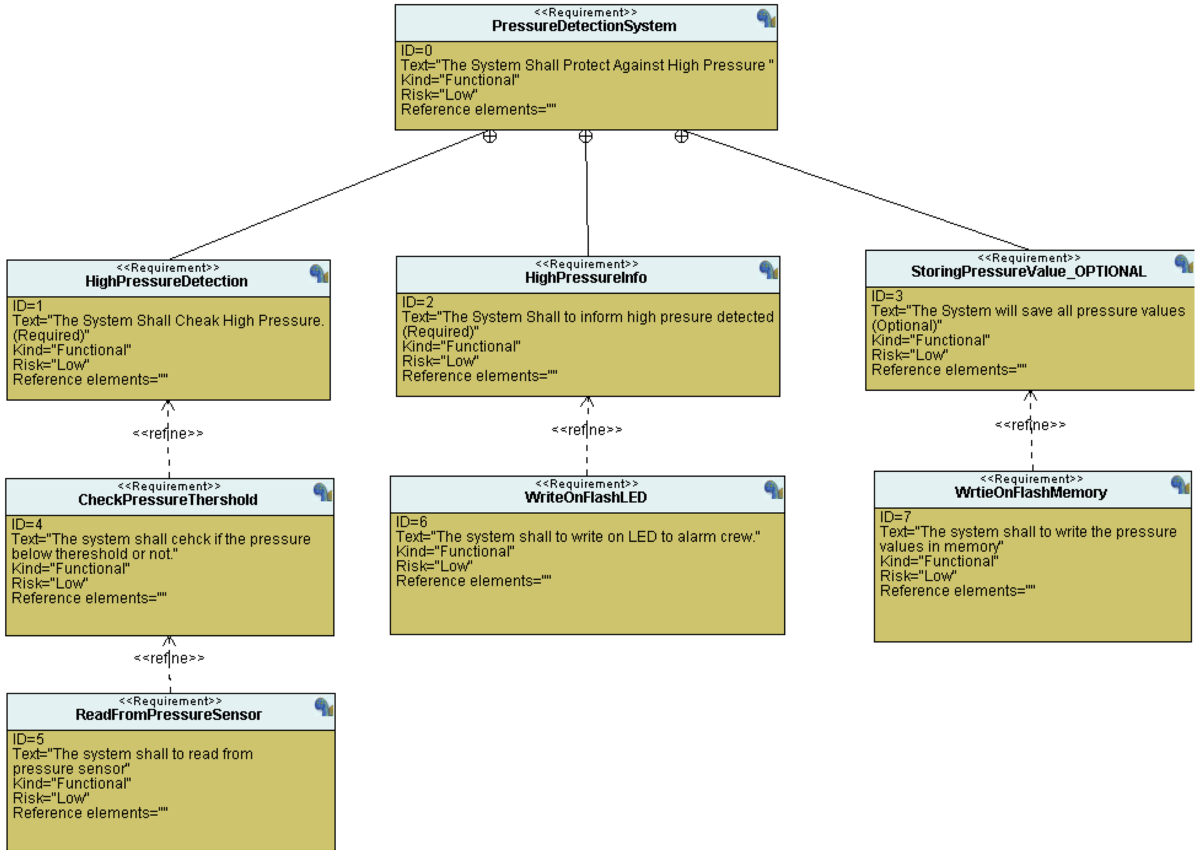


Figure 2:System Requirement

4- Space exploration/partitioning

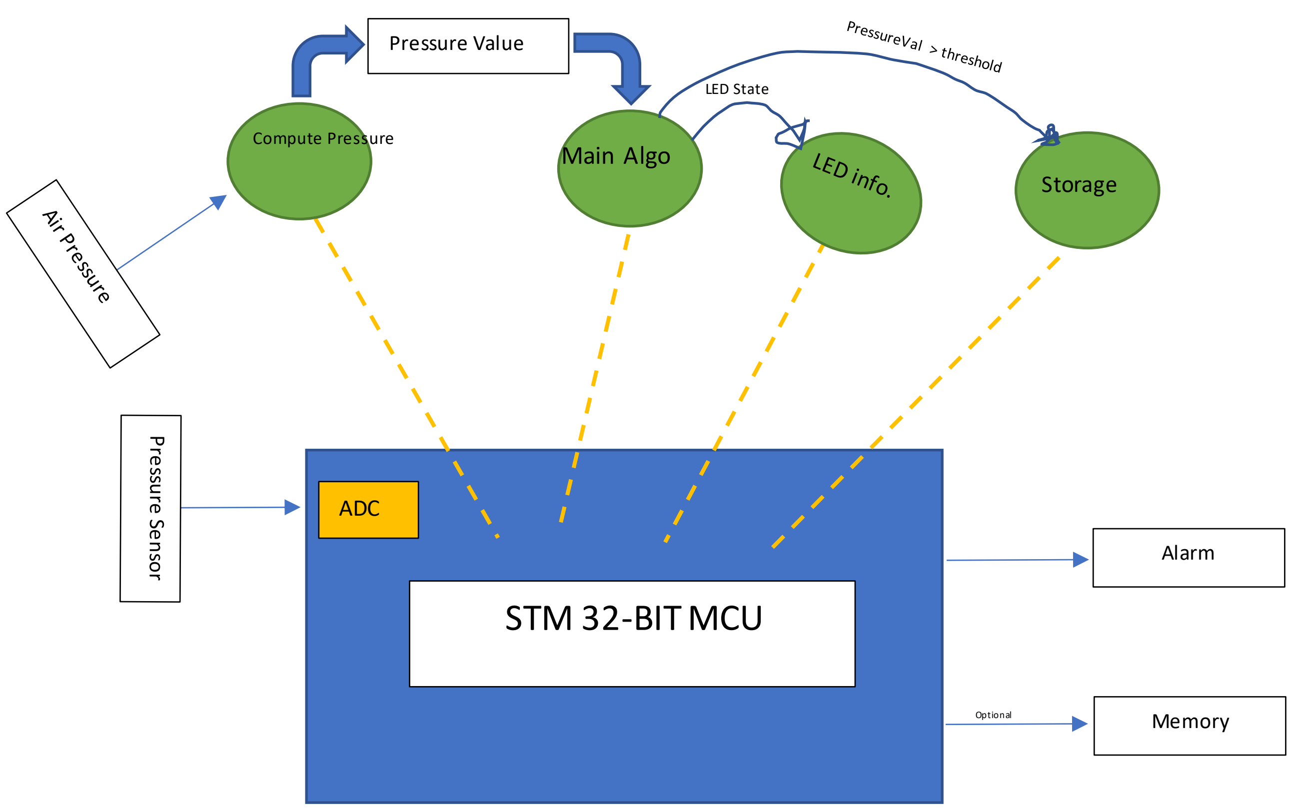


Figure 3:System Partitioning

I used STM32F103C6 MCU which based on ARM Cortex m3 microprocessor its specification

1- ARM 32-bit Cortex™-M3 CPU Core
   i) 72 MHz maximum frequency
   ii) Single-cycle multiplication and hardware division.
2- Memories
   i) 32 Kbytes of Flash memory
   ii) 10 Kbytes of SRAM
3- Clock, reset and supply management
   i) 2.0 to 3.6 V application supply and I/Os.
   ii) 4-to-16 MHz crystal oscillator.
   iii) 32 kHz oscillator for RTC with calibration

