Learn IN Depth

Be Professional In Embedded System

Pressure Detection Project Report

Name	Basem Said
Unit	First Term Projects
Assignment	1
Topic	First Term Project 1

11/11/2023

Contents: -

- Case Study:

A pressure Detection informs in room with an alarm when the pressure exceeds 20 bars in the room.

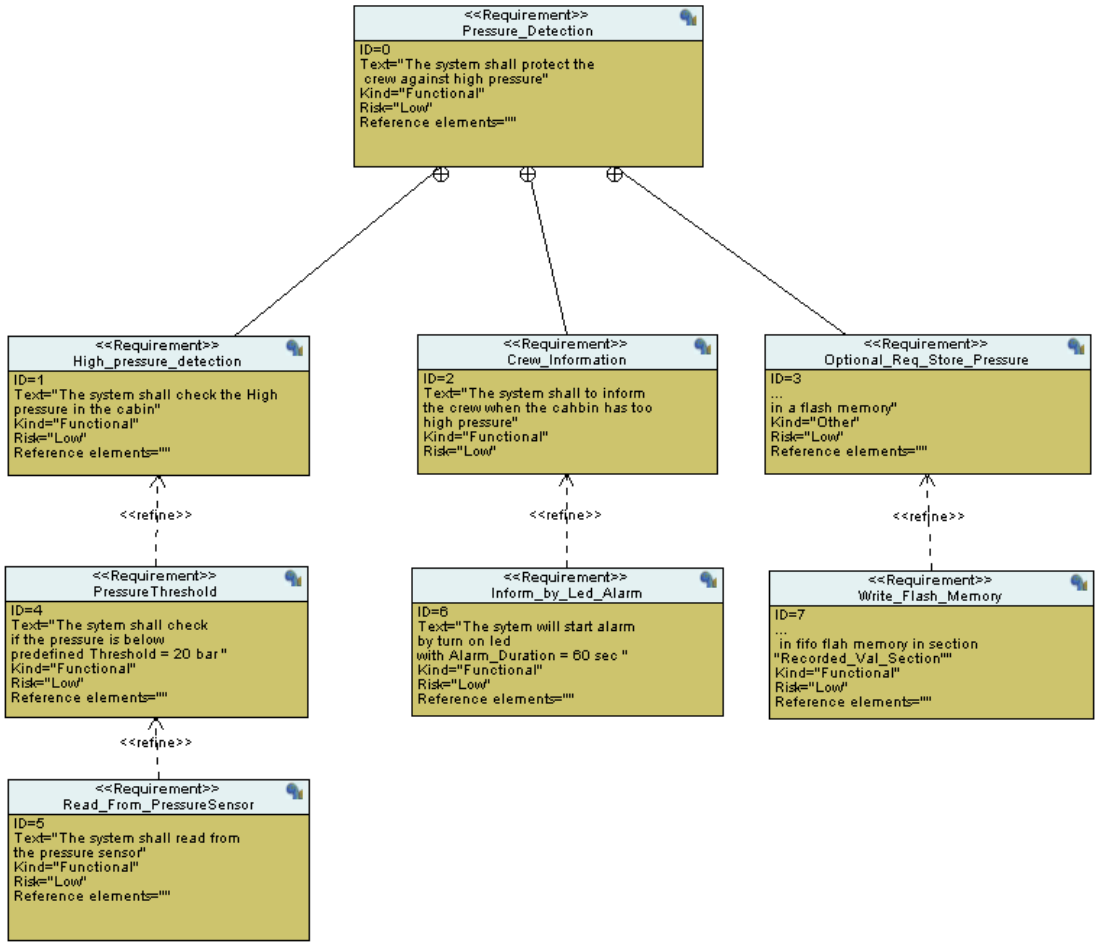
The alarm duration equals 60 seconds.

Keep track of the measured values.

- Methodology:

Since the system has multiple modules that are no easy to integrate, the system will use a testing-based model like v-model. Every phase in this project will be tested and especially the implementation phase. Each software module will be implemented and unit-tested separately then integrated and integration testing will be performed.

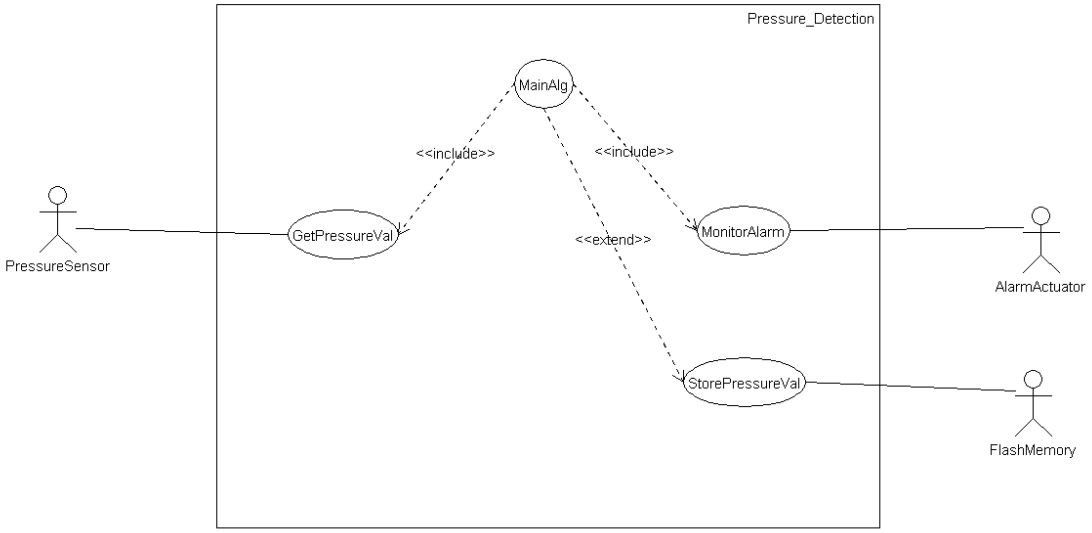
- Requirement Diagram:



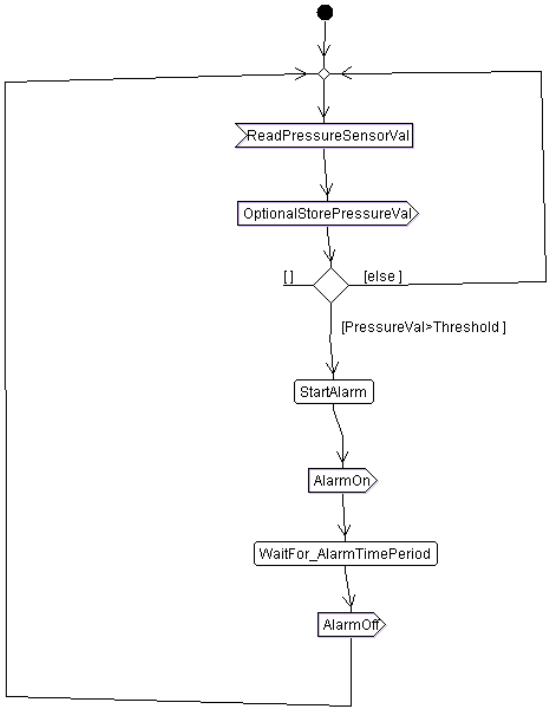
- (HW/SW Partitioning):

For the hardware, we have STM32 microcontroller with a cortex-m3 processor that will be enough for this application.

