MASTERING EMBEDDED SYSTEM ONLINE DIPLOMA

First Term

PROJECT 1

ENG. OMAR SHAWKY MOHAMED

https://www.learn-in-depth.com/online-diploma/omarshawky9@gmail.com

PRESSURE DETECTION PROJECT

1 INTRODUCTION

This project is an implementation all of the previously studied subjects (C, Embedded C, State Machines). through a pressure detection system that gives off an alarm upon the pressure exceeding a certain threshold. The project goes through the full V cycle from requirement diagram till the low-level design.

2 DESIGN SEQUENCE

2.1 Case Study

The client requirements are (notifying the crew through alarm when pressure exceeds 20 bar , the alarm should stay on for 60 seconds , keeping track of the measured values).

basic assumptions to avoid any further conflicts:

- The controller setup and shutdown procedures are not modeled.
- The controller maintenance is not modeled.
- The pressure sensor never fails.
- The alarm actuator never fails.
- The controller never faces power cut.

The keeping track of the measured values will be implemented in the second version of the program.

2.2 SDLC Model

Concerning the Software Development Life Cycle model, we decided to choose the V-Model in order to focus on the implementation phase at first then proceed to the validation & testing phase.

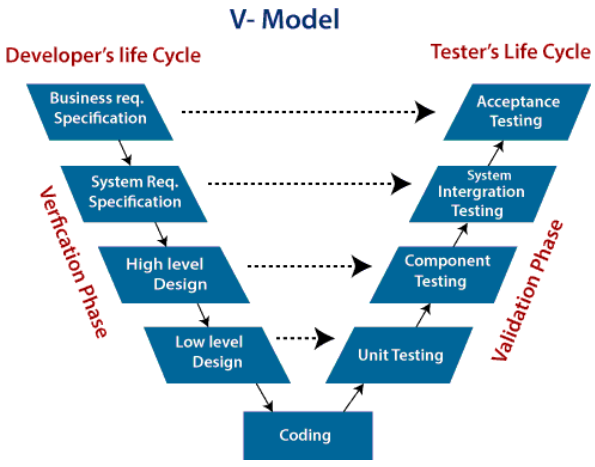


Figure 1 - SDLC V-Model

2.3 Requirement

This is the break down of the requirements listed above.