5- System Analysis

i- Use Case Diagram

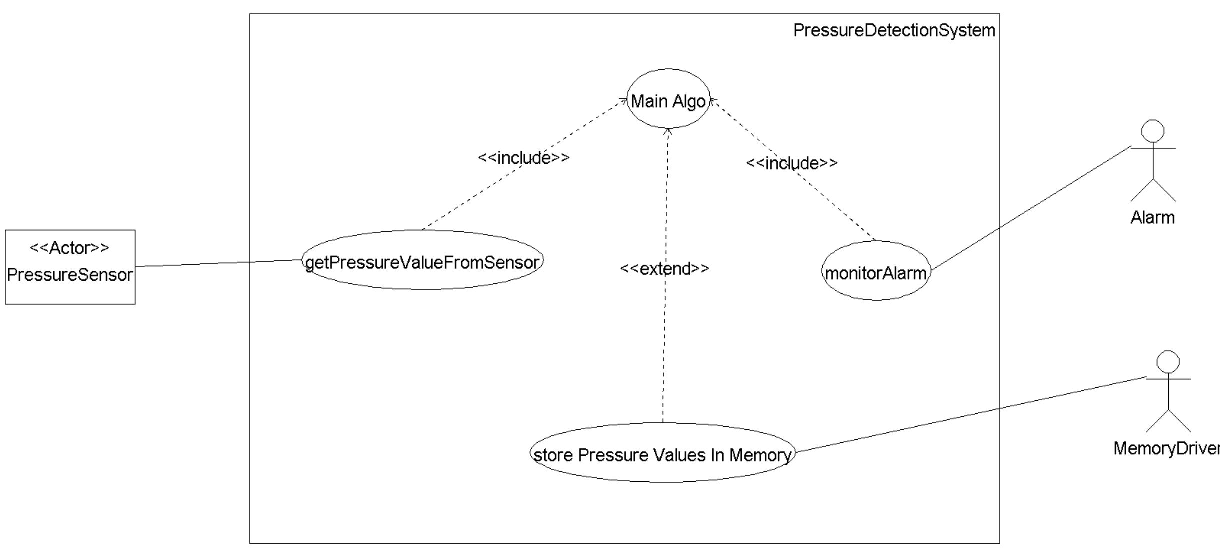


Figure 4:Use Case Diagram

ii- Activity Diagram

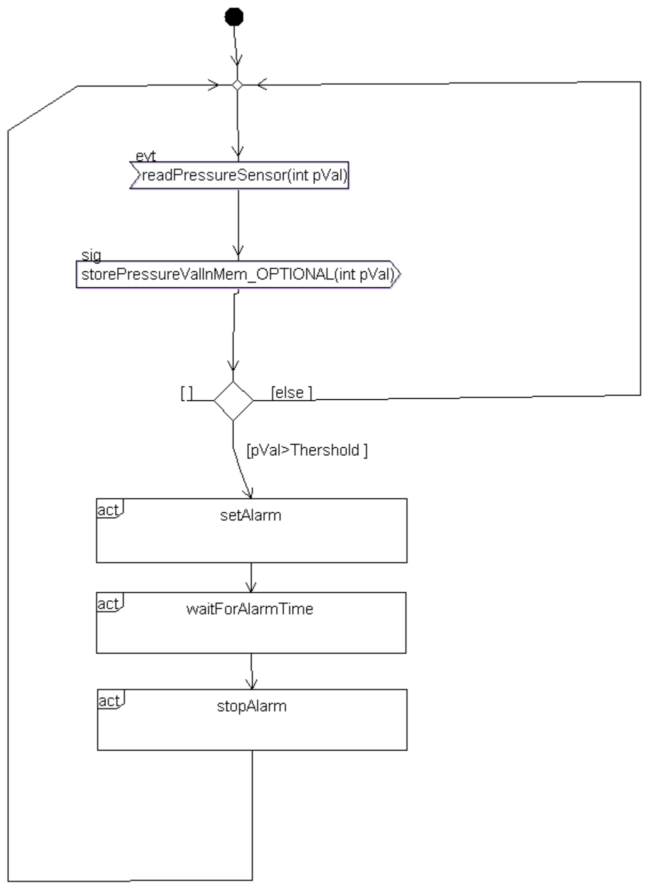


Figure 5:Activity Diagram

iii- Sequence Diagram (UML)

