Mastering Embedded System Online Diploma

www.learn-in-depth.com

First Term (Final Project 1)

Eng. Ahmed Sameh

My Profile: https://www.learn-in-depth.com/online-diploma/ahmeds.elshahid%40gmail.com

# Contents

## Table of Figures:

## Requirements Diagram:

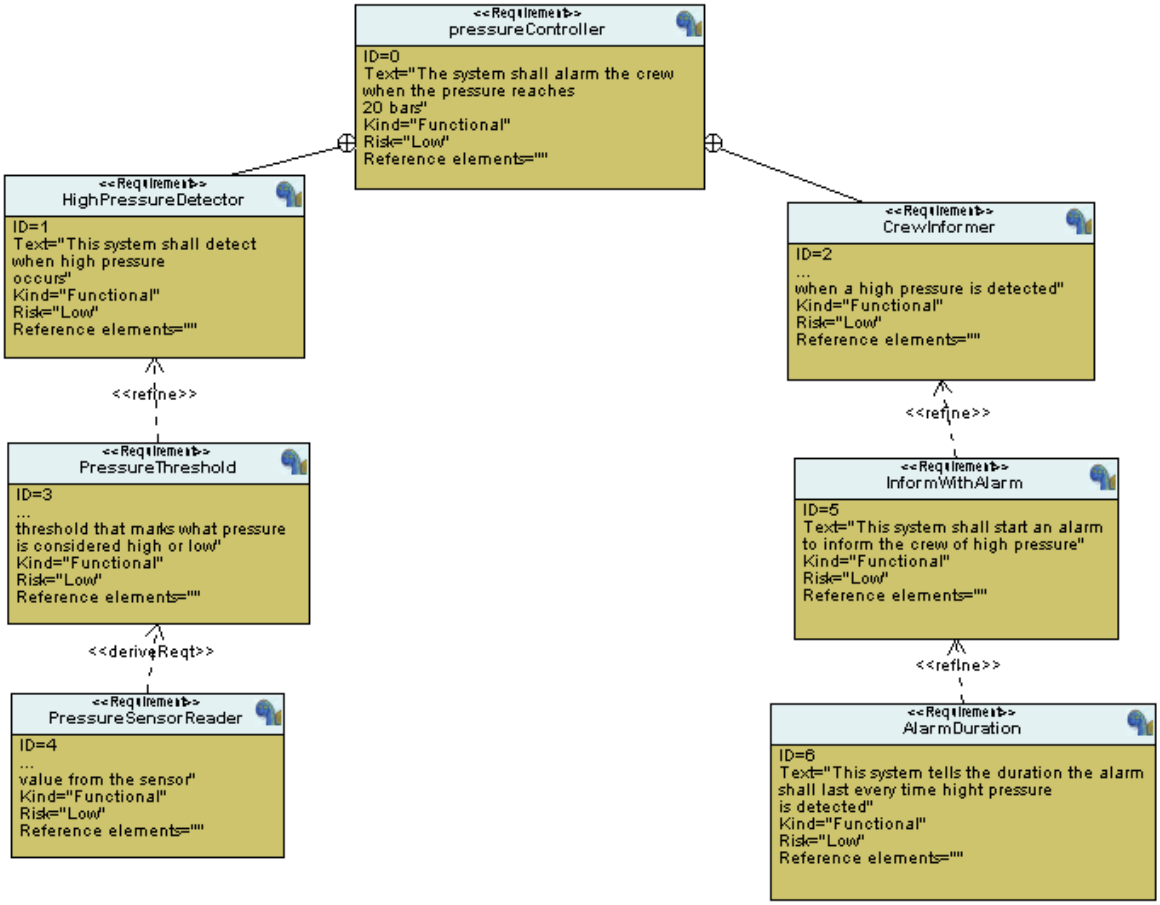


*Figure 1: Requirements Diagram*

## System Analysis:

### Use Case Diagram

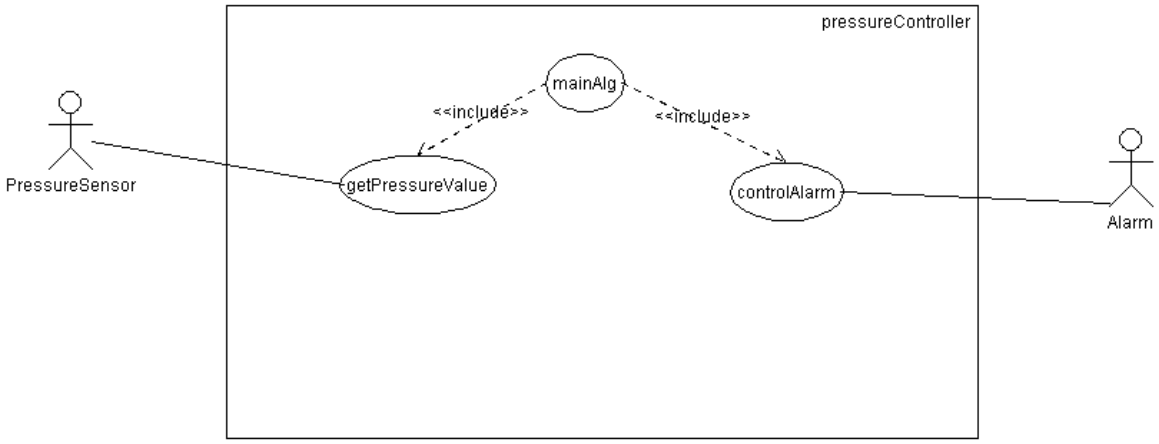


*Figure 2: Use Case Diagram*