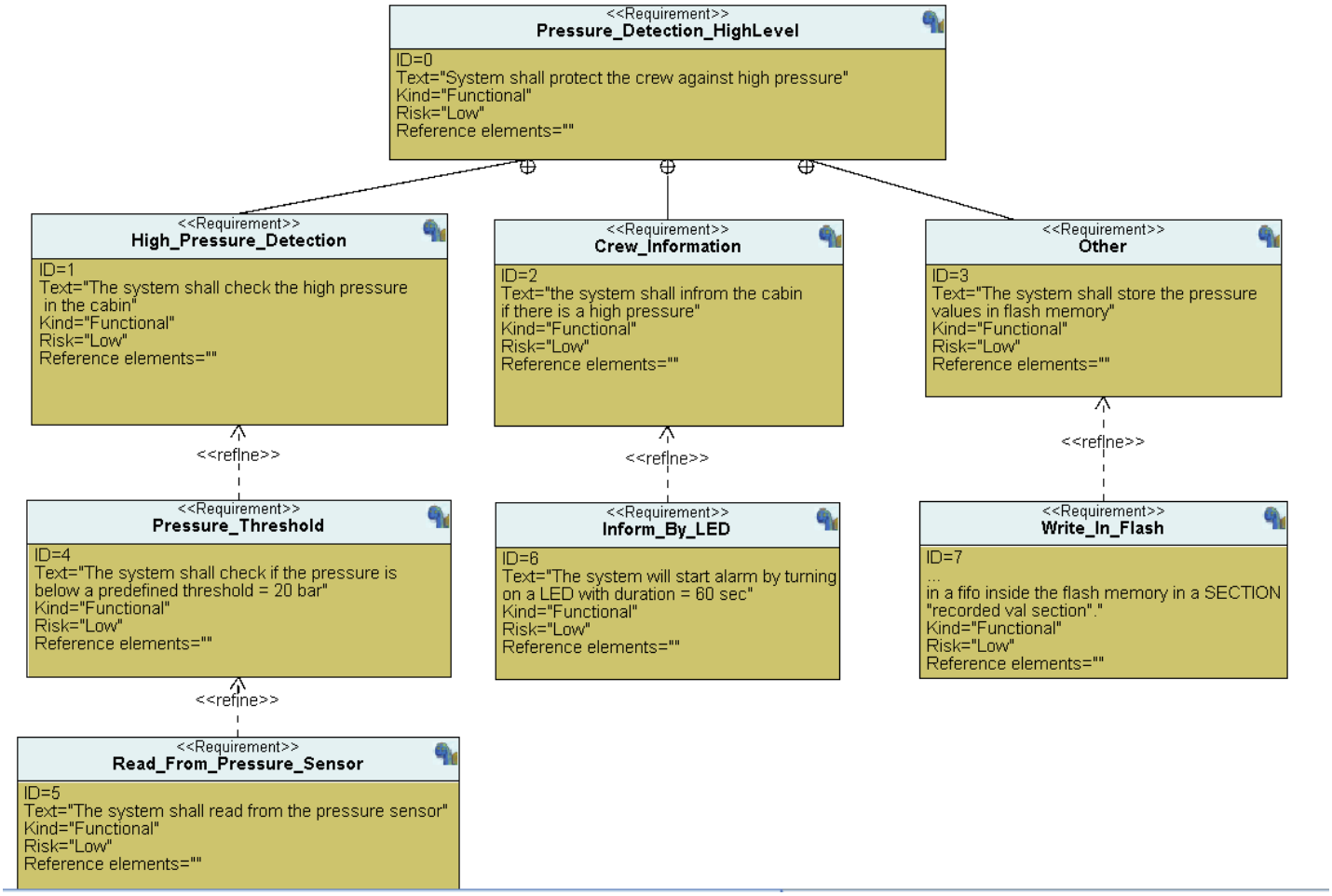


Figure 2 - Requirement Diagram

2.4 System Exploration/Partitioning

In this section, we should discuss which ECU-s we will use to develop this system. We decided that we will implement the following system using the STM32F103F6 microcontroller.

This microcontroller has ARM 32-bits Cortex M3 processor.

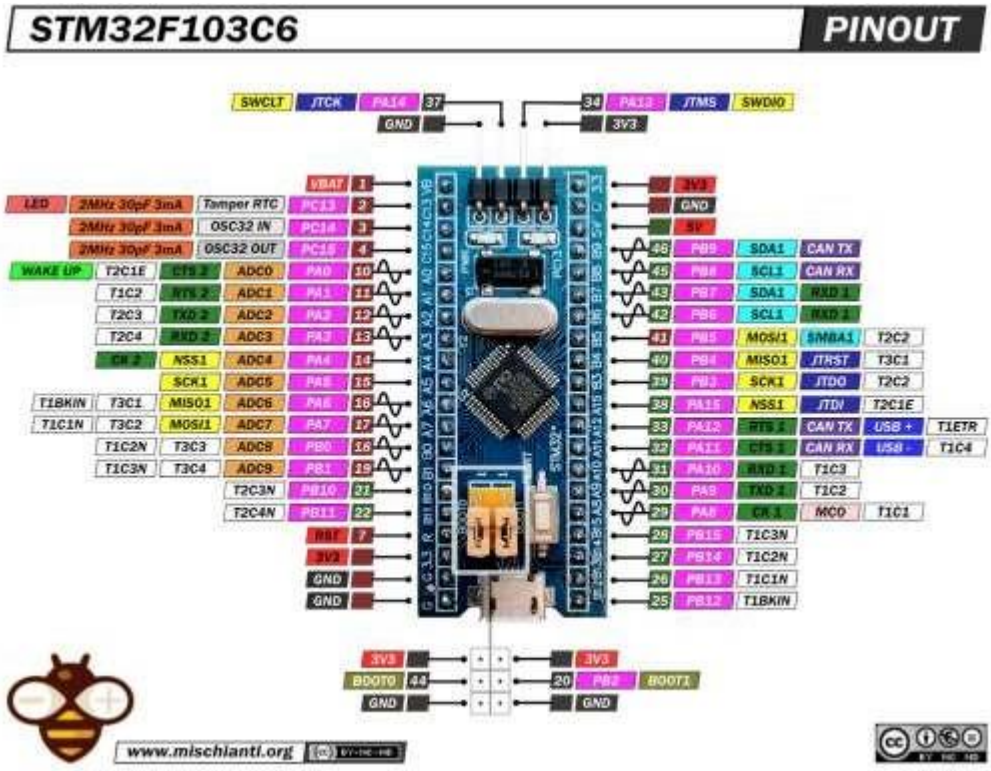


Figure 4 - STM32F103C6 Pinout

2.5 System Analysis

In this section, we will design the main 3 UML diagrams that elaborates the flow of this software system. The UML diagrams to be designed are Use Case Diagram, Activity Diagram and Sequence Diagram.

2.5.1 Use Case Diagram

This diagram shows the main actors in the system and each of their responsibility cases.

Main actors:

- Pressure Sensor.
- Alarm Actuator.
- Flash Memory.

Main Responsibilities:

- Main Algorithm.
- Get Pressure Value.
- Monitor Alarm.
- Store Pressure Value.