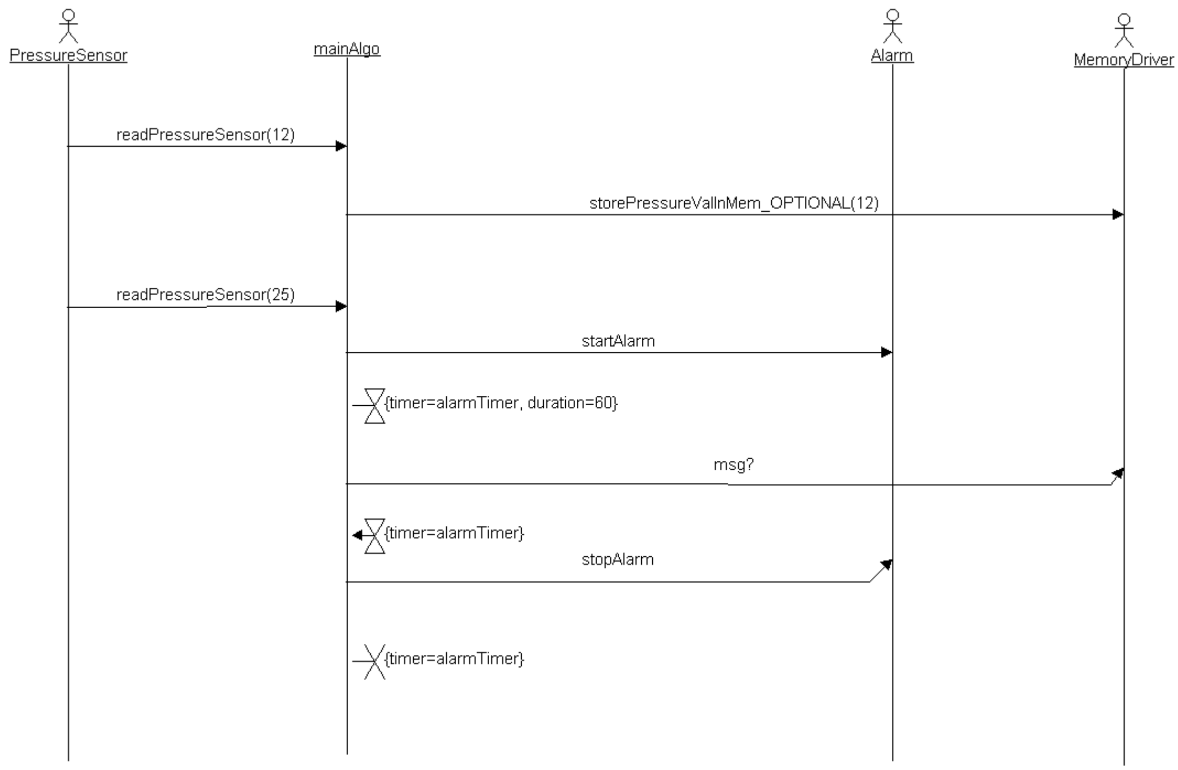


Figure 6:UML Diagram

6- System Design

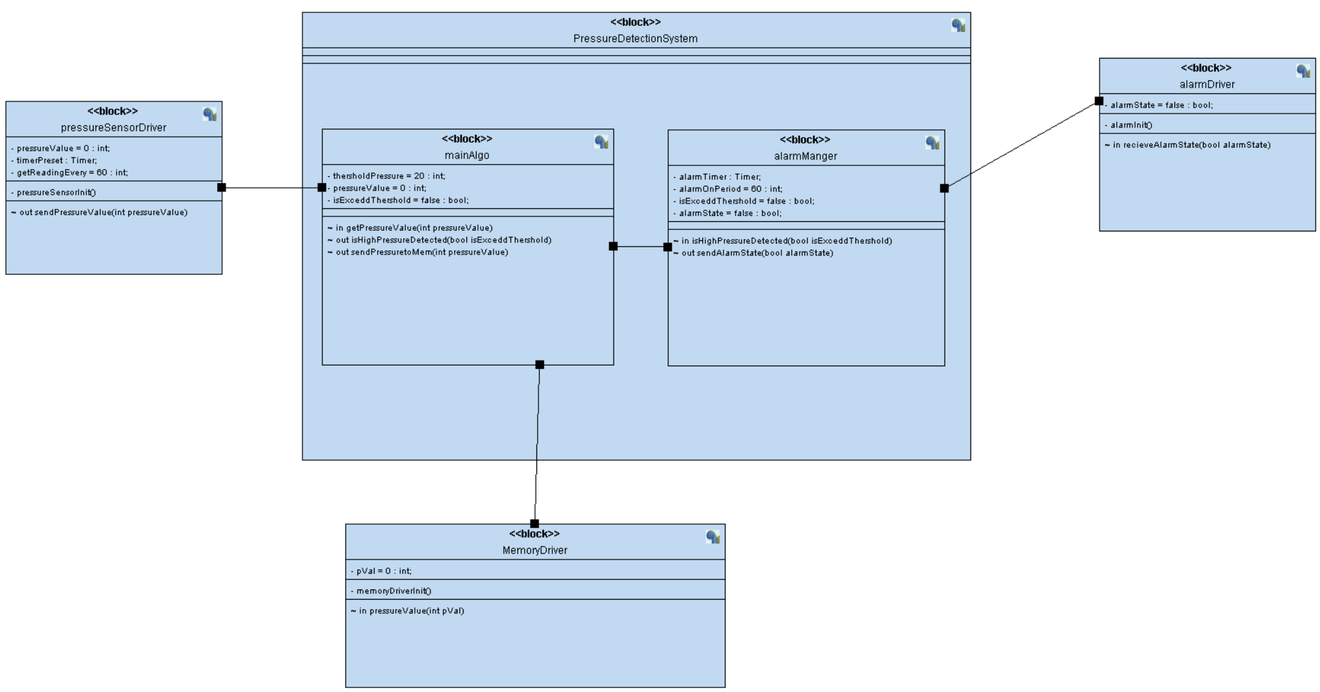


Figure 7:System Design

System Design in Details

1- Pressure Sensor Logic

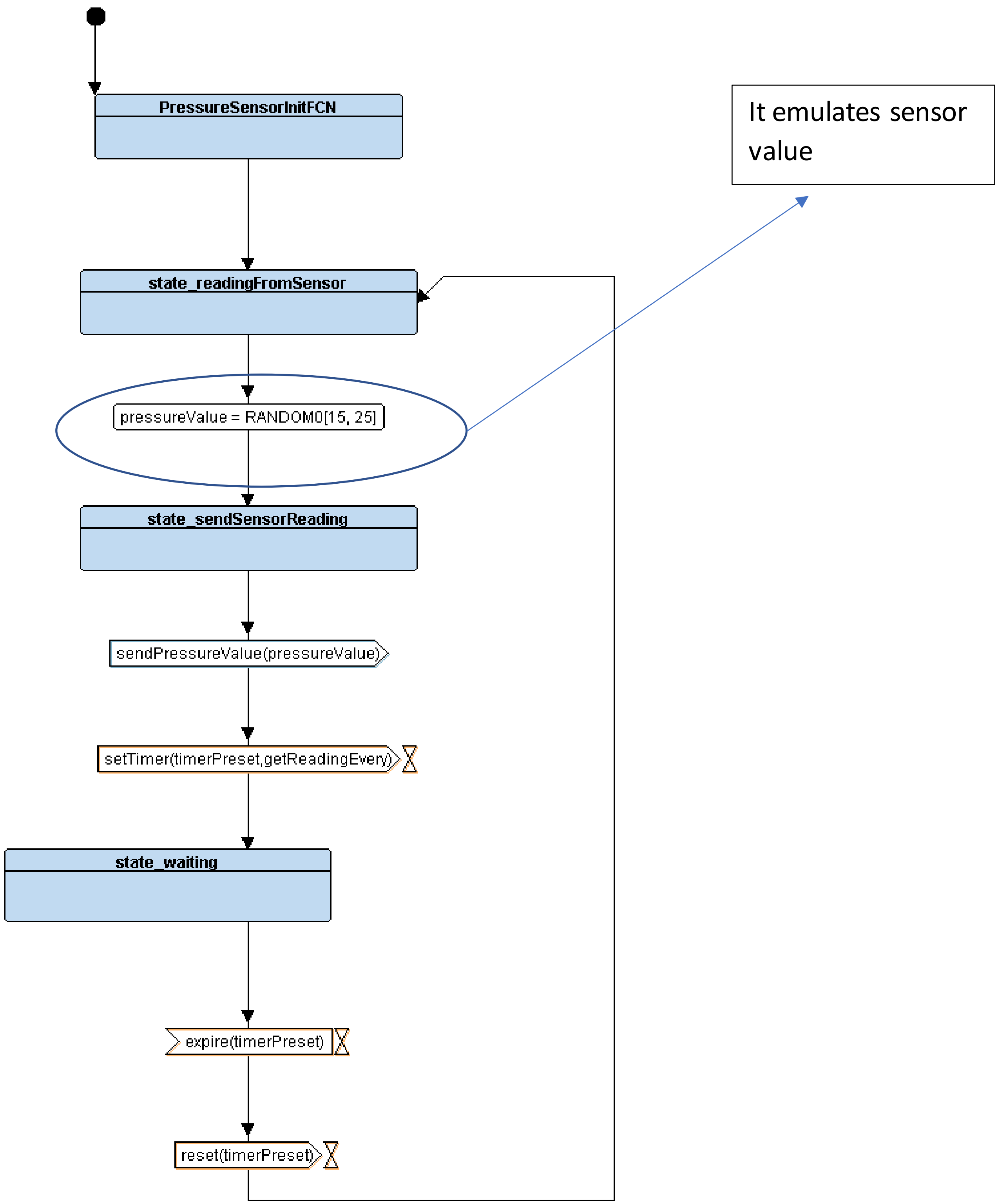


Figure 8:Pressure Sensor Logic

2- Main Algo

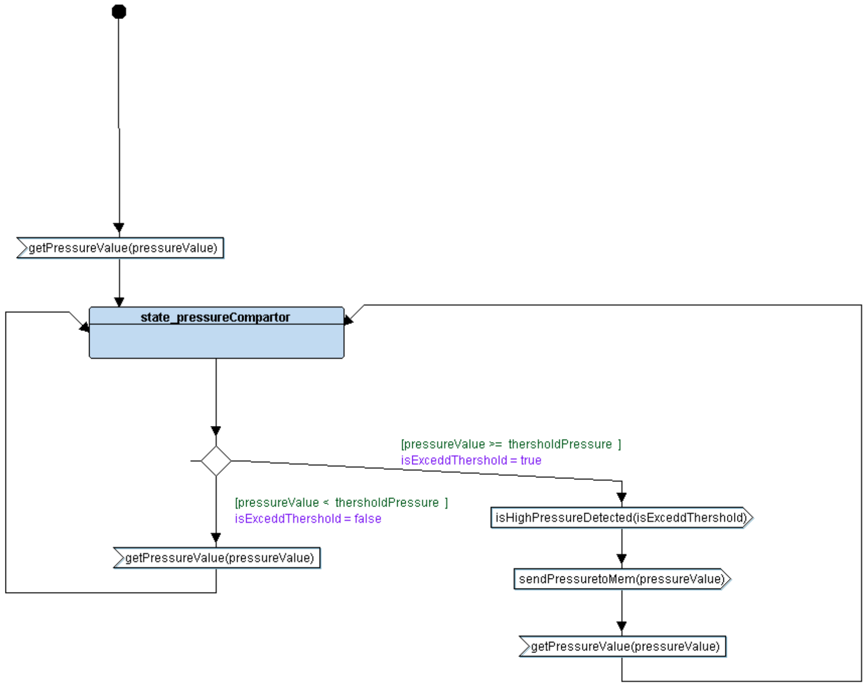


Figure 9:Main Computational Logic

3- Memory Driver Logic

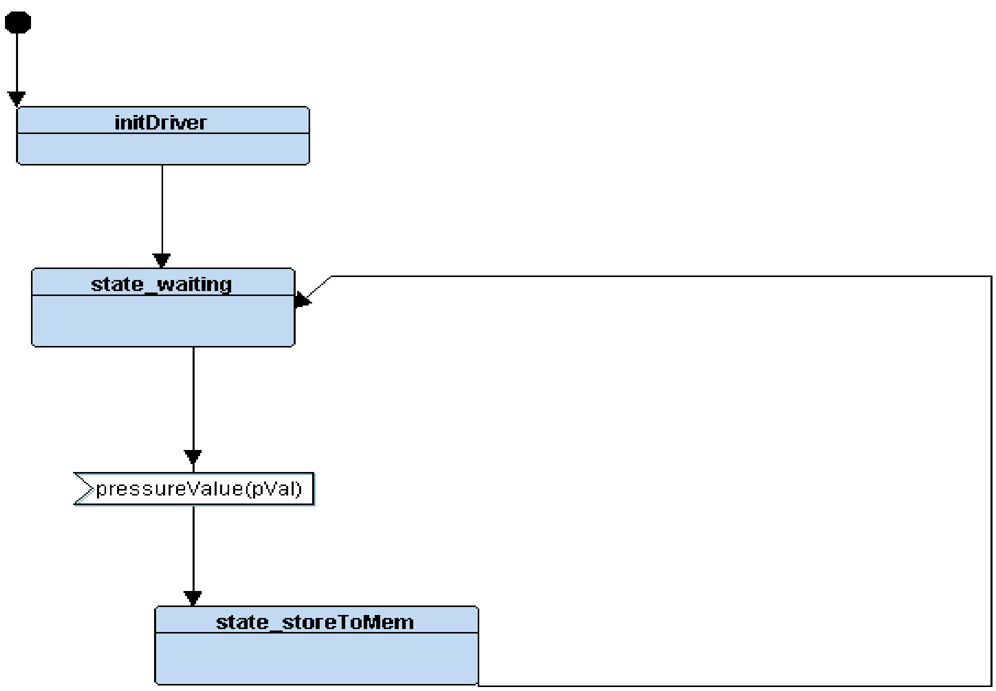


Figure 10:Memory Storing Logic

4- Alarm Manger Logic

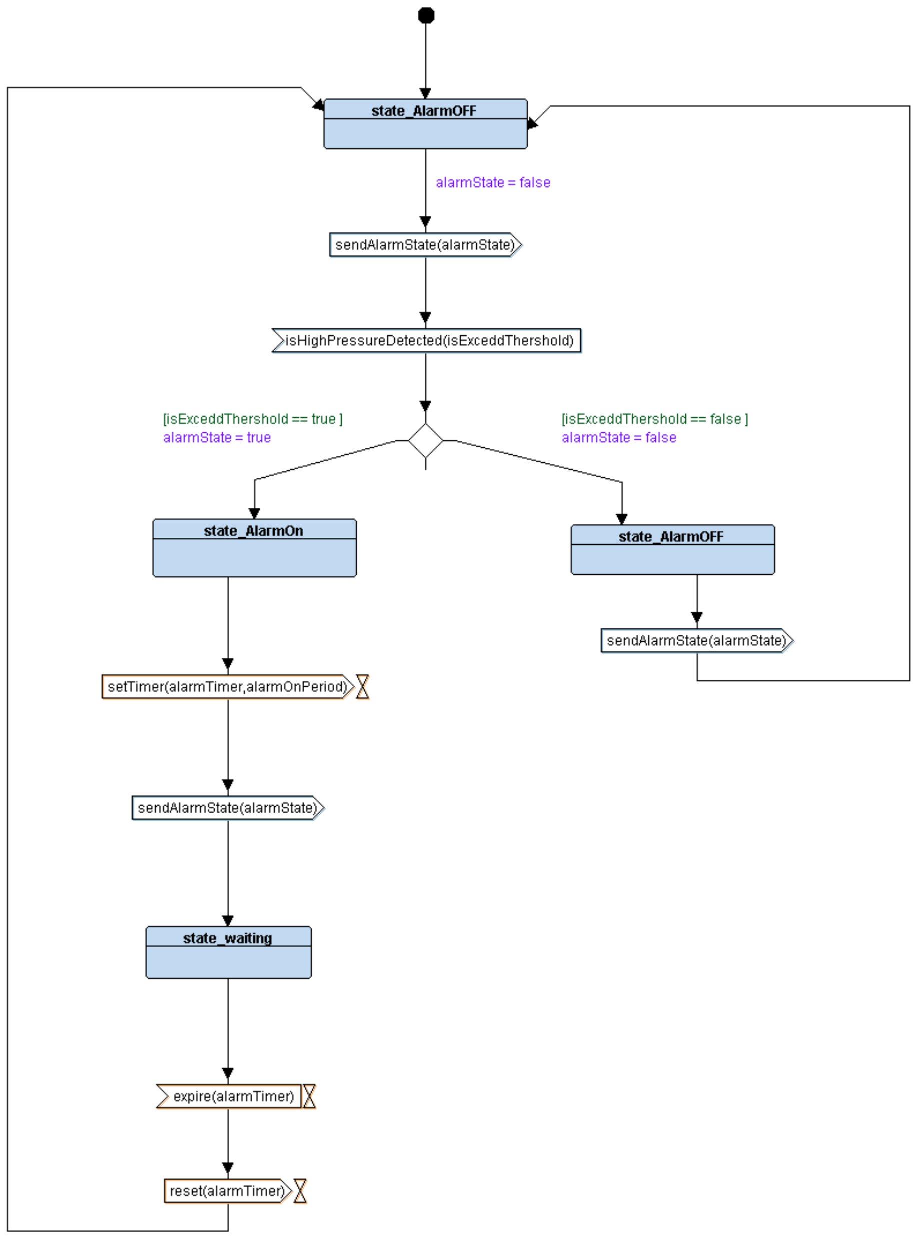


Figure 11:Alarm Manger Logic

5- Alarm Driver

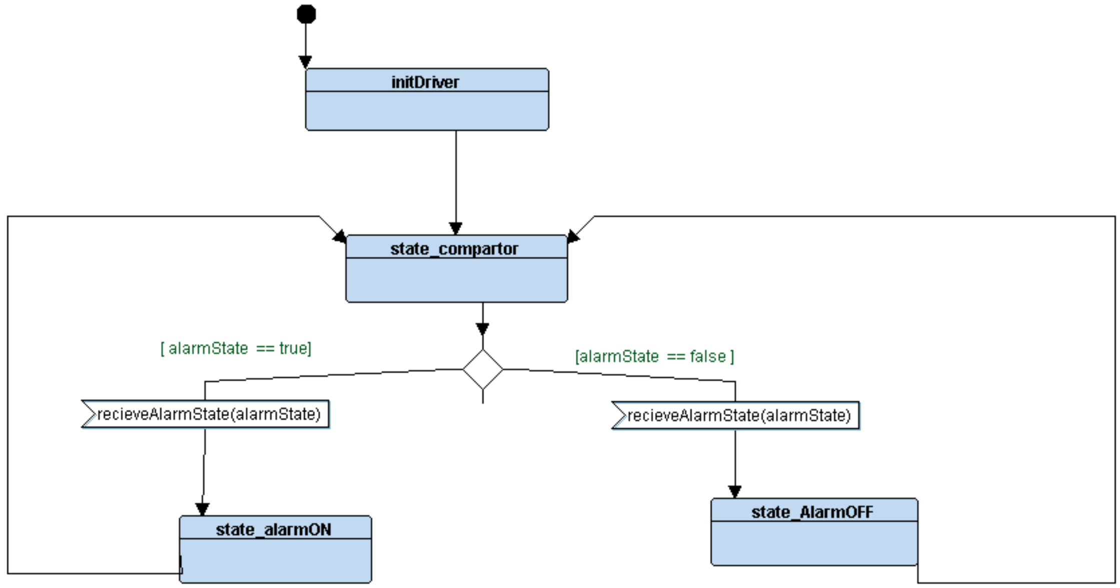


Figure 12:Alarm Logic

Blocks Implementation in C

1- Pressure Sensor

```c
#include "pressureSensor.h"
/*******************************PRIVATE GLOBAL VARs****************/
suint8 Global_suint8PressureValue;
void (*pointerToState_PS)(void);
void pressureSensorInit(void)
{
    // Init SENSOR Driver

    // Init First State
    pointerToState_PS = STATE(PS_readingSensor);
}
STATE_DEFINE(PS_readingSensor)
{
    // Read From Sensor
    Global_suint8PressureValue = getPressureVal();
    // Next time will send reading
    pointerToState_PS = STATE(PS_SendReading);


}
STATE_DEFINE(PS_SendReading)
{
    // start send reading to main algo
    PressureValue_COM(Global_suint8PressureValue);
    // Goto waiting state
    pointerToState_PS = STATE(PS_waiting);
}
STATE_DEFINE(PS_waiting)
{
    // wait for 60 sec timer then goto PS_readingSensor state
    // emulate delay func
    volatile int i = 6000;
    Delay(i);
    pointerToState_PS = STATE(PS_readingSensor);
}
```