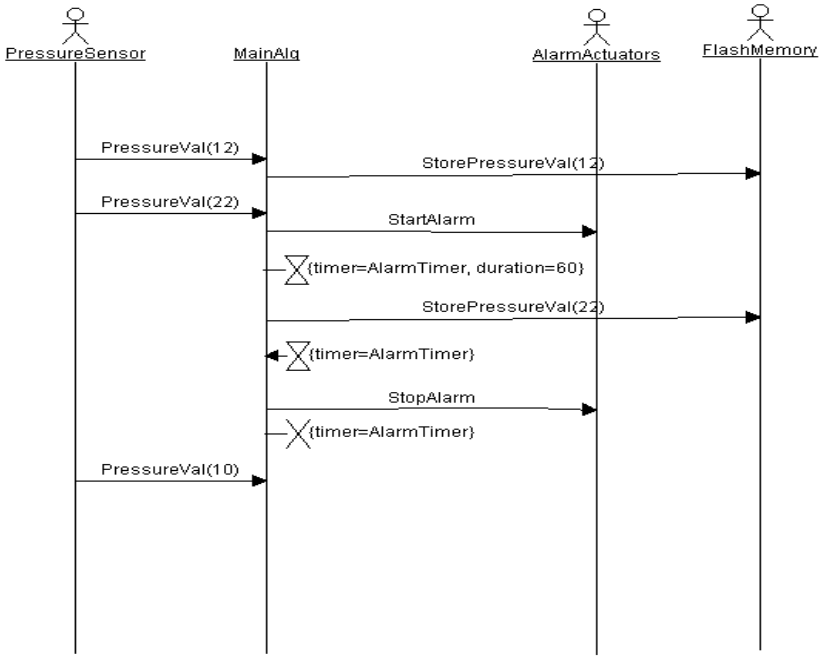
- System Analysis: Use Case Diagram: -



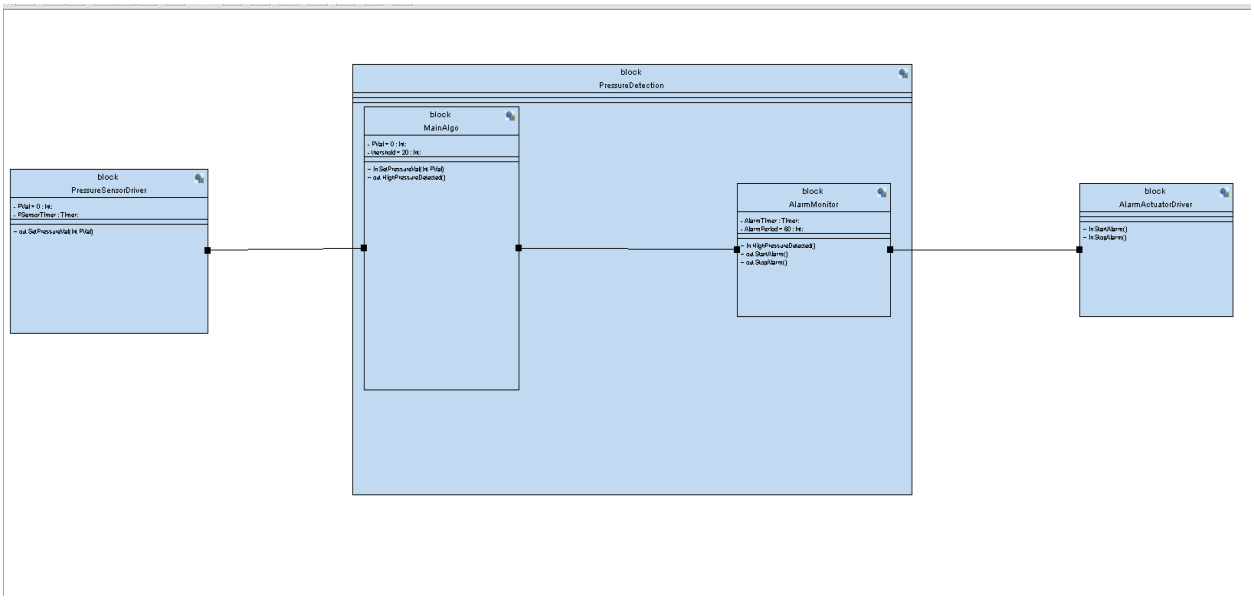
- System Analysis: Activity Diagram:



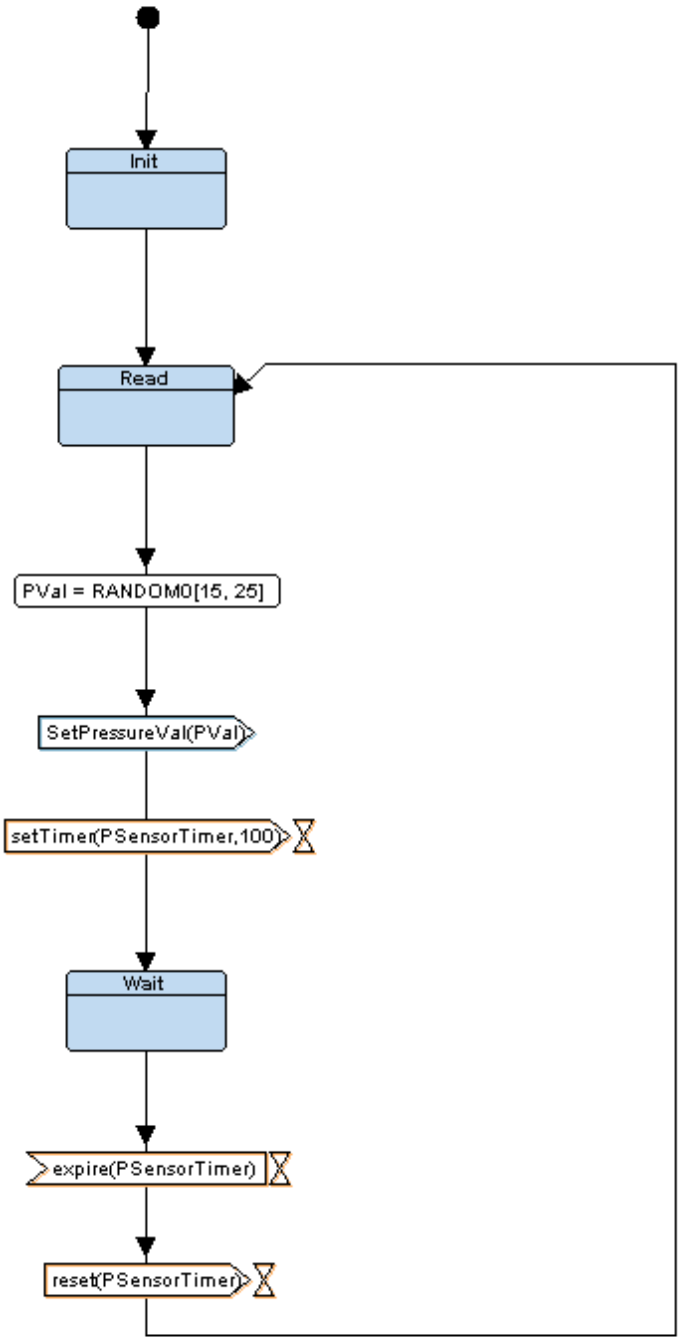
- System Analysis: Sequence Diagram:



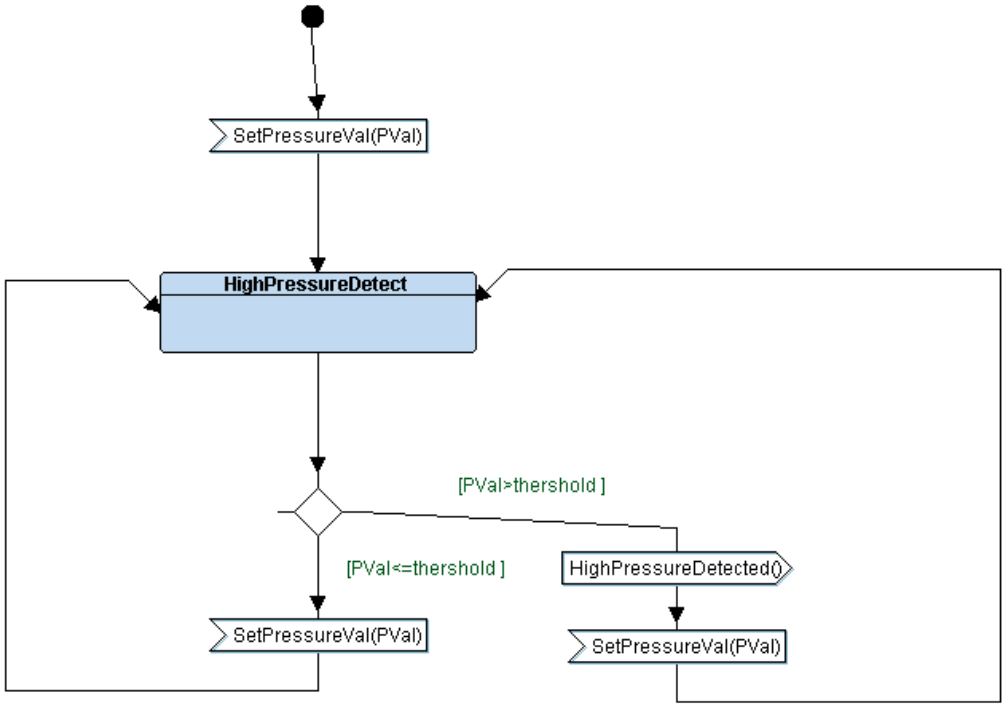
- System Design:



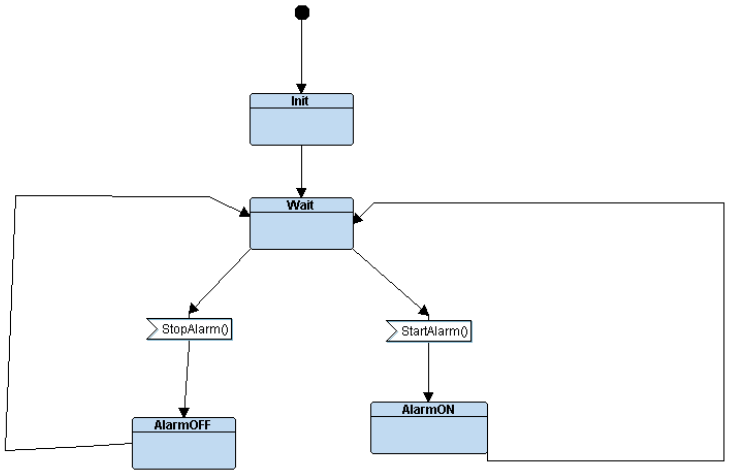
1- Pressure Sensor State Diagram



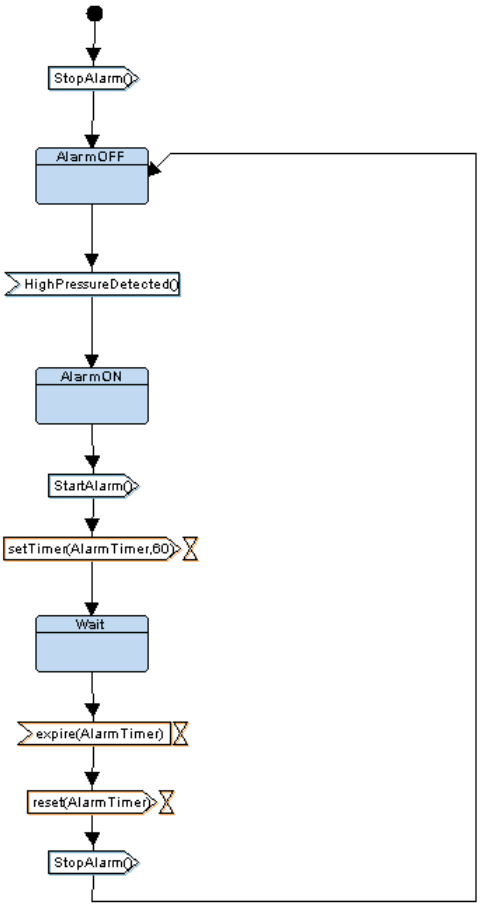
2- Main Program State Diagram:



3- Alarm Actuator State Diagram:



4- Alarm Monitor State Diagram:



- Main.c

```c
/*
 * main.c
 *
 *  Created on: 27/10/2023
 *      Author: Basem
 */

#include "Platform_Types.h"
#include "Util.h"

#include "GPIO.h"
#include "Pressure_Sensor.h"
#include "Alarm_Actuator.h"
#include "Alarm_Monitor.h"
#include "Main_Program.h"

void (*PS_State)() = STATE(PS_Init) ;
void (*Alarm_Act_State)() = STATE(Alarm_Act_Init) ;
void (*AM_State)() = STATE(AM_Alarm_OFF) ;
void (*MP_State)() = STATE(MP_High_Pressure) ;

int main (void)
{
    // Hardware Initialization
    GPIO_Init() ;

    // Infinite Loop
    while (1)
    {
        PS_State() ;
        Alarm_Act_State() ;
        AM_State() ;
        MP_State() ;
    }

    return 0 ;
}
```

- State.h

```c
/*
 * state.h
 *
 *  Created on: 27/10/2023
 *      Author: Basem
 */

#ifndef STATE_H_
#define STATE_H_

#include "GPIO.h"

// Automatic State Function Generated
#define STATE_Define(_statFUN_) void ST_##_statFUN_()
#define STATE(_statFUN_)  ST_##_statFUN_


//States Connection

// Pressure Sensor =====> Main Program
uint32_t PS_Get_Pressure_Value (void) ;

//Alaram Actuator =====> Alaram Monitor
void AA_Start_Alarm (void) ;
void AA_Stop_Alarm (void) ;

//Main Program =====> Alaram Monitor
uint32_t MP_High_Pressure_Detection (void) ;

#endif /* STATE_H_ */
```

- GPIO

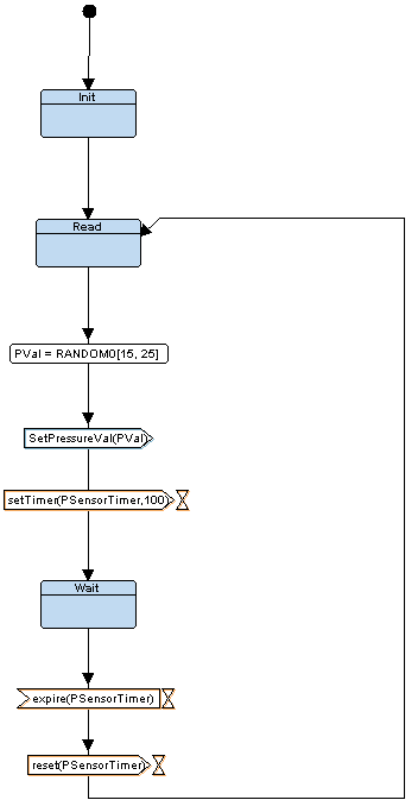
```c
/*
 * GPIO.c
 *
 * Created on: 27/10/2023
 *      Author: Basem
 */

#include "GPIO.h"

void GPIO_Delay(uint32_t nCount)
{
    for(; nCount != 0; nCount--);
}

uint32_t GPIO_GetPressureVal(void){
    return (GPIOA_IDR & 0xFF);
}

void GPIO_Set_Alarm_Actuator(uint32_t i){
    if (i == 1){
        SET_BIT(GPIOA_ODR,13);
    }
    else if (i == 0){
        RESET_BIT(GPIOA_ODR,13);
    }
}

void GPIO_Init (void){
    SET_BIT(APB2ENR, 2);
    GPIOA_CRL &= 0xFF0FFFFF;
    GPIOA_CRL |= 0x00000000;
    GPIOA_CRH &= 0xFF0FFFFF;
    GPIOA_CRH |= 0x22222222;
}
```

```c
/*
 * GPIO.h
 *
 * Created on: 27/10/2023
 *      Author: Basem
 */

#ifndef GPIO_H_
#define GPIO_H_

#include "Platform_Types.h"
#include "Util.h"

#define GPIO_PORTA 0x40010800
#define BASE_RCC   0x40021000

#define APB2ENR   *(volatile uint32_t *)(BASE_RCC + 0x18)

#define GPIOA_CRL *(volatile uint32_t *)(GPIO_PORTA + 0x00)
#define GPIOA_CRH *(volatile uint32_t *)(GPIO_PORTA + 0X04)
#define GPIOA_IDR *(volatile uint32_t *)(GPIO_PORTA + 0x08)
#define GPIOA_ODR *(volatile uint32_t *)(GPIO_PORTA + 0x0C)

void GPIO_Init (void);
void GPIO_Delay(uint32_t nCount);

uint32_t GPIO_GetPressureVal(void);
void GPIO_Set_Alarm_Actuator(uint32_t i);


#endif /* GPIO_H_*/
```

- Pressure Sensor

```c
/*
 * Pressure_Sensor.c
 *
 *  Created on: 27/10/2023
 *      Author: Basem
 */

#include "Pressure_Sensor.h"

// Declare Status
enum{

     PS_Init,
     PS_Reading,
     PS_Waiting
}E_PS_Status;

//Declare Variable to store Data from Pressure_Sensor
static uint32_t PS_Pressure_Value ;

STATE_Define(PS_Init)
{
    // Initialize Pressure Sensor
    // Call Pressure Sensor Functions
    // State Action
    E_PS_Status = PS_Init ;

    // Check Event and Update Status
    PS_State = STATE(PS_Reading) ;
}

STATE_Define(PS_Reading)
{
    // State Action
    E_PS_Status = PS_Reading ;

    // Read From Pressure Sensor
    PS_Pressure_Value = GPIO_GetPressureVal() ;

    // Check Event and Update Status
    PS_State = STATE(PS_Waiting) ;
}

STATE_Define(PS_Waiting)
{
    // State Action
    E_PS_Status = PS_Waiting ;

    // Wait to Get Data From Sensor
    GPIO_Delay(1000) ;

    // Check Event and Update Status
    PS_State = STATE(PS_Reading) ;
}

//Interface and Set Pressure Value with Main Program
uint32_t PS_Get_Pressure_Value(void)
{
    return PS_Pressure_Value ;
}
```