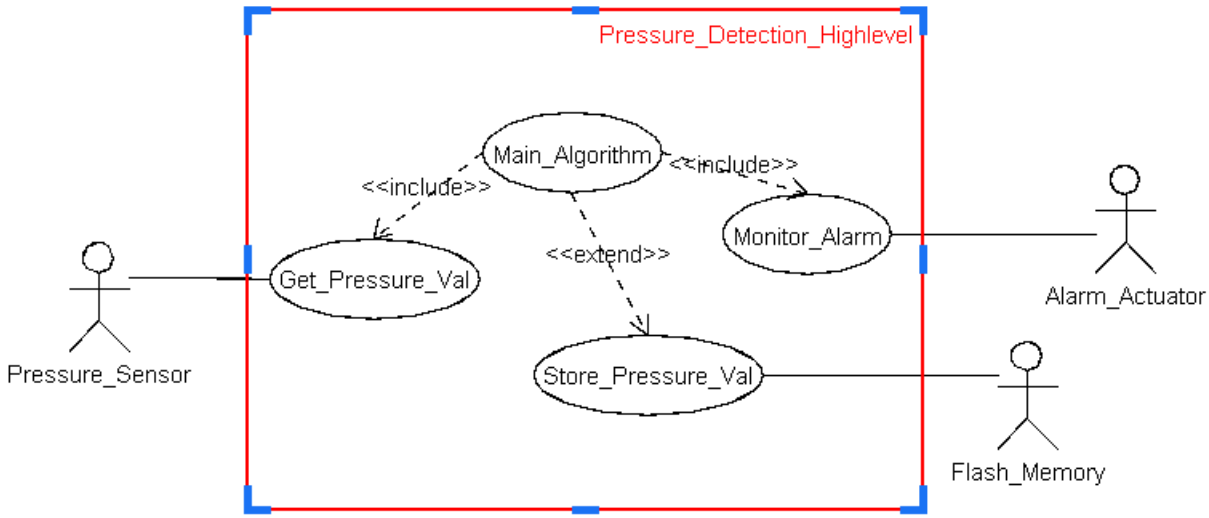


Figure 5 - Use Case Diagram

2.5.2 Activity Diagram

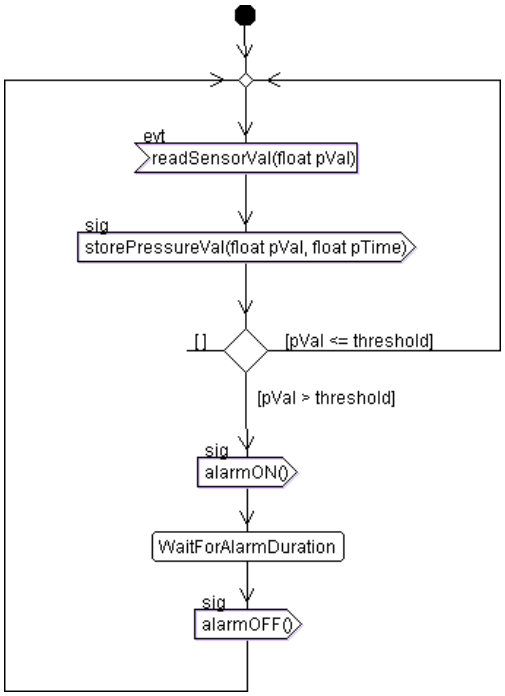


Figure 6 - Activity Diagram

2.5.3 Sequence Diagram

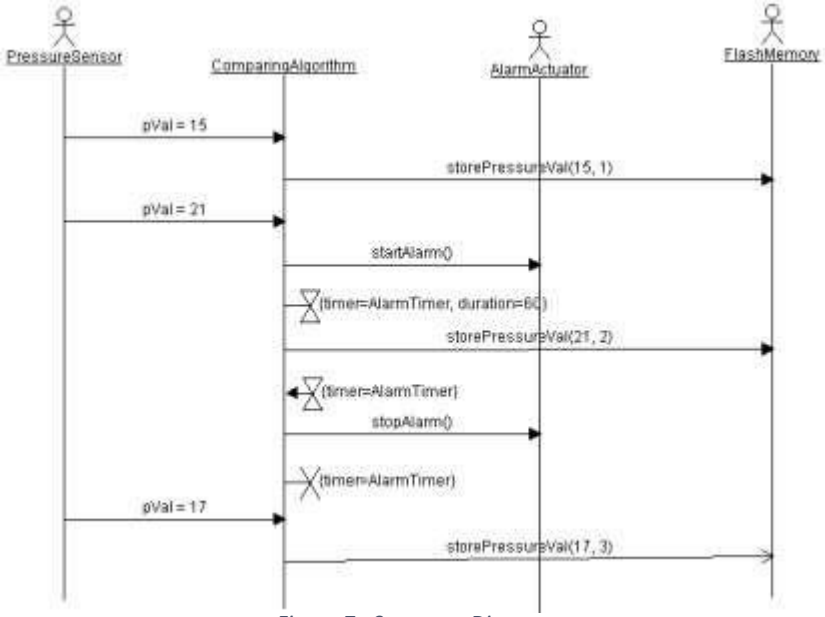


Figure 7 - Sequence Diagram

2.6 System Design

Following the completion of the system analysis phase, the next step is to advance to the system design phase. During this phase, our focus will be on crafting the primary block diagram, which will encompass the modules slated for implementation, along with the interconnections between each module. Subsequently, we will develop state diagrams for each individual module to provide comprehensive elucidation of their operational flow..