Figure 13:Pressure C code

2- Main Algo

```c
#include "mainAlgo.h"
void (*pointerToState_MALGO)(void);
suint8 global_suint8RecievedPressureValue;
void mainAlgoInit(void)
{
    // assign 1st state
    pointerToState_MALGO = STATE(mainAlgo_pressureCompartor);

}
STATE_DEFINE(mainAlgo_pressureCompartor)
{
    // apply compartor on pressure value
    if (global_suint8RecievedPressureValue >= THERSHOLD_PRESSURE)
    {
        // send pressure detected to alarm manger
        HighPressure_COM(HIGH_PRESSURE_DETECTED);
        // store this value in memory OPTIONAL

    }
    else
    {
        // send pressure value again
        HighPressure_COM(NO_PRESSURE_DETECTED);
    }


}
void PressureValue_COM(int pVal)
{
    global_suint8RecievedPressureValue = pVal;
}
```

Figure 14:MainAlgo C code

3- Alarm Manger

```c
#include "alarmManger.h"
void (*pointerToState_ALMANG)(void);
suint8 isPressureDetected;
void alarmMangerInit(void)
{
    // Assign First State 2 pointer
    pointerToState_ALMANG = STATE(alarmManger_sendAlarmOFF);
}
STATE_DEFINE(alarmManger_sendAlarmOFF)
{
    // send signal to stop alarm
    Alarm_COM(ALARM_STOP);
    // check pressure detect signal
    if (isPressureDetected)
    {

        // pointerToState_ALMANG = STATE(alarmManger_sendAlarmON);
        //call
        STATE_CALL(alarmManger_sendAlarmON);
    }
    else
    {
        pointerToState_ALMANG = STATE(alarmManger_sendAlarmOFF);

    }
}
STATE_DEFINE(alarmManger_sendAlarmON)
{
    // SET TIMER FOR 60 SEC
    // SEND SIGNAL TO PLAY ALARM
    Alarm_COM(ALARM_PLAY);
    // ENTER WAITING STATE
    pointerToState_ALMANG = STATE(alarmManger_waiting);

}
STATE_DEFINE(alarmManger_waiting)
{
    // wait for 60 sec timer then goto ALARM OFF STATE
    // emulate delay func
    // WAIT FOR TIMER EXPIRE
    volatile int i = 6000;
    Delay(i);
    // GOTO ALARM OFF STATE
    pointerToState_ALMANG = STATE(alarmManger_sendAlarmOFF);
}
void HighPressure_COM(int isExceddThershold)
{
    isPressureDetected = isExceddThershold;

}
```

Figure 15:Alarm Manger C code

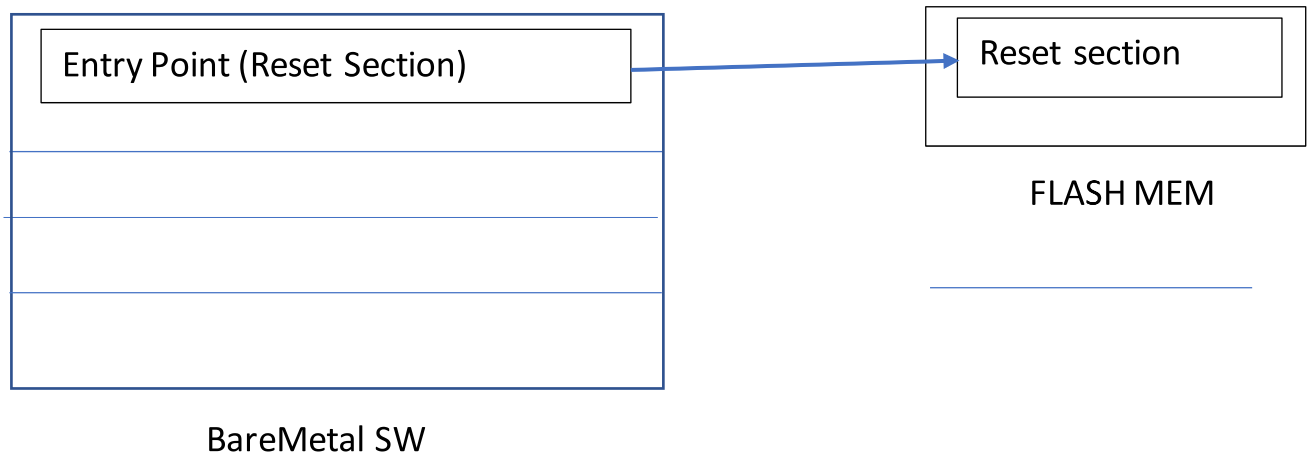
4- Alarm

```
#include "alarm.h"
#include "driver.h"

suint8 global_suint8AlarmState = 0;
void (*pointerToState_ALARM)(void);

void alarmInit(void)
{
    // INIT GPIO DRIVER
    GPIO_INITIALIZATION();
    // INIT POINTER FIRST STATE
    pointerToState_ALARM = STATE(alarmState_compartor);
}
STATE_DEFINE(alarmState_compartor)
{
    if (global_suint8AlarmState == ALARM_PLAY)
    {
        // call state alarm on
        STATE_CALL(alarmState_alarmON);
    }
    else
    {
        // call state alarm off
        STATE_CALL(alarmState_alarmOFF);
    }
}
STATE_DEFINE(alarmState_alarmOFF)
{
    // STOP ALARM FROM DIO
    Set_Alarm_actuator(ALARM_STOP);

    pointerToState_ALARM = STATE(alarmState_compartor);
}
STATE_DEFINE(alarmState_alarmON)
{
    // PLAY ALARM FROM DIO
    Set_Alarm_actuator(ALARM_PLAY);
    pointerToState_ALARM = STATE(alarmState_compartor);

}

void Alarm_COM(int alarmState)
{
    global_suint8AlarmState = alarmState;
}
```

Figure 16:Alarm C code

For Source Code

Boot Sequence



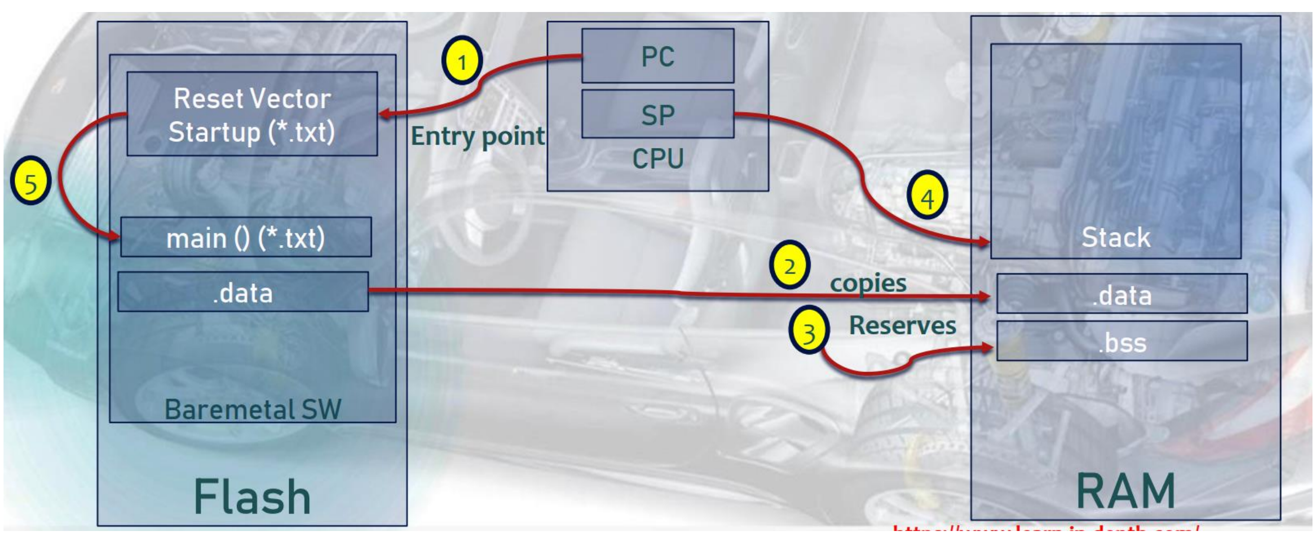


Figure 17:Sequence

Our entry point is reset handler that move .data from FLASH to SRAM and reserve .bss section in SRAM.