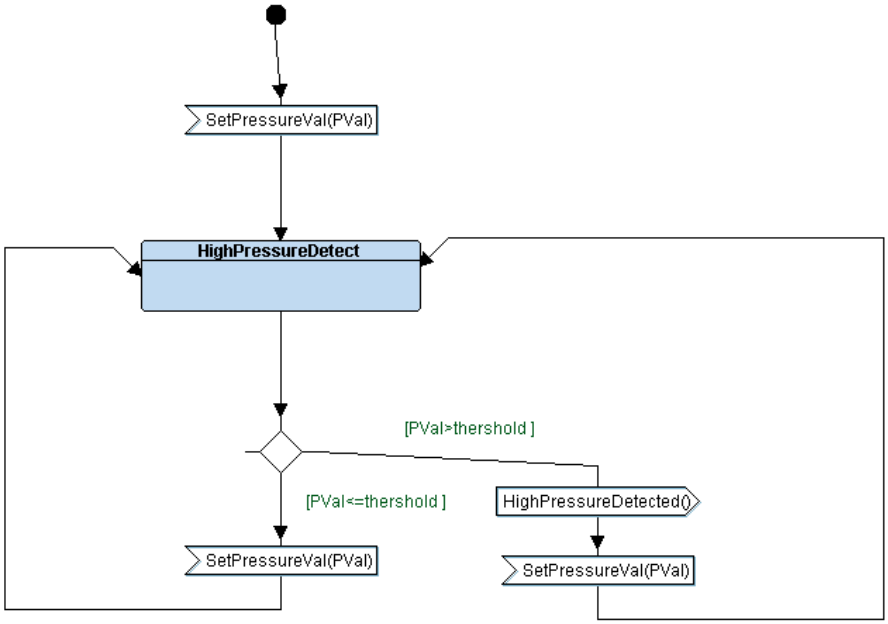
```c
/*
 * Pressure_Sensor.h
 *
 *  Created on: 27/10/2023
 *      Author: Basem
 */

#ifndef PRESSURE_SENSOR_H_
#define PRESSURE_SENSOR_H_

#include "state.h"

// Declaration State Function of Pressure_Sensor
STATE_Define(PS_Init) ;
STATE_Define(PS_Reading) ;
STATE_Define(PS_Waiting) ;

// State Pointer to function
extern void (*PS_State)() ;



#endif /* PRESSURE_SENSOR_H_ */
```



- Main Program

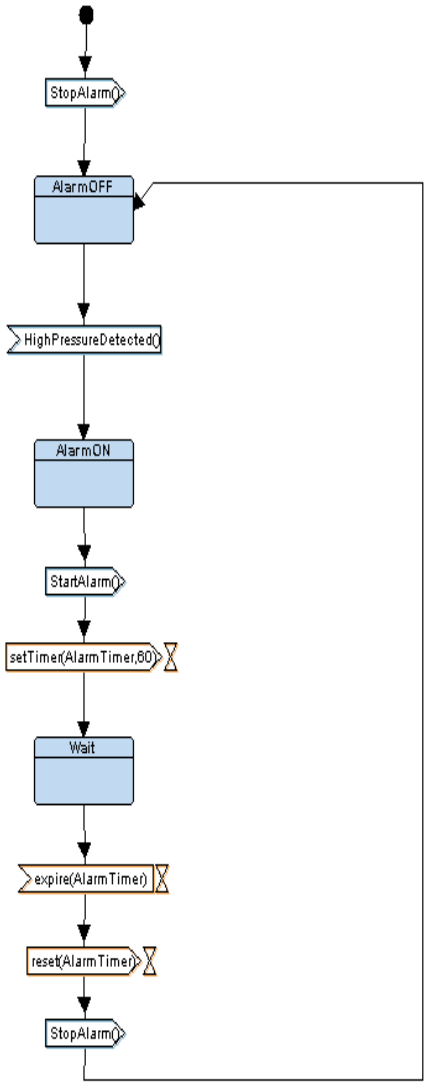
```c
/*
 * Main_Program.c
 *
 *  Created on: 27/10/2023
 *      Author: Basem
 */

#include "Main_Program.h"

uint32_t MP_High_Pressure_Detection(void);

// Define Status
enum {
        MP_High_Pressure
 }E_MP_Status;

// Declare Variable of Pressure Value
static uint32_t MP_Pressure_Value ;
// Define Thershold Variable
static uint32_t MP_Pressure_Thershold = 20 ;

STATE_Define(MP_High_Pressure)
{
    // State Action
    E_MP_Status = MP_High_Pressure ;

    // Read Pressure Value From Pressure Sensor
    MP_Pressure_Value = PS_Get_Pressure_Value() ;

    // Check Event and Update State
    MP_State = STATE(MP_High_Pressure) ;

}

// Main Program =====> Alarm Mointor
uint32_t MP_High_Pressure_Detection(void)
{
    return (MP_Pressure_Value > MP_Pressure_Thershold) ;
}
```

```c
/*
 * Main_Program.h
 *
 *  Created on: 27/10/2023
 *      Author: Basem
 */

#ifndef MAIN_PROGRAM_H_
#define MAIN_PROGRAM_H_

#include "state.h"

// Declare State Function
STATE_Define(MP_High_Pressure) ;

// State Pointer to Function
extern void (*MP_State)() ;

#endif /* MAIN_PROGRAM_H_ */
```



- Alarm Monitor

```c
/*
 * Alarm_Monitor.c
 *
 *  Created on: 27/10/2023
 *      Author: Basem
 */

#include "Alarm_Monitor.h"

// Define Status
enum {
    AM_Alarm_OFF,
    AM_Alarm_ON,
    AM_Waiting
}E_AM_Status ;

STATE_Define(AM_Alarm_OFF)
{
    // State Action
    E_AM_Status = AM_Alarm_OFF ;

    // Stop Alarm Actuator
    AA_Stop_Alarm();

    // Check Event and Update State
    if (MP_High_Pressure_Detection() == TRUE)
    {
        AM_State = STATE(AM_Alarm_ON) ;
    }
}

STATE_Define(AM_Alarm_ON)
{
    // State Action
    E_AM_Status = AM_Alarm_ON ;

    // Start Alarm Actuator
    AA_Start_Alarm() ;

    // Check Event and Update State
        AM_State = STATE(AM_Waiting);
}

STATE_Define(AM_Waiting)
{
    // State Action
    E_AM_Status = AM_Waiting ;

    // Delay
    GPIO_Delay(500) ;

    // Check Event and Update State
        AM_State = STATE(AM_Alarm_OFF);
}
```

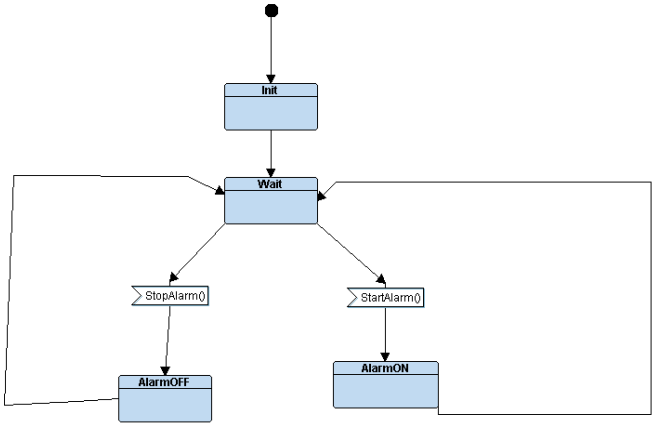



```c
/*
 * Alarm_Monitor.h
 *
 *  Created on: 27/10/2023
 *      Author: Basem
 */

#ifndef ALARM_MONITOR_H_
#define ALARM_MONITOR_H_

#include "state.h"

// Declare State Function
STATE_Define(AM_Alarm_OFF) ;
STATE_Define(AM_Alarm_ON) ;
STATE_Define(AM_Waiting) ;

// State Pointer to Function
extern void (*AM_State)() ;

#endif /* ALARM_MONITOR_H_ */
```