2.6.1 Block Diagram

Main Modules:

MainAlgorithm:

- Attributes: pVal, threshold.
- Signals: in > getPressureVal(int pVal), out > highPressureFlag().

AlarmMonitor:

- Attributes: alarmDuration, aTimer.
- Signals: in > highPressureFlag(), out > alarmON(), out > alarmOFF().

Other Modules:

PressureSensorDriver:

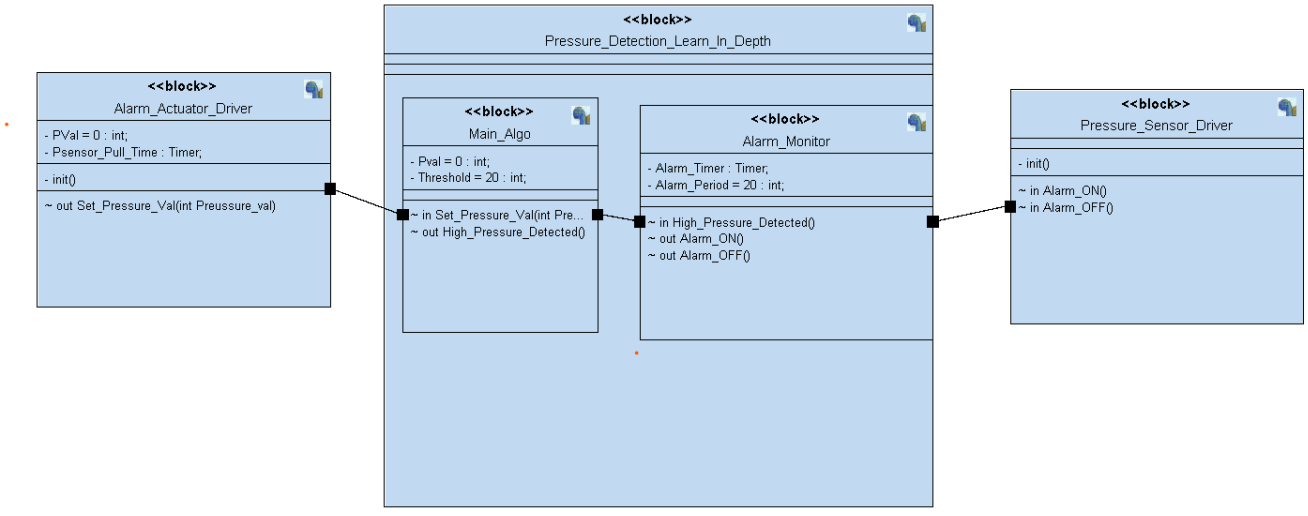
- Attributes: pVal, pTimer.
- Methods: pDriver_init().

Signals: out > getPressureVal(int pVal).

AlarmActuatorDriver:

- Methods: aDriver_init().
- Signals: in > alarmON(), in > alarmOFF().