## Activity Diagram

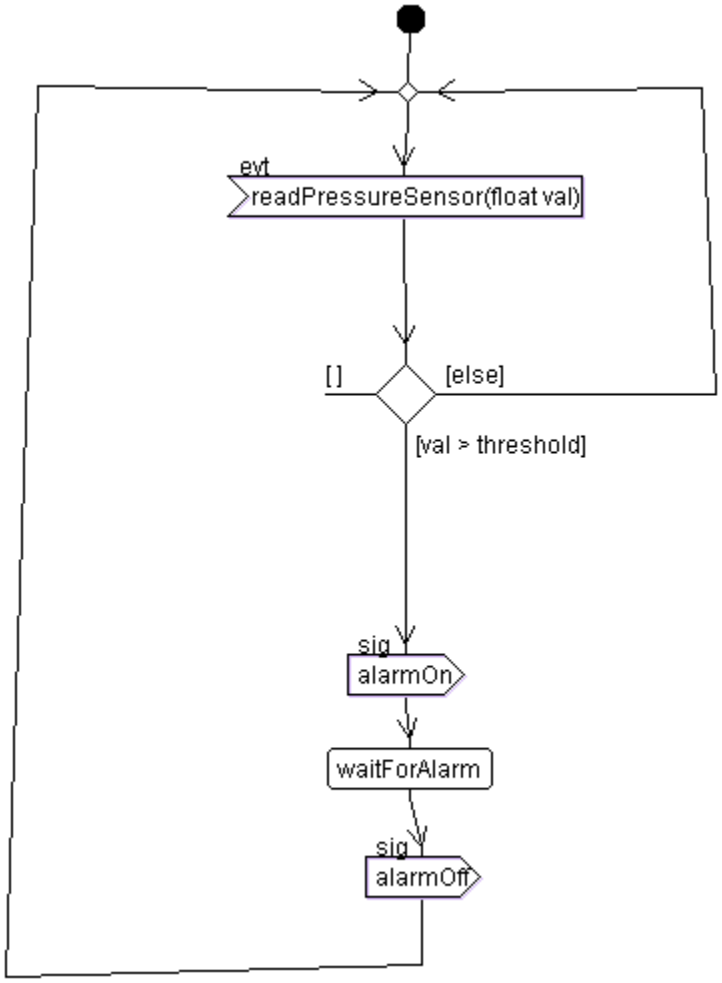


*Figure 3: Activity Diagram*

## Sequence Diagram

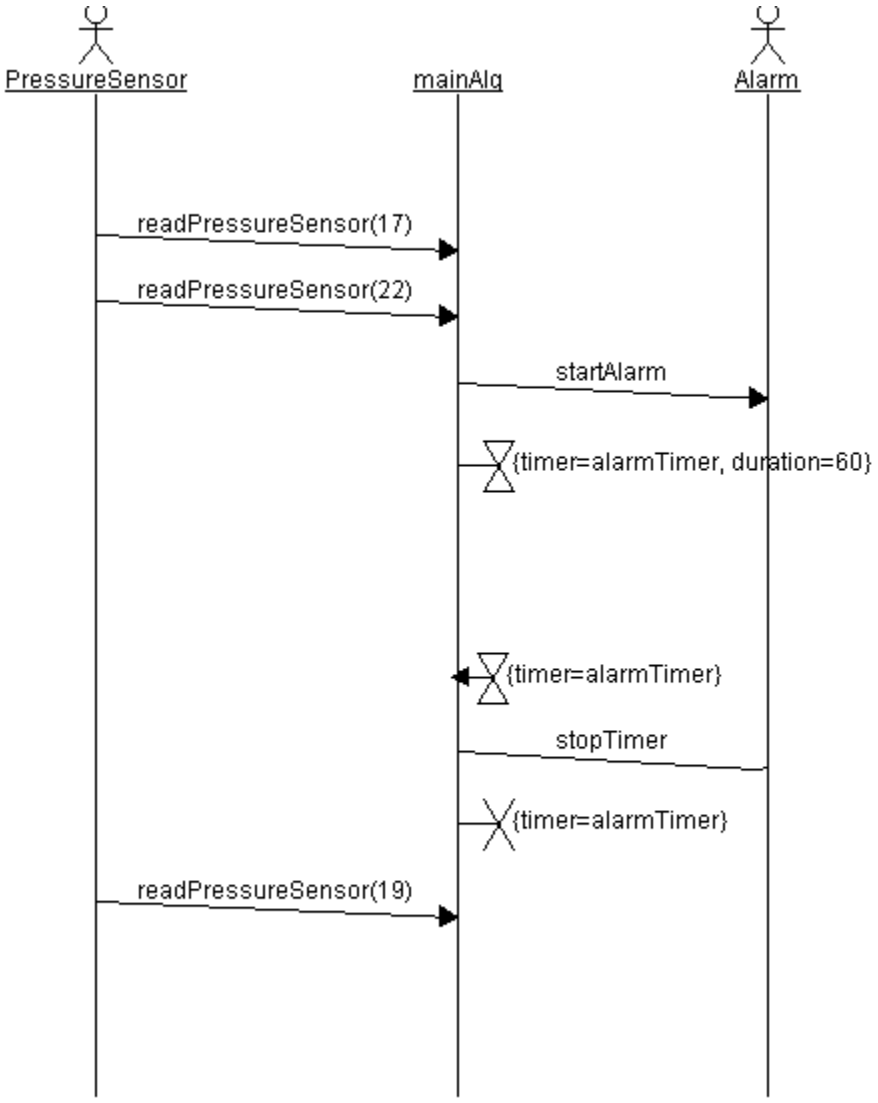


*Figure 4: Sequence Diagram*

## System Design:

### Block Diagram

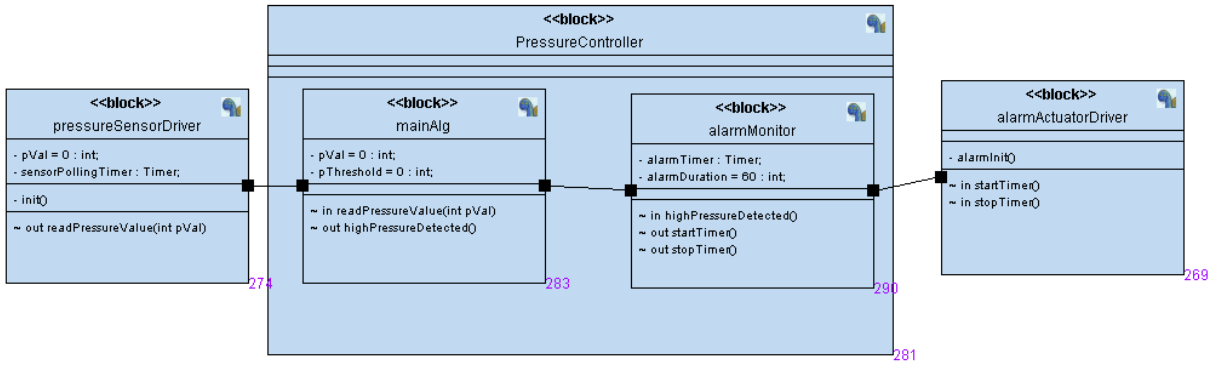


*Figure 5: Block Diagram*

## Pressure Driver State Diagram

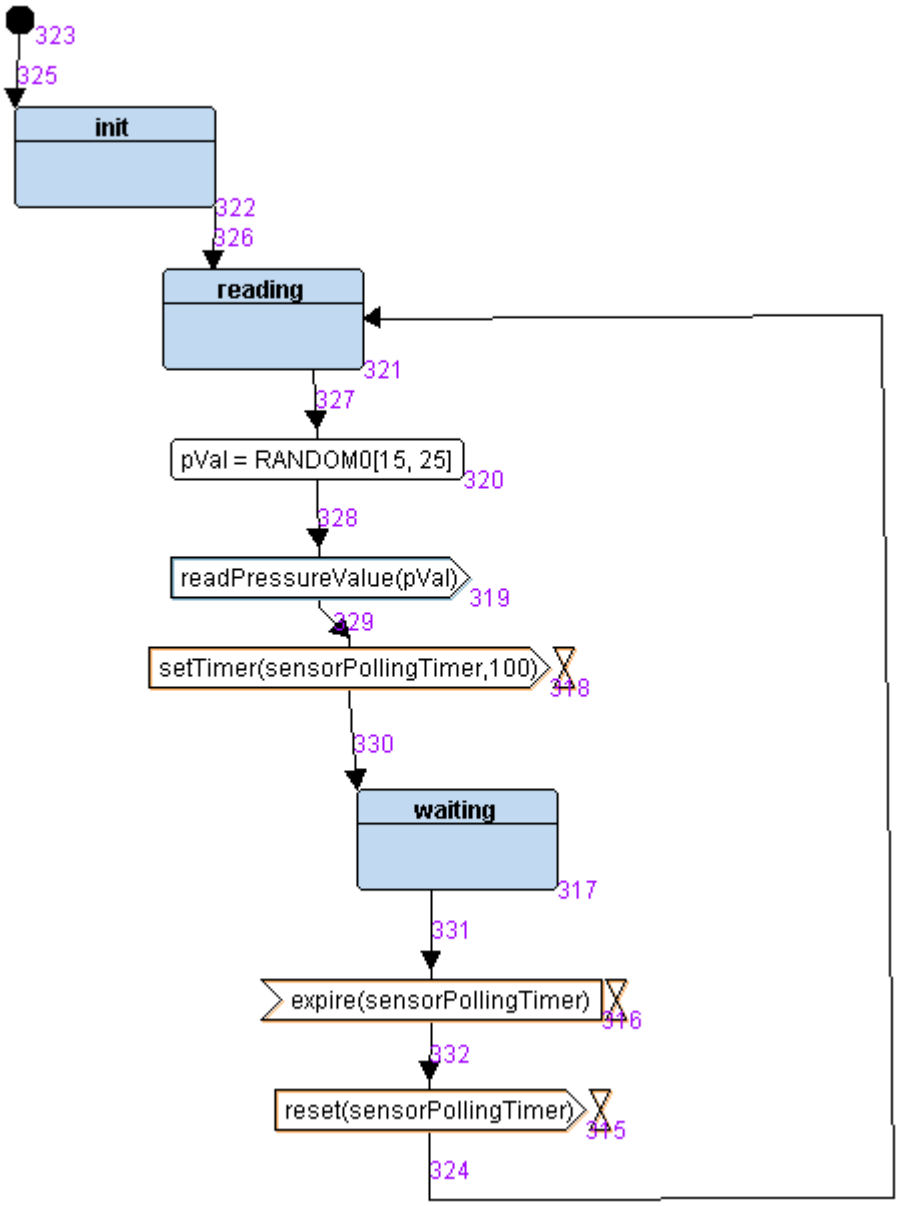


*Figure 6: Pressure Driver State Diagram*

## Main Alg. State Diagram

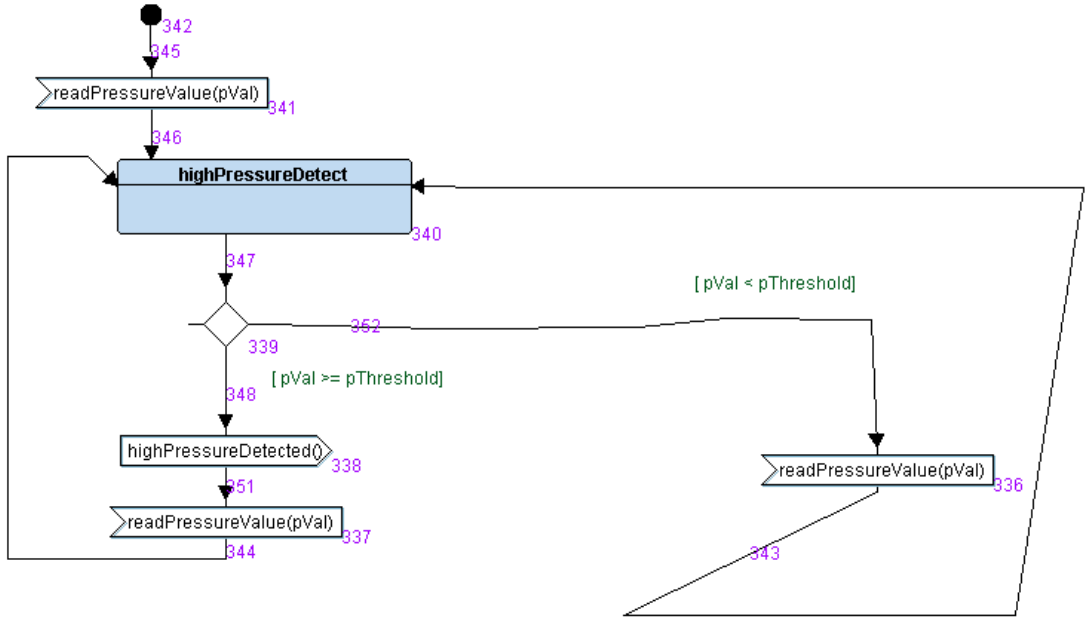