- Alarm Actuator

```c
/*
 * Alarm_Actuator.c
 *
 *  Created on: 27/10/2023
 *      Author: Basem
 */

#include "Alarm_Actuator.h"

// Define Status
enum {
        Alarm_Act_Init,
        Alarm_Act_Waiting,
        Alarm_Act_ON,
        Alarm_Act_OFF
  }E_Alarm_Act_Status ;

STATE_Define(Alarm_Act_Init)
{
    // Initialize The Alarm Actuator
    // Call The Alarm Actuator Driver Function
    // State Action
    E_Alarm_Act_Status = Alarm_Act_Init ;

    // Check Event and Update State
    Alarm_Act_State = STATE(Alarm_Act_Waiting) ;
}

STATE_Define(Alarm_Act_Waiting)
{
    // State Action
    E_Alarm_Act_Status = Alarm_Act_Waiting ;
}

STATE_Define (Alarm_Act_ON)
{
    // State Action
    E_Alarm_Act_Status = Alarm_Act_ON ;

    // Start Alarm Actuator
    GPIO_Set_Alarm_Actuator(TRUE) ;

    // Check Event and Update State
    Alarm_Act_State = STATE(Alarm_Act_Waiting) ;
}

STATE_Define (Alarm_Act_OFF)
{
    // State Action
    E_Alarm_Act_Status = Alarm_Act_OFF ;

    // Start Alarm Actuator
    GPIO_Set_Alarm_Actuator(FALSE) ;

    // Check Event and Update State
    Alarm_Act_State = STATE(Alarm_Act_Waiting) ;
}

void AA_Start_Alarm(void)
{
    // Update State
    Alarm_Act_State = STATE(Alarm_Act_ON) ;
}

void AA_Stop_Alarm(void)
{
    // Update State
    Alarm_Act_State = STATE(Alarm_Act_OFF) ;
}
```

```c
/*
 * Alarm_Actuator.h
 *
 *  Created on: 27/10/2023
 *      Author: Basem
 */

#ifndef ALARM_ACTUATOR_H_
#define ALARM_ACTUATOR_H_

#include "state.h"

// Declare State Fuctions
STATE_Define(Alarm_Act_Init) ;
STATE_Define(Alarm_Act_Waiting) ;
STATE_Define(Alarm_Act_ON) ;
STATE_Define(Alarm_Act_OFF) ;

// State Pointer to Function
extern void (*Alarm_Act_State)() ;

#endif /* ALARM_ACTUATOR_H_ */
```



- Startup.c

```c
/*
startup.c
Eng.Basem
*/

#include<stdint.h>

extern int main(void);

void Reset_Handler();

void Default_Handler()
{
    Reset_Handler ();
}

void NMI_Handler() __attribute__((weak, alias("Default_Handler")));;
void H_Fault_Handler()  __attribute__((weak, alias("Default_Handler")));;
void MM_Fault_Handler()  __attribute__((weak, alias("Default_Handler")));;
void Bus_Fault()  __attribute__((weak, alias("Default_Handler")));;
void Usage_Fault_Handler()  __attribute__((weak, alias("Default_Handler")));;

unsigned int _stack_top ;

//( __attribute__) to move vectors from .data to .text in flash.
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

};

```

```c
extern unsigned int _E_text ;
extern unsigned int _S_DATA ;
extern unsigned int _E_DATA ;
extern unsigned int _S_bss ;
extern unsigned int _E_bss ;

void Reset_Handler()
{
    // Copy data section from Flash to RAM
    unsigned int DATA_Size = (unsigned char*)&_E_DATA - (unsigned char*)&_S_DATA ;        /* We handling with address so we add & before it
                                                                                              bc its variable not symbol and Casting into char to copy byte by byte */
    unsigned char* P_src =  (unsigned char*)&_E_text ;                                    /* Pointer point to source which is started copy from this address */
    unsigned char* P_dst =  (unsigned char*)&_S_DATA ;                                    /* Pointer point to distenation which is ended copy in this address */

    for(int i=0 ; i < DATA_Size ; i++ )
    {
        *((unsigned char*) P_dst++) = *((unsigned char*)P_src++) ;                        /* Copy Data from value of pointer source to value of pointer distenation */
    }

    // init .bss section in SRAM = 0
    unsigned int Bss_Size = (unsigned char*)&_E_bss - (unsigned char*)&_S_bss ;
    P_dst = (unsigned char*)&_S_bss ;

    for(int i=0 ; i < Bss_Size ; i++ )
    {
        *((unsigned char*)P_dst++) = (unsigned char)0 ;                               // initialize Var in bss equal to Zero
    }

    // Jump to main ()

    main () ;
}
```

- Linker_script.ld

```
/* Linker_Script.ld CortexM3
Eng.Basem
*/

MEMORY
{
    flash(RX) : ORIGIN = 0x08000000,  LENGTH = 128k
    sram(WRX) : ORIGIN = 0x20000000,  LENGTH = 20k
}

SECTIONS
{
    .text :
    {                /* Output Section */
            *(.vectors*)        /* take any vectors from files.o */
            *(.text*)           /* take any text from files.o */
            *(.rodata*)         /* take any rodata from files.o */
            _E_text = . ;       /* End of Text Section */
    }> flash

    .data :
    {
    _S_DATA = . ;       /* Start of data Section */
    *(.data*)           /* take any data from files.o */
    . = ALIGN (4) ;     /* Align memory before end data section and start bss section */
    _E_DATA = . ;       /* End of data Section */
    }> sram AT> flash

    .bss :
    {
    _S_bss = . ;      /* Start of bss Section */
    *(.bss*)          /* take any bss from files.o */
    _E_bss = . ;      /* End of bss Section */

    . = ALIGN (4) ;   /* Align memory before end bss section and start stack section */
    . = . + 0x1000;
    _stack_top = . ;

    }> sram
}
```