Figure 8 - Block Diagram



2.6.2 State Machines Diagram

2.6.2.1 PressureSensorDriver State Machine

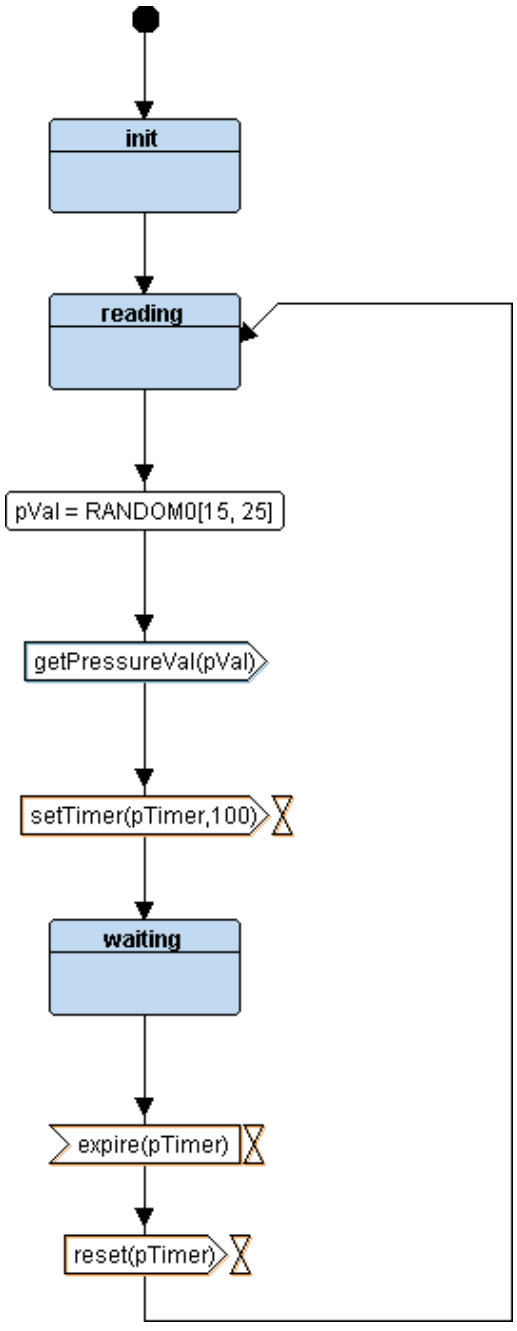


Figure 9 - PressureSensorDriver State Machine

2.6.2.2 ComparingAlgorithm State Machine

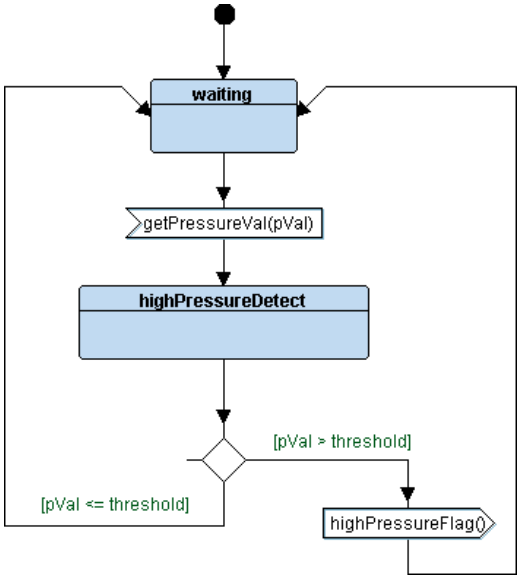


Figure 10 - ComparingAlgorithm State Machine

2.6.2.3 AlarmMonitor State Machine

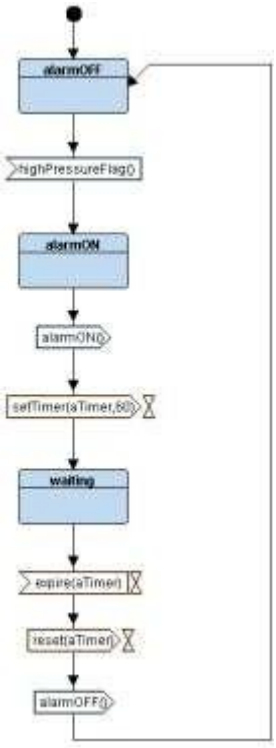


Figure 11 - AlarmMonitor State Machine

2.6.2.4 AlarmActuatorDriver State Machine

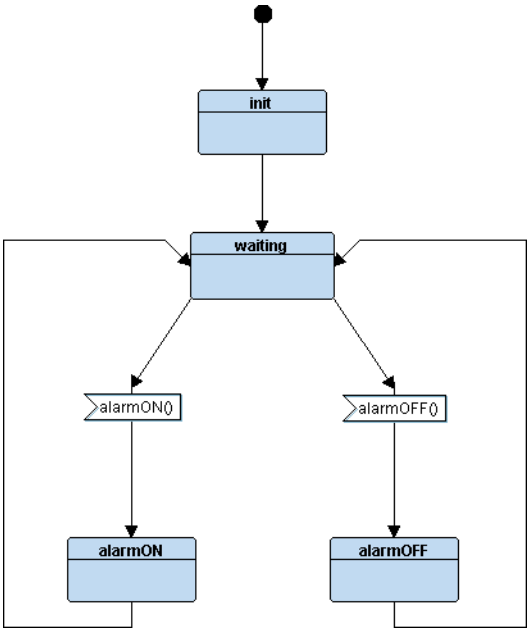


Figure 12 - AlarmActuatorDriver State Machine

2.6.3 State Machine Simulation

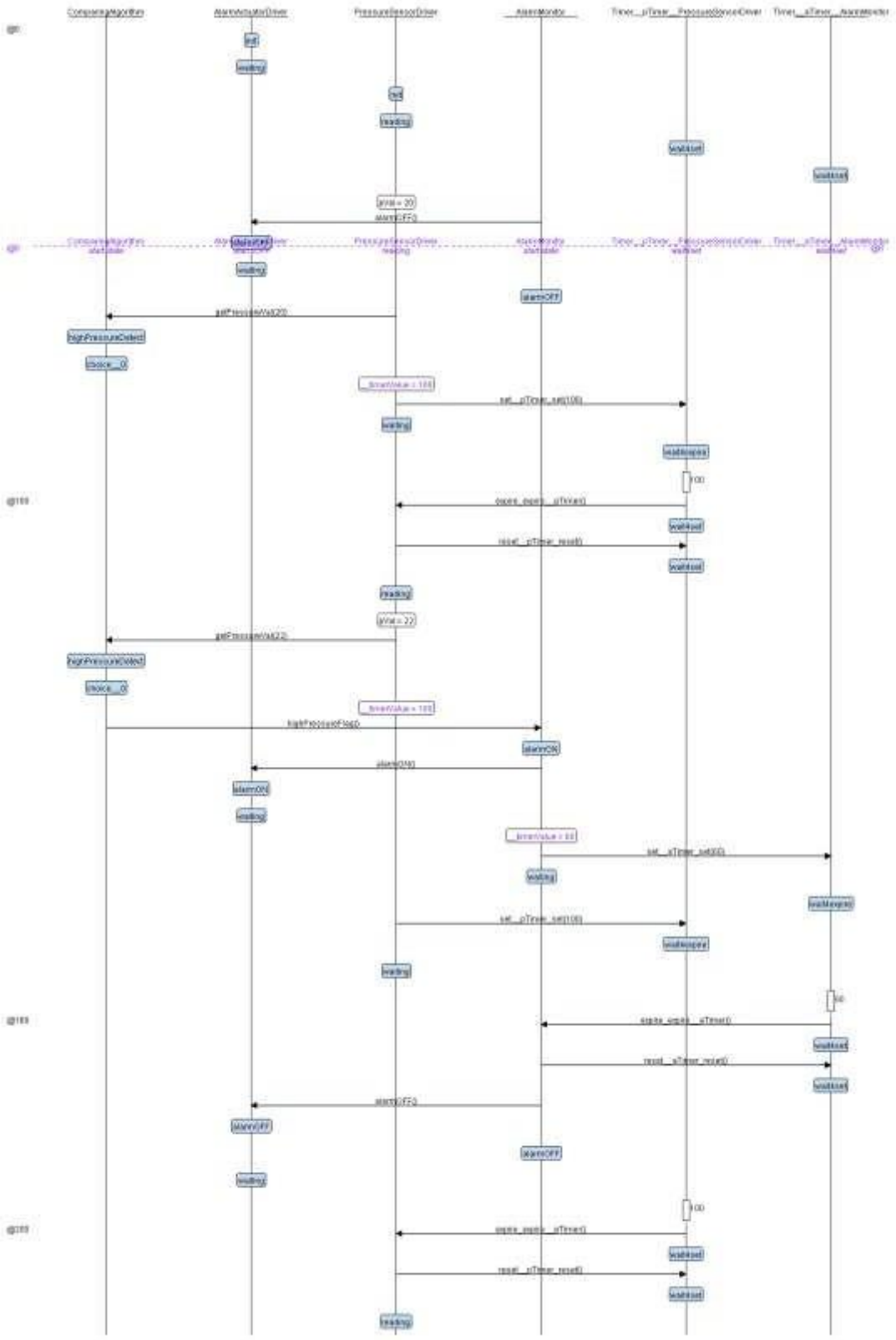


Figure 13 - State Machine Simulation

3 IMPLEMENTATION

In this section, we will implement the software system according to the designed state machines considering their flow.

3.1 Source Code

3.1.1 PressureSensor.h

```h
//Protection from multiple declarations
#ifndef PS_H_
#define PS_H_

#include "state.h"
#include "stdio.h"
#include "stdlib.h"

//Define states
enum {
    PS_Waiting,
    PS_Reading
} PS_state_id;

//Declares state functions CA
STATE_define(PS_waiting);
STATE_define(PS_reading);

void PS_init(void);

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

#endif
```

Figure 14 - PressureSensor.h

3.1.2 PressureSensor.c

```c
#include "PressureSensor.h"

//Variables
int PSD_pVal = 0 ;
const int PSD_Timer = 1000 ;
int CA_threshold = 4 ;


void PS_init (void) {

    //Initialize the Pressure Sensor
}

//STATE Pointer to function
void (*PS_state)();

STATE_define(PS_reading)
{
    //State name
    PS_state_id = PS_Reading ;
    PSD_pVal = getPressureVal();
    setPressureVal(PSD_pVal);
    PS_state = STATE(PS_waiting);


}

STATE_define(PS_waiting)
{
    PS_state_id = PS_Waiting ;
    Delay(PSD_Timer);
    PS_state = STATE(PS_reading);

}
```