*Figure 7: Main Alg. State Diagram*

## Alarm Monitor State Diagram

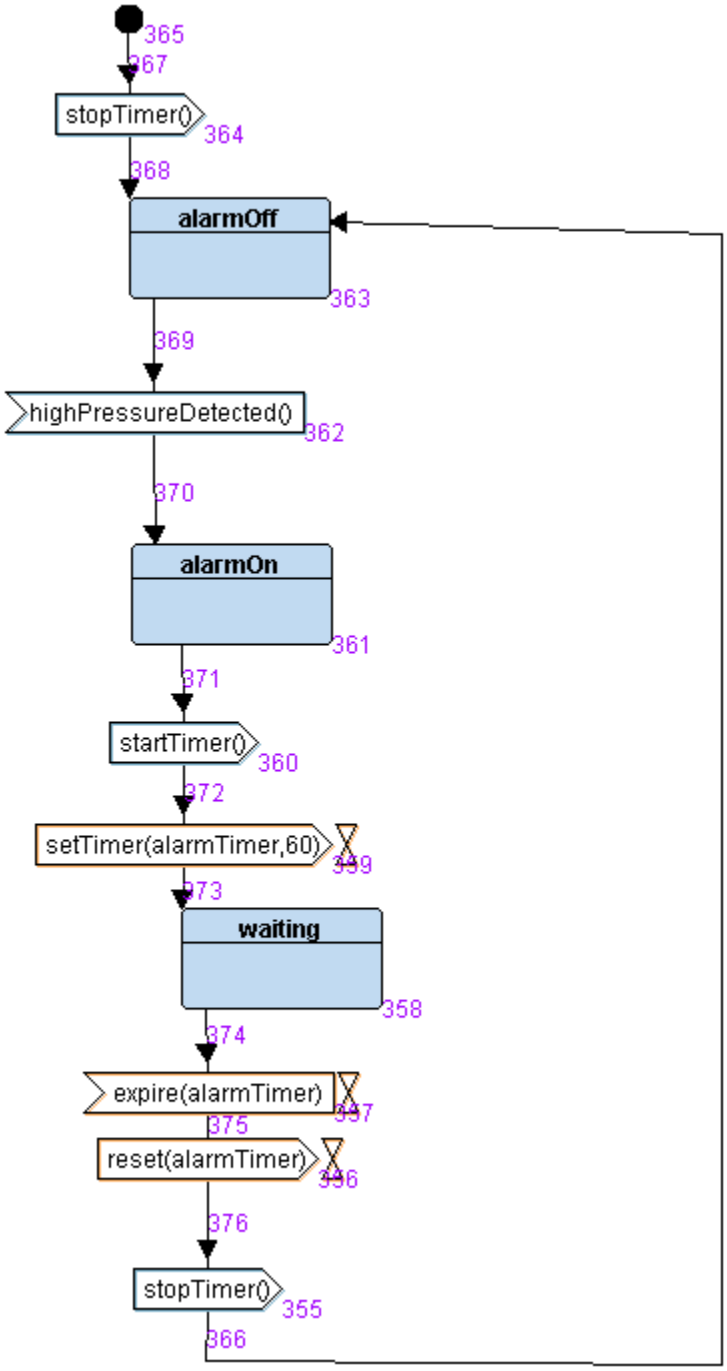


*Figure 8: Alarm Monitor State Diagram*

## Alarm Driver State Diagram

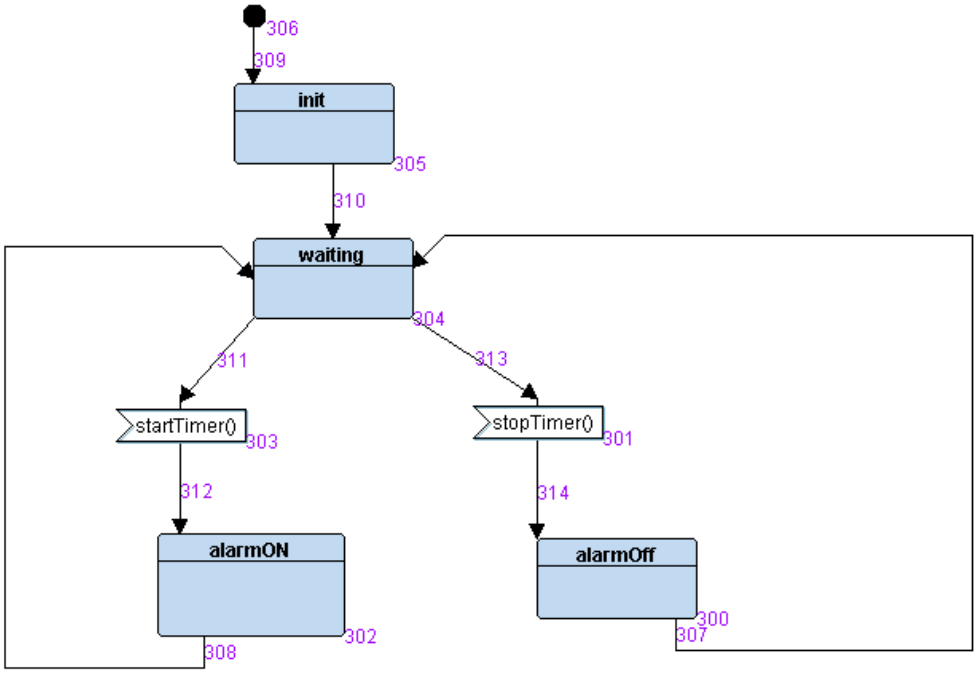


*Figure 9: Alarm Driver State Diagram*

## Codes:

.c and .h Files

```c
#include "alarmMonitor.h"
#include "mainAlg.h"
#include "alarmDriver.h"


void setup()
{
    GPIO_INITIALIZATION();

    pressure_init();
    alarmDriver_init();


    PRESSURE_state = STATE(PRESSURE_reading);
    MAIN_ALG_state = STATE(MAIN_ALG_highPressureDetect);
    ALARM_MONITOR_state = STATE(ALARM_MONITOR_alarmOFF);
    ALARM_DRIVER_state = STATE(ALARM_DRIVER_waiting);

}

int main ()
{
    setup();

    while (1)
    {
        //Implement your Design
        PRESSURE_state();
        MAIN_ALG_state();
        ALARM_MONITOR_state();
        ALARM_DRIVER_state();
    }

}
```