- Map_File.map

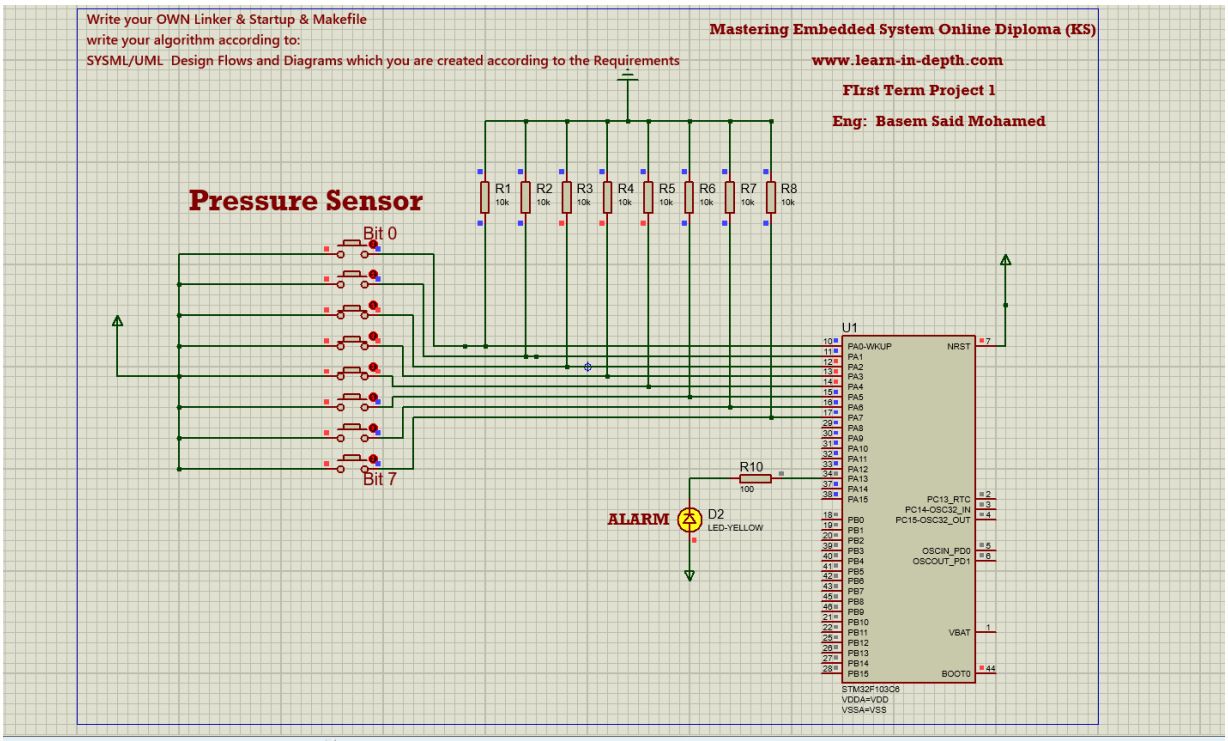
```
Memory Configuration

Name             Origin             Length             Attributes
flash            0x08000000         0x00020000         xr
sram             0x20000000         0x00005000         xrw
*default*        0x00000000         0xffffffff

Linker script and memory map


.text           0x08000000      0x3cc
 *(.vectors*)
 .vectors       0x08000000       0x1c startup.o
                0x08000000                vectors
 *(.text*)
 .text          0x0800001c       0xc4 Alarm_Actuator.o
                0x0800001c                ST_Alarm_Act_Init
                0x08000040                ST_Alarm_Act_Waiting
                0x08000058                ST_Alarm_Act_ON
                0x08000080                ST_Alarm_Act_OFF
                0x080000a8                AA_Start_Alarm
                0x080000c4                AA_Stop_Alarm
 .text          0x080000e0       0x7c Alarm_Monitor.o
                0x080000e0                ST_AM_Alarm_OFF
                0x08000110                ST_AM_Alarm_ON
                0x08000134                ST_AM_Waiting
 .text          0x0800015c       0xc4 GPIO.o
                0x0800015c                GPIO_Delay
                0x0800017e                GPIO_GetPressureVal
                0x08000194                GPIO_Set_Alarm_Actuator
                0x080001d0                GPIO_Init
 .text          0x08000220       0x34 main.o
                0x08000220                main
 .text          0x08000254       0x58 Main_Program.o
                0x08000254                ST_MP_High_Pressure
                0x08000284                MP_High_Pressure_Detection
 .text          0x080002ac       0x90 Pressure_Sensor.o
                0x080002ac                ST_PS_Init
                0x080002d0                ST_PS_Reading
                0x08000300                ST_PS_Waiting
                0x08000328                PS_Get_Pressure_Value
 .text          0x0800033c       0x90 startup.o
                0x0800033c                H_Fault_Handler
                0x0800033c                MM_Fault_Handler
                0x0800033c                Usage_Fault_Handler
                0x0800033c                Bus_Fault
                0x0800033c                Default_Handler
                0x0800033c                NMI_Handler
                0x08000348                Reset_Handler
 *(.rodata*)
                0x080003cc                _E_text = .
```

- Symbol Table

MINGW64:/e/Embedded/Kero/FirstTerm/Project 1/Project_1_PressureDetection

```
Basem@DESKTOP-NA5H97K MINGW64 /e/Embedded/Kero/FirstTerm/Project 1/Project_1_Pre
ssureDetection
$ arm-none-eabi-nm.exe Pressure_Detection_Lab.elf
2000002c B _E_bss
20000014 D _E_DATA
080003cc T _E_text
20000014 B _S_bss
20000000 D _S_DATA
2000102c B _stack_top
080000a8 T AA_Start_Alarm
080000c4 T AA_Stop_Alarm
20000004 D Alarm_Act_State
20000008 D AM_State
20000015 B AM_Status
0800033c W Bus_Fault
0800033c T Default_Handler
20000014 B E_Alarm_Act_Status
20000018 B E_MP_Status
20000020 B E_PS_Status
0800015c T GPIO_Delay
0800017e T GPIO_GetPressureVal
080001d0 T GPIO_Init
08000194 T GPIO_Set_Alarm_Actuator
0800033c W H_Fault_Handler
08000220 T main
0800033c W MM_Fault_Handler
08000284 T MP_High_Pressure_Detection
20000010 d MP_Pressure_Thershold
2000001c b MP_Pressure_Value
2000000c D MP_State
0800033c W NMI_Handler
08000328 T PS_Get_Pressure_Value
20000024 b PS_Pressure_Value
20000000 D PS_State
08000348 T Reset_Handler
0800001c T ST_Alarm_Act_Init
08000080 T ST_Alarm_Act_OFF
08000058 T ST_Alarm_Act_ON
08000040 T ST_Alarm_Act_Waiting
080000e0 T ST_AM_Alarm_OFF
08000110 T ST_AM_Alarm_ON
08000134 T ST_AM_Waiting
08000254 T ST_MP_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
```