Figure 15 - PressureSensor.c

3.1.3 Algorithm.h

```c
//Protection from multiple declarations
#ifndef ALG_H_
#define ALG_H_

#include "state.h"
#include "stdio.h"
#include "stdlib.h"
//Define states
enum {
    Waiting,
    Comparing
} Al_state_id;

//Declares state functions CA
STATE_define(Al_waiting);
STATE_define(Al_comparing);

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

#endif
```

Figure 16 - Algorithm.h

3.1.4 Algorithm.c

```c
#include "Algorithm.h"

//Variables
int PVal = 0 ;
const int PVal_threshold = 20 ;

//STATE Pointer to function
void (*Al_state)();

void setPressureVal(int s){
    PVal = s;
}

STATE_define(Al_waiting)
{
    Al_state_id = Waiting ;
    Al_state = STATE(Al_comparing);

}

STATE_define(Al_comparing)
{

    Al_state_id = Comparing ;
    if(PVal > PVal_threshold) {
        alarmON();
    }
    else{alarmOFF();}
    Al_state = STATE(Al_waiting);
}
```

Figure 17 - Algorithm.c

3.1.5 AlarmMonitor.h

```h
//Protection from multiple declarations
#ifndef AM_H_
#define AM_H_

#include "state.h"
#include "stdio.h"
#include "stdlib.h"
//Define states
enum {
    AM_AlarmOFF,
    AM_AlarmON,
    AM_Waiting
} AM_state_id;

//Declares state functions DC
STATE_define(AM_alarmON);
STATE_define(AM_alarmOFF);
STATE_define(AM_waiting);


void AM_init(void);

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

#endif
```

Figure 18 - AlarmMonitor.h

3.1.6 AlarmMonitor.c

```c
#include "AlarmMonitor.h"
//Variables
const int ATimer = 60000;
int AlarmFlag = 0 ;
void alarmON (void) {
    AlarmFlag = 1;
    AM_state = STATE(AM_alarmON);
}
void alarmOFF (void) {
    AlarmFlag = 0;
    AM_state = STATE(AM_alarmOFF);
}
//STATE Pointer to function
void (*AM_state)();

void AM_init(void){
    //init the Alarm Monitor required registers
}
STATE_define(AM_alarmOFF)
{
    //State name
    AM_state_id = AM_AlarmOFF ;

}
STATE_define(AM_alarmON)
{
    //State name
    AM_state_id = AM_AlarmON ;
    AM_state = STATE(AM_waiting);
}
STATE_define(AM_waiting)
{
    //State name
    AM_state_id = AM_Waiting ;
    Delay(ATimer);
    AM_state = STATE(AM_alarmOFF);
}
```

Figure 19 - AlarmMonitor.c

3.1.7 AlarmAcutator.h

```h
//Protection from multiple declarations
#ifndef AA_H_
#define AA_H_

#include "state.h"
#include "stdio.h"
#include "stdlib.h"
//Define states
enum {
    AA_AlarmOFF,
    AA_AlarmON,
    AAD_Waiting
} AA_state_id;

//Declares state functions CA
STATE_define(AA_waiting);
STATE_define(AA_alarmOFF);
STATE_define(AA_alarmON);

#define turnON  0
#define turnOFF 1

void AA_init();

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

#endif
```

Figure 20 - AlarmAcutator.h

3.1.8 AlarmAcutator.c

```c
#include "AlarmActuator.h"

void AA_init()
{
    //initialize the Alarm Actuator
}
//STATE Pointer to function
void (*AA_state)();

STATE_define(AA_waiting)
{
    //State name
    AA_state_id = AAD_Waiting ;
    (AlarmFlag == 1)?( AA_state = STATE(AA_alarmON)):( AA_state = STATE(AA_alarmOFF));
}
STATE_define(AA_alarmON)
{
    //State name
    AA_state_id = AA_AlarmON ;
    Set_Alarm_actuator(turnON);
    AA_state = STATE(AA_waiting);


}
STATE_define(AA_alarmOFF)
{
    //State name
    AA_state_id = AA_AlarmOFF ;
    Set_Alarm_actuator(turnOFF);
    AA_state = STATE(AA_waiting);

}
```

Figure 21 - AlarmAcutator.c

3.1.9 driver.h

```
#ifndef DRIV_H_
#define DRIV_H_

#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 ();

#endif
```

Figure 22 - driver.h

3.1.10 driver.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 23 - driver.c

3.1.11 main.c

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

#include "driver.h"
#include "AlarmActuator.h"
#include "AlarmMonitor.h"
#include "Algorithm.h"
#include "PressureSensor.h"

void setup()
{
    //initialzations
    GPIO_INITIALIZATION();
    PS_init();
    AA_init();
    AM_init();
    PS_state = STATE(PS_reading);
    Al_state = STATE(Al_waiting);
    AM_state = STATE(AM_alarmOFF);
    AA_state = STATE(AA_waiting);
}

int main (){
    setup();
    while (1)
    {
        PS_state();
        Al_state();
        AM_state();
        AA_state();
    }