*Figure 10: main.c*

```c
#include "driver.h"
#include <stdint.h>
#include <stdio.h>
void Delay(int nCount)
{
    for(; nCount != 0; nCount--);
}

int getPressureVal(){
    return (GPIOA_IDR & 0xFF);
}

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

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

*Figure 11: driver.c*

```c
#include <stdint.h>
#include <stdio.h>

#define SET_BIT(ADDRESS,BIT)   ADDRESS |=  (1<<BIT)
#define RESET_BIT(ADDRESS,BIT) ADDRESS &= ~(1<<BIT)
#define TOGGLE_BIT(ADDRESS,BIT)  ADDRESS ^=  (1<<BIT)
#define READ_BIT(ADDRESS,BIT) ((ADDRESS) &   (1<<(BIT)))


#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 Delay(int nCount);
int getPressureVal();
void Set_Alarm_actuator(int i);
void GPIO_INITIALIZATION ();

```

*Figure 12: driver.h*

```c
#include "alarmDriver.h"
#include "driver.h"

void (*ALARM_DRIVER_state) ();
void alarmDriver_init()
{
    DPRINTF("ALARM INIT...\n");
    Set_Alarm_actuator(1);
}
void startTimer()
{
    ALARM_DRIVER_state = STATE(ALARM_DRIVER_alarmON);
    ALARM_DRIVER_state();
}
void stopTimer()
{
    ALARM_DRIVER_state = STATE(ALARM_DRIVER_alarmOFF);
    ALARM_DRIVER_state();
}
STATE_define(ALARM_DRIVER_alarmON)
{
    alarmDriver_state_id = alarmDriver_alarmON;

    Set_Alarm_actuator(0);

    ALARM_DRIVER_state = STATE(ALARM_DRIVER_waiting);
}
STATE_define(ALARM_DRIVER_alarmOFF)
{
    alarmDriver_state_id = alarmDriver_alarmOFF;

    Set_Alarm_actuator(1);

    ALARM_DRIVER_state = STATE(ALARM_DRIVER_waiting);
}
STATE_define(ALARM_DRIVER_waiting)
{
    alarmDriver_state_id = alarmDriver_waiting;
}
```

*Figure 13: alarmDriver.c*

```c
#ifndef _ALARM_DRIVER_H_
#define _ALARM_DRIVER_H_

#include "stateMachine.h"

enum {
    alarmDriver_alarmON,
    alarmDriver_alarmOFF,
    alarmDriver_waiting
}alarmDriver_state_id;

void alarmDriver_init();

STATE_define(ALARM_DRIVER_alarmON);
STATE_define(ALARM_DRIVER_alarmOFF);
STATE_define(ALARM_DRIVER_waiting);

extern void (*ALARM_DRIVER_state) ();

#endif
```

*Figure 14: alarmDriver.c*

```c
#include "alarmMonitor.h"
#include "driver.h"

void (*ALARM_MONITOR_state) ();
alarmDuration = 600000;
void highPressureDetected()
{
    DPRINTF("high pressure detected: starting alarm in progress.....\n");
    ALARM_MONITOR_state = STATE(ALARM_MONITOR_alarmON);
}
STATE_define(ALARM_MONITOR_alarmON)
{
    alarmMonitor_state_id = alarmMonitor_alarmON;
    DPRINTF("ALARM_MONITOR_alarmON state.....\n");

    startTimer();


    ALARM_MONITOR_state = STATE(ALARM_MONITOR_waiting);
}
STATE_define(ALARM_MONITOR_alarmOFF)
{
    alarmMonitor_state_id = alarmMonitor_alarmOFF;
    DPRINTF("ALARM_MONITOR_alarmOFF state.....\n");

    stopTimer();

    ALARM_MONITOR_state = STATE(ALARM_MONITOR_alarmOFF);
}
STATE_define(ALARM_MONITOR_waiting)
{
    alarmMonitor_state_id = alarmMonitor_waiting;
    DPRINTF("ALARM_MONITOR_waiting state.....\n");
    Delay(alarmDuration);
    ALARM_MONITOR_state = STATE(ALARM_MONITOR_alarmOFF);
    ALARM_MONITOR_state();
}
```

*Figure 15: alarmMonitor.c*

```c
#ifndef _ALARM_MONITOR_H_
#define _ALARM_MONITOR_H_

#include "stateMachine.h"

enum {
    alarmMonitor_alarmON,
    alarmMonitor_alarmOFF,
    alarmMonitor_waiting
}alarmMonitor_state_id;

STATE_define(ALARM_MONITOR_alarmON);
STATE_define(ALARM_MONITOR_alarmOFF);
STATE_define(ALARM_MONITOR_waiting);

extern void (*ALARM_MONITOR_state) ();

#endif
```

*Figure 16: alarmMonitor.h*

```c
#include "mainAlg.h"
#include "platform_Types.h"

static uint32 pVal = 0;
static uint32 pThreshold = 20;
void (*MAIN_ALG_state) ();

void readPressureValue(int pValue)
{
    pVal = pValue;
    DPRINTF("mainAlg:..............pressure value=%d.............../n",pVal);
}

STATE_define(MAIN_ALG_highPressureDetect)
{
    mainAlg_state_id = mainAlg_highPressireDetect;

    if(pVal >= pThreshold)
        highPressureDetected();
    DPRINTF("MAIN_ALG_highPressureDetect.............../n",pVal);
    MAIN_ALG_state = STATE(MAIN_ALG_highPressureDetect);
}

```

*Figure 17: mainAlg.c*

```
#ifndef _MAIN_ALG_H_
#define _MAIN_ALG_H_

#include "stateMachine.h"

enum {
        mainAlg_highPressireDetect
}mainAlg_state_id;


STATE_define(MAIN_ALG_highPressureDetect);

extern void (*MAIN_ALG_state) ();