Linker script file

Memory Areas and stack size

```
/* _Heap_Size = 0x200; */                              /* required amount of 512B heap  */
_Stack_Size = 0x1000;                          /* required amount of 4kB stack */
ENTRY(resetHandler)                            /* Entry Point */

/* Specify Memory Areas */
MEMORY
{
    FLASH (rx) :    ORIGIN = 0x08000000 , LENGTH = 32K
    SRAM  (rwx) :   ORIGIN = 0x20000000 , LENGTH = 10K
}
```

Figure 18:Memory Areas in Linker Script

Sections

```
/* Define Output Sections */
SECTIONS
{

    /* The program code and other data goes into FLASH */
    .text   :
    {
        . = ALIGN(4);
        _TEXT_S_ = . ;
        *(.vectors*)                              /* .rodata* sections (constants, strings, etc.) */
        *(.text*)                                 /* .text* sections (code) */
        _E_TEXT_SEC = . ;
        . = ALIGN(4);
    }>FLASH

    /* Initialized data sections goes into RAM, load LMA copy after code */
    .data :
    {
        . = ALIGN(4);
        _S_DATA_SEC = . ;
        *(.data*)
        *(.rodata*)
        _E_DATA_SEC = . ;
        . = ALIGN(4);
    }> SRAM AT>FLASH

    /* Uninitialized data section */
    .bss :
    {
        . = ALIGN(4);
        _S_BSS_SEC = . ;
        *(.bss)
        _E_BSS_SEC = .;
        . = ALIGN(4);
    }>SRAM
    /* size of Heap and Stack */
    /*   . = . + _Heap_Size;   */
    . = . + _Stack_Size;
    _STACK_TOP = . ;
    . = ALIGN(4);

}
```

Figure 19:Memory Section in Linker script

Startup

resetHandler()

```c
void resetHandler(void)
{
    /*                                   .data FORM ROM TO RAM                                    */
    uint16 DATA_SIZE =  (uint16 *)&_E_DATA_SEC - (uint16 *)&_S_DATA_SEC ;  // uint16 because sram rang 0x0x2000 0000:0x2000 1000 so we use the LS part
    uint16 * P_SRC = (uint16 *)&_E_TEXT_SEC ;
    uint16 * P_DST = (uint16 *)&_S_DATA_SEC ;

    for(uint16 i = 0 ; i < DATA_SIZE ; i++)
    {
        *((uint16 *)(P_DST++)) = *((uint16 *)(P_SRC++));
    }
    /*                                   bss SECTION                               */
    uint16 BSS_SIZE =   (uint16 *)&_E_DATA_SEC - (uint16 *)&_S_DATA_SEC ;
    P_DST = (uint16 *)&_S_BSS_SEC ;
    for(uint16 i = 0 ; i < BSS_SIZE ; i++)
    {
        *((uint16 *)(P_DST++)) = ((uint16)0) ;
    }
    /*                                   JUMMP MAIN                                    */
    main();
}
```

Figure 20:Reset Handler in startup code

Vector section

```c
uint32 vectors[] __attribute__((section(".vectors"))) = /* add this array in section vectors that hold 1st address in flash */
{
    (uint32) &_STACK_TOP ,
    (uint32) &resetHandler,                /* 1 Reset */
    (uint32) &NMI_Handler,                 /* 2 NMI */
    (uint32) &HardFault_Handler,           /* 3 Hard Fault */
    (uint32) &MMFault_Handler,             /* 4 MM Fault */
    (uint32) &BusFault_Handler,            /* 5 Bus Fault */
    (uint32) &UsageFault_Handler,          /* 6 Usage Fault */
    (uint32) &RESEVERD_Handler,            /* 7 RESEVERD */
    (uint32) &RESEVERD_Handler,            /* 8 RESEVERD */
    (uint32) &RESEVERD_Handler,            /* 9 RESEVERD */
    (uint32) &RESEVERD_Handler,            /* 10 RESEVERD */
    (uint32) &SVcall_Handler,              /* 11 SV call */
    (uint32) &DebugReserved_Handler,       /* 12 Debug reserved */
    (uint32) &RESEVERD_Handler,            /* 13 RESEVERD */
    (uint32) &PendSV_Handler,              /* 14 PendSV */
    (uint32) &SysTick_Handler,             /* 15 SysTick */
    (uint32) &IRQ0_Handler                 /* 16 IRQ0 ...*/
};
```

Figure 21:Vector Section in startup code

Vector table functions

```c
extern uint32 _STACK_TOP ;
extern void main(void);
void resetHandler(void);
void defaultHandler(void);
void NMI_Handler(void)__attribute__((weak,alias("defaultHandler"))) ;
void HardFault_Handler(void)__attribute__((weak,alias("defaultHandler"))) ;
void MMFault_Handler(void)__attribute__((weak,alias("defaultHandler"))) ;
void BusFault_Handler(void)__attribute__((weak,alias("defaultHandler"))) ;
void UsageFault_Handler(void)__attribute__((weak,alias("defaultHandler"))) ;
void RESEVERD_Handler(void)__attribute__((weak,alias("defaultHandler"))) ;
void SVcall_Handler(void)__attribute__((weak,alias("defaultHandler"))) ;
void DebugReserved_Handler(void)__attribute__((weak,alias("defaultHandler"))) ;
void PendSV_Handler(void)__attribute__((weak,alias("defaultHandler"))) ;
void SysTick_Handler(void)__attribute__((weak,alias("defaultHandler"))) ;
void IRQ0_Handler(void)__attribute__((weak,alias("defaultHandler"))) ;
```

Figure 22: vector functions

Weak --> to be overwritten, alias to make declaration emitted to be alias for another function.

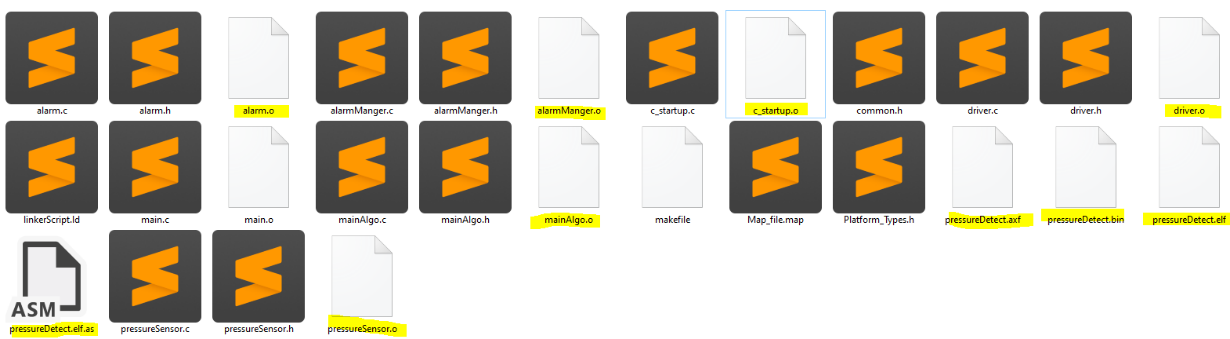
Default handler

```c
void defaultHandler(void)
{
    resetHandler();
}
```