- Simulation



Source Code | Schematic Capture

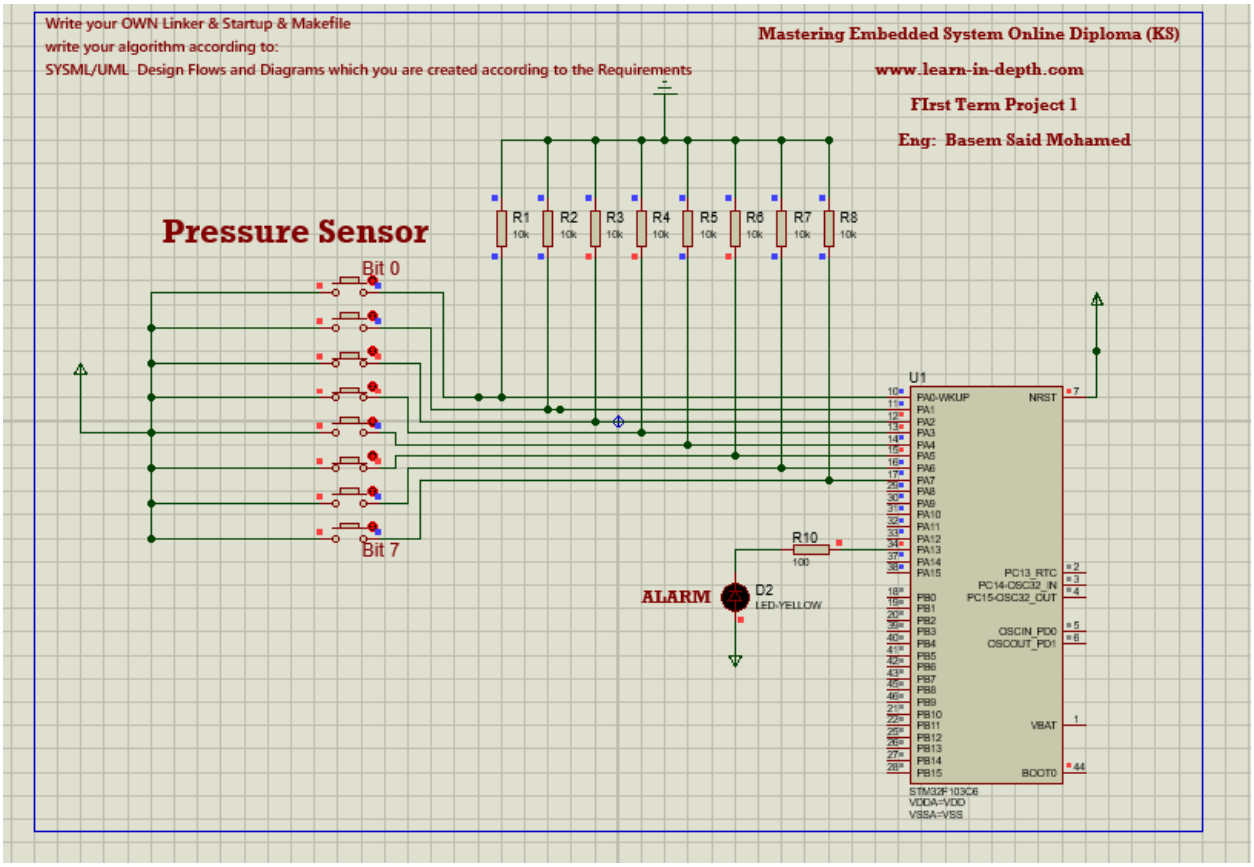
CM3 Source Code - U1

..\GPIO.c

```
-------- /*
-------- * GPIO.c
-------- *
-------- * Created on: 27/10/2023
-------- *      Author: Basem
-------- */
--------
-------- #include "GPIO.h"
--------
-------- void GPIO_Delay(uint32_t nCount)
800015C  {
8000164      for(; nCount != 0; nCount--);
8000172  }
--------
800017E  uint32_t GPIO_GetPressureVal(void){
8000182      return (GPIOA_IDR & 0xFF);
8000188  }
--------
8000194  void GPIO_Set_Alarm_Actuator(uint32_t i){
800019C      if (i == 1){
80001A2          SET_BIT(GPIOA_ODR,13);
--------     }
80001B0      else if (i == 0){
80001B6          RESET_BIT(GPIOA_ODR,13);
--------     }
80001AE  }
--------
80001D0  void GPIO_Init (void){
80001D4      SET_BIT(APB2ENR, 2);
80001E0      GPIOA_CRL &= 0xFF0FFFFF;
80001EC      GPIOA_CRL |= 0x00000000;
80001F4      GPIOA_CRH &= 0xFF0FFFFF;
8000200      GPIOA_CRH |= 0x22222222;
800020C  }
```

CM3 Variables - U1

Name	Address	Value
AM_Status	20000015	AM_Alarm_ON (1)
E_MP_Status	20000018	MP_High_Pressure (0)
MP_Pressure_Value	2000001C	28
MP_Pressure_Thershold	20000010	20
E_PS_Status	20000020	PS_Reading (1)
PS_Pressure_Value	20000024	28
_stack_top	2000102C	0
⊞vectors	08000000	dword[7]
E_Alarm_Act_Status	20000014	Alarm_Act_ON (2)
i	BP+12 = @20000FF0	1



Source Code | Schematic Capture

CM3 Source Code - U1

..\Alarm_Actuator.c

```
800005C      E_Alarm_Act_Status = Alarm_Act_ON ;
--------
--------     // Start Alarm Actuator
8000062      GPIO_Set_Alarm_Actuator(TRUE) ;
--------
--------     // Check Event and Update State
8000068      Alarm_Act_State = STATE(Alarm_Act_Waiting) ;
800006E  }
--------
-------- STATE_Define (Alarm_Act_OFF)
8000080  {
--------     // State Action
8000084      E_Alarm_Act_Status = Alarm_Act_OFF ;
--------
--------     // Start Alarm Actuator
800008A      GPIO_Set_Alarm_Actuator(FALSE) ;
--------
--------     // Check Event and Update State
8000090      Alarm_Act_State = STATE(Alarm_Act_Waiting) ;
8000096  }
--------
-------- void AA_Start_Alarm(void)
80000A8  {
--------     // Update State
80000AC      Alarm_Act_State = STATE(Alarm_Act_ON) ;
80000B2  }
--------
-------- void AA_Stop_Alarm(void)
80000C4  {
--------     // Update State
80000C8      Alarm_Act_State = STATE(Alarm_Act_OFF) ;
80000CE  }
--------
--------
--------
--------
--------
```

CM3 Variables - U1

Name	Address	Value
AM_Status	20000015	AM_Alarm_OFF (0)
E_MP_Status	20000018	MP_High_Pressure (0)
MP_Pressure_Value	2000001C	44
MP_Pressure_Thershold	20000010	20
E_PS_Status	20000020	PS_Reading (1)
PS_Pressure_Value	20000024	44
_stack_top	2000102C	0
⊞vectors	08000000	dword[7]
E_Alarm_Act_Status	20000014	Alarm_Act_OFF (3)