#endif
```

*Figure 18: mainAlg.h*

```c
#include "pressureDriver.h"
#include "driver.h"
#include "platform_Types.h"

void (*PRESSURE_state) ();
static uint32 pVal = 0;

void pressure_init()
{
    DPRINTF("PRESSURE INIT...\n");
}

STATE_define(PRESSURE_reading)
{
    pressureDriver_state_id = pressure_reading;

    //reading from sensor
    pVal = getPressureVal();

    DPRINTF("pressure_reading state, pVal = %d\n",pVal);

    //sending the pressure signal
    readPressureValue(pVal);

    //next state
    PRESSURE_state = STATE(PRESSURE_waiting);

}


STATE_define(PRESSURE_waiting)
{
    pressureDriver_state_id = pressure_waiting;

    //Delay
    Delay(200000);

    //next state
    PRESSURE_state = STATE(PRESSURE_reading);
}

```

*Figure 19: pressureDriver.c*

```c
#ifndef _PRESSURE_DRIVER_H_
#define _PRESSURE_DRIVER_H_

#include "stateMachine.h"

enum {
    pressure_reading,
    pressure_waiting
}pressureDriver_state_id;

void pressure_init();

STATE_define(PRESSURE_reading);
STATE_define(PRESSURE_waiting);

extern void (*PRESSURE_state) ();

#endif
```

*Figure 20: pressureDriver.h*

```c
#ifndef _STATEMAHCINE_H_
#define _STATEMAHCINE_H_

#define STATE_define(_stateFunc_)   void ST_##_stateFunc_(void)
#define STATE(_stateFunc_)  ST_##_stateFunc_

#define DPRINTF(...)    {/*fflush(stdin);\
                        fflush(stdout);\
                        printf(__VA_ARGS__);\
                        fflush(stdin);\
                        fflush(stdout);*/}

#include <stdio.h>
#include <stdlib.h>
#include <platform_Types.h>

//states connection
void readPressureValue(int pValue);
void highPressureDetected();
void startTimer();
void stopTimer();

#endif
```

*Figure 21: stateMachine.h*

```
 4   *      Created on: 27 Jan 2023
 5   *      Author:     Ahmed Sameh
 6   */
 7  #ifndef PLATFORM_TYPES_H_
 8  #define PLATFORM_TYPES_H_
 9  #ifndef boolean
10  #define boolean unsigned char
11  #endif
12  #define true    1
13  #define false   0
14  #ifndef FALSE
15  #define FALSE   (boolean)false
16  #endif
17  #ifndef TRUE
18  #define TRUE    (boolean)true
19  #endif
20  typedef char sint8;
21  typedef unsigned char uint8;
22  typedef char char_t;
23  typedef short sint16;
24  typedef unsigned short uint16;
25  typedef long sint32;
26  typedef unsigned long uint32;
27  typedef long long sint64;
28  typedef unsigned long long uint64;
29  typedef volatile char vint8_t;
30  typedef volatile unsigned char vuint8_t;
31
32  typedef volatile short vint16_t;
33  typedef volatile unsigned short vuint16_t;
34
35  typedef volatile long vint32_t;
36  typedef volatile unsigned long vuint32_t;
37
38  typedef volatile long long vint64_t;
39  typedef volatile unsigned long long vuint64_t;
40
41  typedef float float32;
42  typedef double float64;
43
44
45  #endif /* PLATFORM_TYPES_H_ */
```

*Figure 22: platformTypes.h*

## Makefile

```
# created by Eng Ahmed Sameh in Mastering embedded systems diploma unit 5 project 1
#CC stands for C compiler as it is used to change compiler toolchain
CC=arm-none-eabi-
#CFlags means the option used while compiling
CFLAGS=-mcpu=cortex-m3 -mthumb -gdwarf-2
#INCS stands for include as we should include all files in the folders
INCS=-I .
#Libs stands for libraries as it is used to include libraries made before
LIBS=

SRC=$(wildcard *.c)
OBJ=$(SRC:.c=.o)
As=$(wildcard *.s)
AsOBJ=$(As:.s=.o)

ProjectName= FirstTermProject1

# Add an @ before a command to stop it from being printed
all: $(ProjectName).bin
    @echo '***************Done***************'

# the 2> log is used to forward error messages into a log file so that the terminal stays clean
# we can't insert comments besides the commands or the target: dependency, comments must be in sperated lines
# %.o meany any file.o, $@ means target file, $< means first dependency file
%.o: %.s
    $(CC)as.exe $(CFLAGS) $< -o $@ 2> log

%.o: %.c
    $(CC)gcc.exe -c $(CFLAGS) $(INCS) $< -o $@

$(ProjectName).elf: $(OBJ) $(AsOBJ)
    $(CC)ld.exe -T linker_script.ld $(LIBS) $(OBJ) $(AsOBJ) -o $@ -Map=Map_file.map

$(ProjectName).bin: $(ProjectName).elf
    $(CC)objcopy.exe -O binary $< $@

clean_all:
    rm *.o *.elf *.bin

clean:
    rm *.elf *.bin
```

*Figure 23: Makefile*

## Linker file

```
/*
    Author: Ahmed Sameh Elshahed
    project: FirstTerm project 1
    file name: linker_script.ld
*/

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 = .;

        . = ALIGN(4);
        . = . + 0x1000;
        _stack_top = .;
    } > sram
}
```

*Figure 24: linker_script.ld*

## Startup file