Figure 23:Default Handler

Make

```
Abnaby@DESKTOP-159V4HP MINGW64 /e/COURSES/Learn-in-Depth/Repo/firstTerm/ProjectO
ne/FIRST_TERM_project1/Src (main)
$ make
----------------------Start Building Process----------------

Linking Statge Successfully Generated ELF Img and AXF Img for Target -mcpu=corte
x-m3.
Binary File Successfully Generated...

---------------------Build is Done.-------------------------

Abnaby@DESKTOP-159V4HP MINGW64 /e/COURSES/Learn-in-Depth/Repo/firstTerm/ProjectO
ne/FIRST_TERM_project1/Src (main)
$ |
```



Map File

- Memory Configuration

```
/* Specify Memory Areas */
MEMORY
{
    FLASH (rx) :    ORIGIN = 0x08000000 , LENGTH = 32K
    SRAM  (rwx) :   ORIGIN = 0x20000000 , LENGTH = 10K
}
```

In LinkerScript.ld

E:\COURSES\Learn-in-Depth\Repo\firstTerm\ProjectOne\FIRST_TERM_project1\Src\Map_file.map - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

Map_file.map

```
Name             Origin             Length             Attributes
FLASH            0x08000000         0x00008000         xr
SRAM             0x20000000         0x00002800         xrw
*default*        0x00000000         0xffffffff
```

In map file

Figure 24: Memory configuration in Linkerscript and map file.

- Memory map

```
*(.vectors*)
.vectors          0x08000000          0x44 c_startup.o
                  0x08000000                   vectors
```

Figure 25:Vector table position in map file

Start address of flash memory

Disassemble elf image

```
$ arm-none-eabi-objdump.exe pressureDetect.elf -D

pressureDetect.elf:     file format elf32-littlearm


Disassembly of section .text:

08000000 <vectors>:
 8000000:       2000102c        andcs   r1, r0, ip, lsr #32
 8000004:       08000191        stmdaeq r0, {r0, r4, r7, r8}
 8000008:       08000185        stmdaeq r0, {r0, r2, r7, r8}
 800000c:       08000185        stmdaeq r0, {r0, r2, r7, r8}
 8000010:       08000185        stmdaeq r0, {r0, r2, r7, r8}
 8000014:       08000185        stmdaeq r0, {r0, r2, r7, r8}
 8000018:       08000185        stmdaeq r0, {r0, r2, r7, r8}
 800001c:       08000185        stmdaeq r0, {r0, r2, r7, r8}
 8000020:       08000185        stmdaeq r0, {r0, r2, r7, r8}
 8000024:       08000185        stmdaeq r0, {r0, r2, r7, r8}
 8000028:       08000185        stmdaeq r0, {r0, r2, r7, r8}
 800002c:       08000185        stmdaeq r0, {r0, r2, r7, r8}
 8000030:       08000185        stmdaeq r0, {r0, r2, r7, r8}
 8000034:       08000185        stmdaeq r0, {r0, r2, r7, r8}
 8000038:       08000185        stmdaeq r0, {r0, r2, r7, r8}
 800003c:       08000185        stmdaeq r0, {r0, r2, r7, r8}
 8000040:       08000185        stmdaeq r0, {r0, r2, r7, r8}
```

Figure 26:Disassemble of elf image

Memory dump

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

pressureDetect.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000042c  08000000  08000000  00010000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000000  20000000  0800042c  00020000  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          0000002c  20000000  0800042c  00020000  2**2
                  ALLOC
  3 .debug_info   00001ae4  00000000  00000000  00020000  2**0
                  CONTENTS, READONLY, DEBUGGING
  4 .debug_abbrev 000008b2  00000000  00000000  00021ae4  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    0000062c  00000000  00000000  00022396  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 000000e0  00000000  00000000  000229c2  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_line   00000511  00000000  00000000  00022aa2  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .debug_str    000008ef  00000000  00000000  00022fb3  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      0000007e  00000000  00000000  000238a2  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  00023920  2**0
                  CONTENTS, READONLY
 11 .debug_frame  00000398  00000000  00000000  00023954  2**2
                  CONTENTS, READONLY, DEBUGGING
```

Debug info

Figure 27:Memory Dump with debug section

```
/* Initialized data sections goes into RAM, load LMA copy after code */
.data :
{
    . = ALIGN(4);
    _S_DATA_SEC = . ;
    *(.data*)
    *(.rodata*)
    _E_DATA_SEC = . ;
    . = ALIGN(4);
}> SRAM AT>FLASH
```

```
MINGW64:/e/COURSES/Learn-in-Depth/Repo/firstTerm/ProjectOne/FIRST_TERM_project1/Src
pressureDetect.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000042c  08000000  08000000  00010000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000000  20000000  0800042c  00020000  2**0
```

Figure 28:position of .data section in flash and sram

Sections without debug

```
$ make dump_elf

pressureDetect.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000042c  08000000  08000000  00010000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000000  20000000  0800042c  00020000  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          0000002c  20000000  0800042c  00020000  2**2
                  ALLOC
  3 .comment      0000007e  00000000  00000000  00020000  2**0
                  CONTENTS, READONLY
  4 .ARM.attributes 00000033  00000000  00000000  0002007e  2**0
                  CONTENTS, READONLY
```

Figure 29:Memory dump without dung sections

Symbols

```
$ arm-none-eabi-nm.exe pressureDetect.elf
20000010 B _E_BSS_SEC
20000000 D _E_DATA_SEC
0800042c T _E_TEXT_SEC
20000000 B _S_BSS_SEC
20000000 D _S_DATA_SEC
00001000 A _Stack_Size
2000102c B _STACK_TOP
08000000 T _TEXT_S_
080000b8 T Alarm_COM
20000010 B alarm_state
08000044 T alarmInit
20000018 B alarmManger_state
080000d4 T alarmMangerInit
08000184 W BusFault_Handler
08000184 W DebugReserved_Handler
08000184 T defaultHandler
08000214 T Delay
08000234 T getPressureVal
20000000 b global_suint8AlarmState
2000000c b Global_suint8PressureValue
20000008 b global_suint8RecievedPressureValue
08000288 T GPIO_INITIALIZATION
08000184 W HardFault_Handler
08000168 T HighPressure_COM
08000184 W IRQ0_Handler
20000004 b isPressureDetected
080002f0 T main
20000021 B mainAlgo_state
0800033c T mainAlgoInit
08000184 W MMFault_Handler
08000184 W NMI_Handler
08000184 W PendSV_Handler
20000014 B pointerToState_ALARM
2000001c B pointerToState_ALMANG
20000024 B pointerToState_MALGO
20000028 B pointerToState_PS
20000020 B pressureSensor_state
08000398 T pressureSensorInit
0800037c T PressureValue_COM
08000190 T resetHandler
08000184 W RESEVERD_Handler
0800024c T Set_Alarm_actuator
080002d8 T setup
080000f0 T ST_alarmManger_sendAlarmOFF
08000120 T ST_alarmManger_sendAlarmON
0800013c T ST_alarmManger_waiting
08000080 T ST_alarmState_alarmOFF
0800009c T ST_alarmState_alarmON
08000060 T ST_alarmState_compartor
08000358 T ST_mainAlgo_pressureCompartor
080003b4 T ST_PS_readingSensor
080003dc T ST_PS_SendReading
08000400 T ST_PS_waiting
08000184 W SVcall_Handler
08000184 W SysTick_Handler
08000184 W UsageFault_Handler
08000000 T vectors
```

Figure 30:Symbols of ELF image

All symbols successfully resolved

For each symbol check link

ELF image details

```
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           ARM
  Version:                           0x1
  Entry point address:               0x8000191
  Start of program headers:          52 (bytes into file)
  Start of section headers:          134272 (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:         9
  Section header string table index: 8
```

Figure 31:ELF image details