```c
// created by Ahmed Sameh in Mastering embedded systems diploma
#include <platform_Types.h>
extern int main(void);
void Reset_Handler(void);
void Default_Handler(){
    Reset_Handler();
}
void NMI_Handler(void) __attribute__((weak, alias("Default_Handler")));
void H_fault_Handler(void) __attribute__((weak, alias("Default_Handler")));
void MM_fault_Handler(void) __attribute__((weak, alias("Default_Handler")));
void Bus_fault(void) __attribute__((weak, alias("Default_Handler")));
void Usage_fault_Handler(void) __attribute__((weak, alias("Default_Handler")));
//reserve stack
static uint32 Stack_top[256];
void (* const vectors_arr[])(void) __attribute__((section(".vectors"))) = {
    (void(*)(void))     ((uint32)Stack_top + (uint32)sizeof(Stack_top)),
    &Reset_Handler,
    &NMI_Handler,
    &H_fault_Handler,
    &MM_fault_Handler,
    &Bus_fault,
    &Usage_fault_Handler };
extern uint32 _E_text;
extern uint32 _S_data;
extern uint32 _E_data;
extern uint32 _S_bss;
extern uint32 _E_bss;
void Reset_Handler(void) {
    // copy data section from flash to ram
    uint32 DataSize = (uint8 *)&_E_data - (uint8 *)& _S_data;
    uint8 *P_src = (uint8 *)&_E_text;
    uint8 *P_dst = (uint8 *)&_S_data;
    uint32 i;
    for(i = 0; i < DataSize; i++)
        *((uint8 *)P_dst++) = *((uint8 *)P_src++);
    // initialize the bss section with zeros
    uint32 BssSize = (uint8 *) &(_E_bss) - (uint8 *) &(_S_bss);
    P_dst = (uint8*) &(_S_bss);
    for(i = 0; i < BssSize; i++)
        *((uint8 *)P_dst++) = (uint8)0;
    // jump to main
    main(); }
```

*Figure 25: startup.c*

## Map_file.map

```
                    0x1                alarmDriver.o
alarmMonitor_state_id
                    0x1                alarmMonitor.o
mainAlg_state_id    0x1                main.o
pressureDriver_state_id
                    0x1                main.o
MAIN_ALG_state      0x4                mainAlg.o
PRESSURE_state      0x4                pressureDriver.o
ALARM_MONITOR_state
                    0x4                alarmMonitor.o

Memory Configuration

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

Linker script and memory map


.text           0x08000000      0x4f0
 *(.vectors*)
 .vectors       0x08000000       0x1c startup.o
                0x08000000                vectors_arr
 *(.text*)
 .text          0x0800001c       0xd0 alarmDriver.o
                0x0800001c                alarmDriver_init
                0x0800002c                startTimer
                0x08000050                stopTimer
                0x08000074                ST_ALARM_DRIVER_alarmON
                0x080000a4                ST_ALARM_DRIVER_alarmOFF
                0x080000d4                ST_ALARM_DRIVER_waiting
 .text          0x080000ec       0xb8 alarmMonitor.o
                0x080000ec                highPressureDetected
                0x08000108                ST_ALARM_MONITOR_alarmON
                0x08000134                ST_ALARM_MONITOR_alarmOFF
                0x08000160                ST_ALARM_MONITOR_waiting
 .text          0x080001a4      0x10c driver.o
                0x080001a4                Delay
                0x080001c8                getPressureVal
                0x080001e0                Set_Alarm_actuator
                0x08000230                GPIO_INITIALIZATION
 .text          0x080002b0       0x98 main.o
                0x080002b0                setup
                0x0800030c                main
 .text          0x08000348       0x64 mainAlg.o
                0x08000348                readPressureValue
                0x08000368                ST_MAIN_ALG_highPressureDetect
 .text          0x080003ac       0x88 pressureDriver.o
                0x080003ac                pressure_init
                0x080003b8                ST_PRESSURE_reading
                0x08000400                ST_PRESSURE_waiting
 .text          0x08000434       0xbc startup.o
                0x08000434                Bus_fault
                0x08000434                MM_fault_Handler
                0x08000434                Usage_fault_Handler
                0x08000434                H_fault_Handler
                0x08000434                Default_Handler
```

*Figure 26: Map_file.map*

## Symbol Table

```
20000410 B _E_bss
20000008 D _E_data
080004f0 T _E_text
20000008 B _S_bss
20000000 D _S_data
20001410 B _stack_top
20001410 B ALARM_DRIVER_state
2000141c B ALARM_MONITOR_state
0800001c T alarmDriver_init
20001414 B alarmDriver_state_id
20000000 D alarmDuration
20001418 B alarmMonitor_state_id
08000434 W Bus_fault
08000434 T Default_Handler
080001a4 T Delay
080001c8 T getPressureVal
08000230 T GPIO_INITIALIZATION
08000434 W H_fault_Handler
080000ec T highPressureDetected
0800030c T main
20001424 B MAIN_ALG_state
20001420 B mainAlg_state_id
08000434 W MM_fault_Handler
08000434 W NMI_Handler
080003ac T pressure_init
20001428 B PRESSURE_state
20001421 B pressureDriver_state_id
20000004 d pThreshold
20000008 b pVal
2000000c b pVal
08000348 T readPressureValue
08000440 T Reset_Handler
080001e0 T Set_Alarm_actuator
080002b0 T setup
080000a4 T ST_ALARM_DRIVER_alarmOFF
08000074 T ST_ALARM_DRIVER_alarmON
080000d4 T ST_ALARM_DRIVER_waiting
08000134 T ST_ALARM_MONITOR_alarmOFF
08000108 T ST_ALARM_MONITOR_alarmON
08000160 T ST_ALARM_MONITOR_waiting
08000368 T ST_MAIN_ALG_highPressureDetect
080003b8 T ST_PRESSURE_reading
08000400 T ST_PRESSURE_waiting
20000010 b Stack_top
0800002c T startTimer
08000050 T stopTimer
08000434 W Usage_fault_Handler
08000000 T vectors_arr
```

*Figure 27: symbol Table*

## Simulation Results:

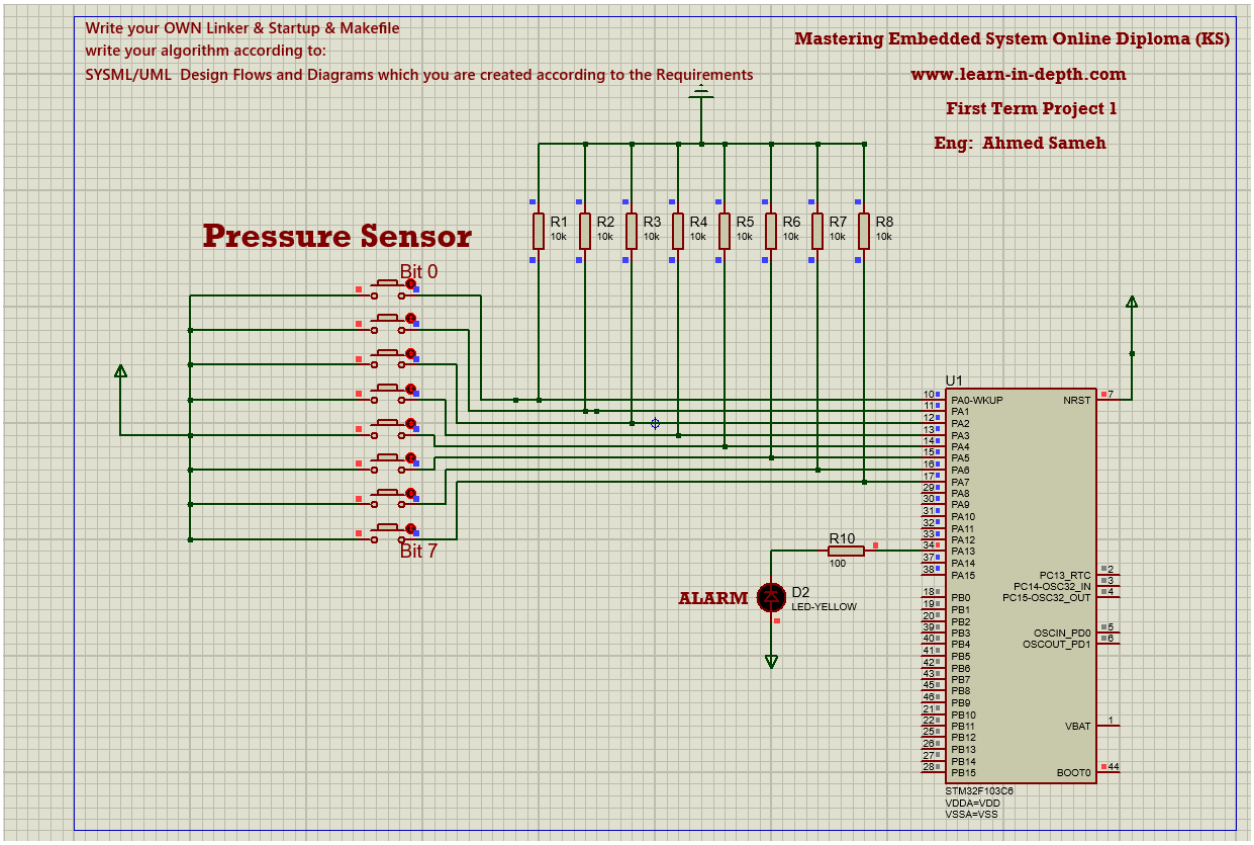


*Figure 28: simulation results when pressure is less than 20*

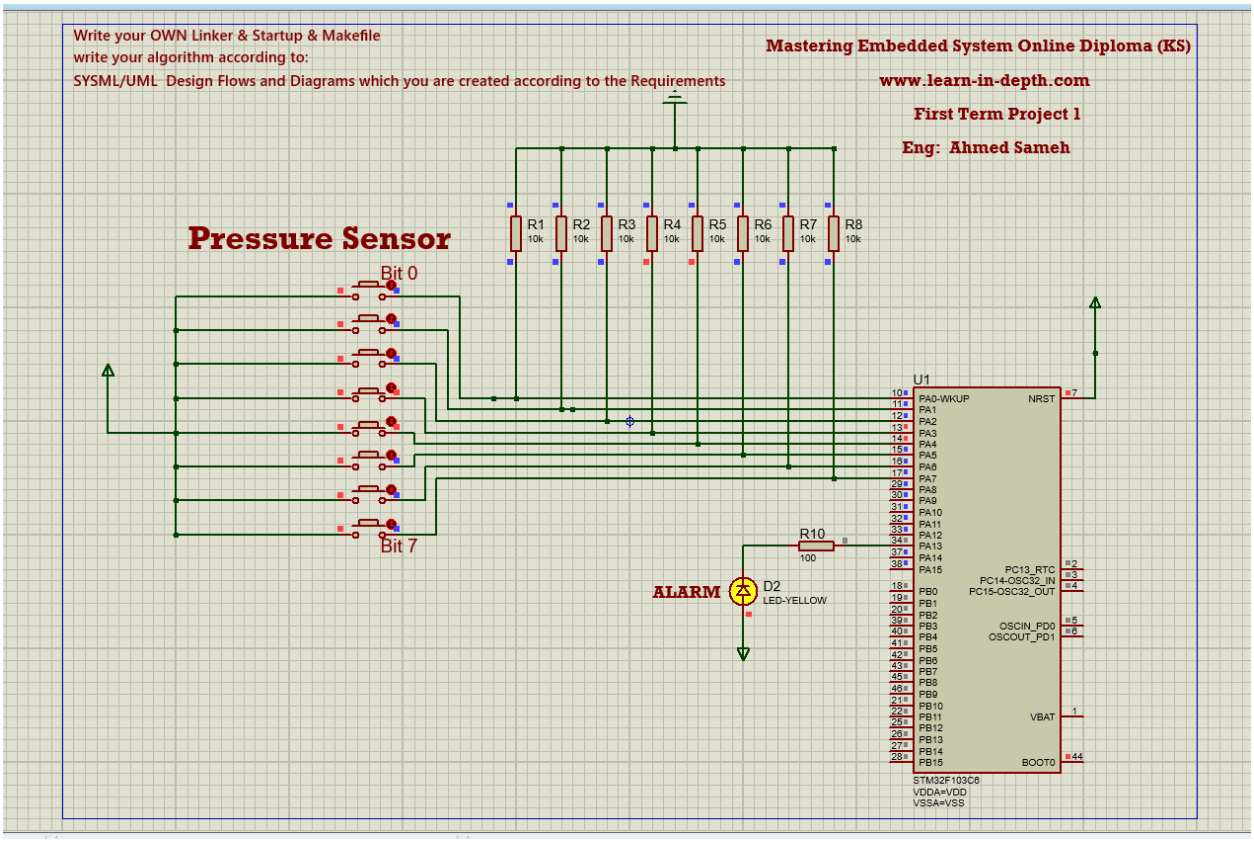


*Figure 29: simulation results when pressure is larger than or equal to 20*