```
Attribute Section: aeabi
File Attributes
  Tag_CPU_name: "Cortex-M3"
  Tag_CPU_arch: v7
  Tag_CPU_arch_profile: Microcontroller
  Tag_THUMB_ISA_use: Thumb-2
  Tag_ABI_PCS_wchar_t: 4
  Tag_ABI_FP_denormal: Needed
  Tag_ABI_FP_exceptions: Needed
  Tag_ABI_FP_number_model: IEEE 754
  Tag_ABI_align_needed: 8-byte
  Tag_ABI_align_preserved: 8-byte, except leaf SP
  Tag_ABI_enum_size: small
  Tag_ABI_optimization_goals: Aggressive Debug
  Tag_CPU_unaligned_access: v6
```

Figure 32:ELF image attributes

Hardware Simulation

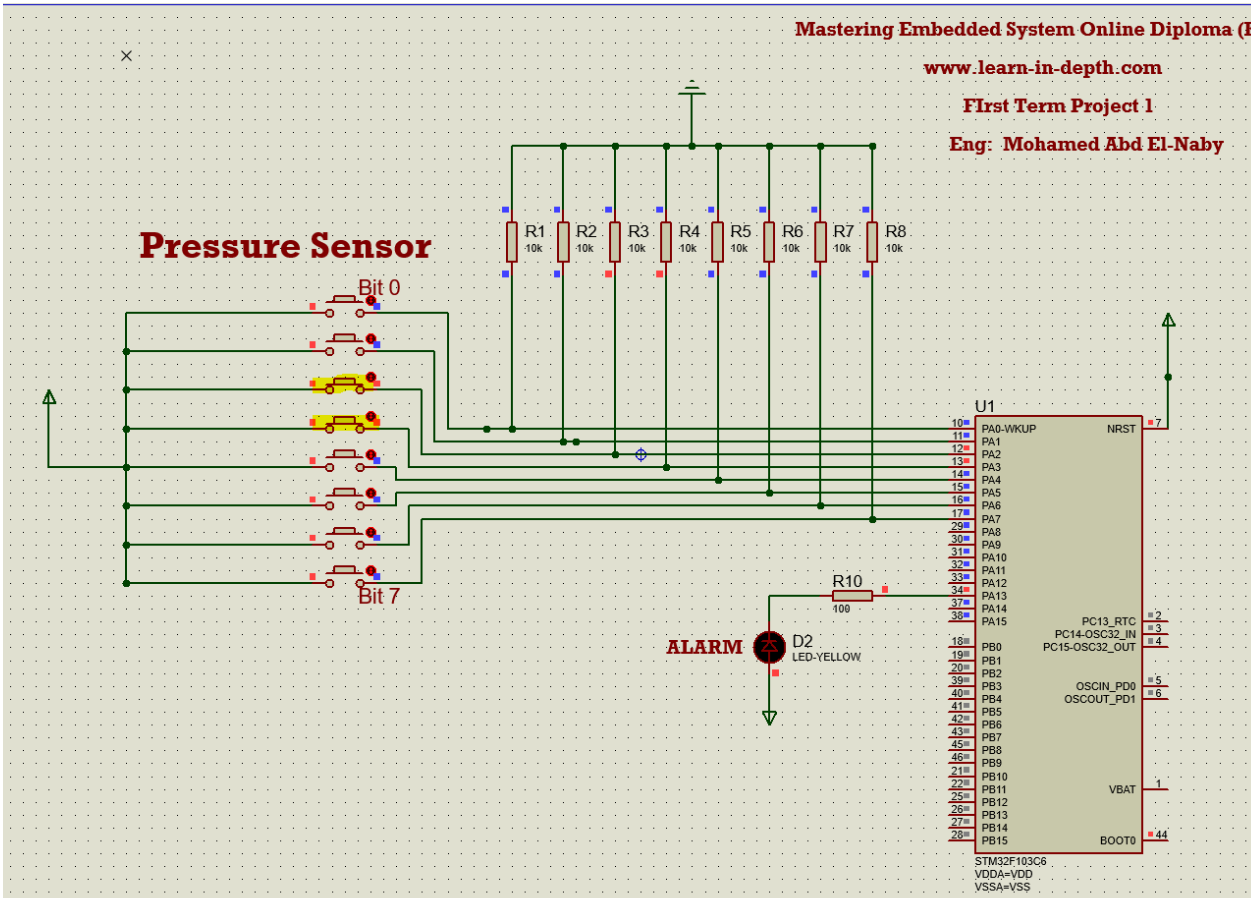


Figure 33:simulation test case 1

I/P Pressure = 12 bar < 20 BAR

O/P Alarm = OFF

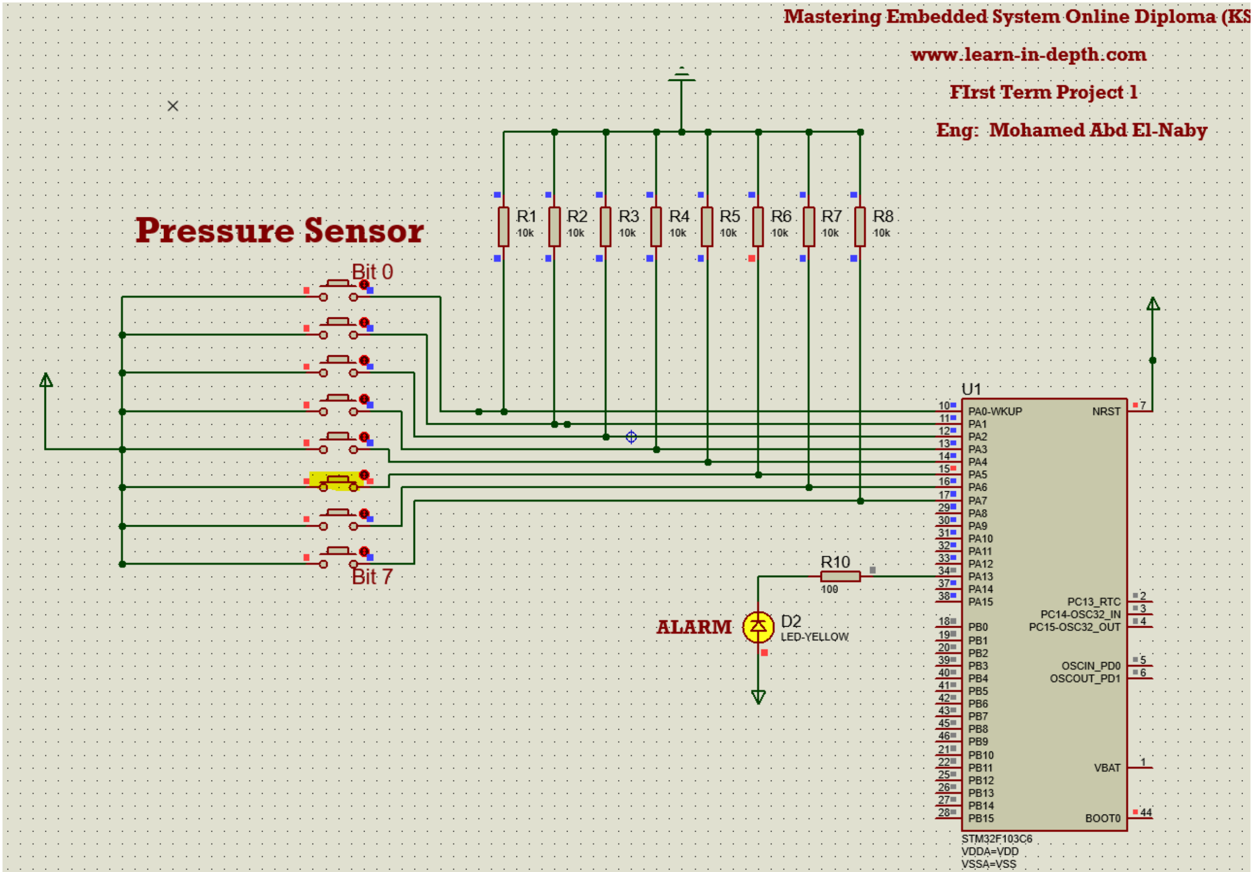


Figure 34:simulation test case 2

I/P Pressure = 32 bar > 20 BAR

O/P Alarm = ON FOR 60 SEC