}
```

Figure 24 - main.c

3.1.12 startup.c

```c
//Startup.c
//Eng. Omar Shawky

#include <stdint.h>

extern void main(void);
extern unsigned int _STACK_TOP ;

void 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_Handler()__attribute__((weak ,alias("Default_Handler")));
void Usage_fault_Handler()__attribute__((weak ,alias("Default_Handler")));

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_Handler,
    (uint32_t)  &Usage_fault_Handler
};

extern uint32_t _E_TEXT ;
extern uint32_t _S_DATA ;
extern uint32_t _E_DATA ;
extern uint32_t _S_BSS ;
extern uint32_t _E_BSS ;
```

```c
void Reset_Handler()
{
    //copy data from flash to ram
    uint32_t DATA_SIZE = (unsigned char*)&_E_DATA  -  (unsigned char*)&_S_DATA ;
    unsigned char * P_src  = (unsigned char*)&_E_TEXT;
    unsigned char * P_dest = (unsigned char*)&_S_DATA;
    for(int i = 0 ; i < DATA_SIZE ; i++){
        *((unsigned char*)P_dest++) = *((unsigned char*)P_src++);
    }

    //initialize .bss section in sram with zeros
    uint32_t BSS_SIZE = (unsigned char*)&_E_BSS  -  (unsigned char*)&_S_BSS ;
    P_dest = (unsigned char*)&_S_BSS;
    for(int i = 0 ; i < BSS_SIZE ; i++){
        *((unsigned char*)P_dest++) = (unsigned char) 0;
    }
    //Jump to main
    main();
}

void Default_Handler(){
    Reset_Handler();
}
```

Figure 25 - startup.c

3.1.13 linker_script.ld

```
/* Linker script Cortex M3
Eng. Omar Shawky */

MEMORY
{
    flash(RX): ORIGIN = 0x08000000 , LENGTH = 128K  /*Define flash memory from address 0x08000000 with length 128K READ ONLY*/
    sram(RWX): ORIGIN = 0x20000000 , LENGTH = 20K   /*Define sram  memory from address 0x20000000 with length 20k  READ WRITE*/
}

SECTIONS
{
    .text :{    /*output a section called .text*/
            *(.vectors*)    /*take input called .vectors section found in any .o file */
            *(.rodata)      /*take input called .rodata  section found in any .o file */
            *(.text*)       /*take input called .text    section found in any .o file */
            _E_TEXT = . ;   /*save the current location to the variable name _E_TEXT  */
    } > flash               /*put the output section in the memory defined as flash in both runtime and loadtime*/

    .data :{    /*output a section called .data*/
            _S_DATA = . ;   /*save the current location to the variable name _S_DATA  */
            *(.data)        /*take input called .data section found in any .o file */
            . = ALIGN(4);   /*Align the counter to a 4 byte address*/
            _E_DATA = . ;   /*save the current location to the variable name _E_DATA  */
    } > sram AT> flash      /*put the output section in the memory defined as flash loadtime then copy it to flash at runtime*/

    .bss  :{    /*output a section called .text*/
            _S_BSS = . ;    /*save the current location to the variable name _S_BSS  */
            *(.bss)         /*take input called .bss section found in any .o file */
            _E_BSS = . ;    /*save the current location to the variable name _E_BSS  */
            . = ALIGN(4);   /*Align the counter to a 4 byte address*/
            . = . + 0x1000; /*Increment the counter by the stack size*/
            _STACK_TOP = .;
    } > sram                /*put the output section in the memory defined as sram in both runtime and loadtime*/
}
```

Figure 26 - linker_script.ld

3.1.14 Makefile

```
#Copyright Omar Shawky
#-------------------------------------------------------#
#Define target name
Target_Name = learn_in_depth_cortex_m3
#Define the cross-toolchain
CC=arm-none-eabi-
#Define the flags for the cross-toolchain (Debugging info enabled , processor specified)
CSTD = -std=c99
CFLAGS = -gdwarf-2 -mcpu=cortex-m3 -mthumb
#Define the includes
INCS = -I.
#Define the libraries
LIBS =
#-------------------------------------------------------#
#Get all .c files inside the folder
SRC = $(wildcard *.c)
#Get all .s files inside the folder
ASM = $(wildcard *.s)
#Get all .c and convert it to .o
OBJ = $(SRC:.c=.o)
#Get all .S and convert it to .o
OBJASM = $(ASM:.s=.o)
#-------------------------------------------------------#
#Build all
all: $(Target_Name).bin
    @echo "==========BUILD IS DONE=========="

#Assemble .o file from .s files
%.o: %.s
    $(CC)as.exe  $(CFLAGS)  $< -o $@

#Compile .o file from .c files
%.o: %.c
    $(CC)gcc.exe -c  $(CFLAGS) $(CSTD) $(INCS) $< -o $@
```

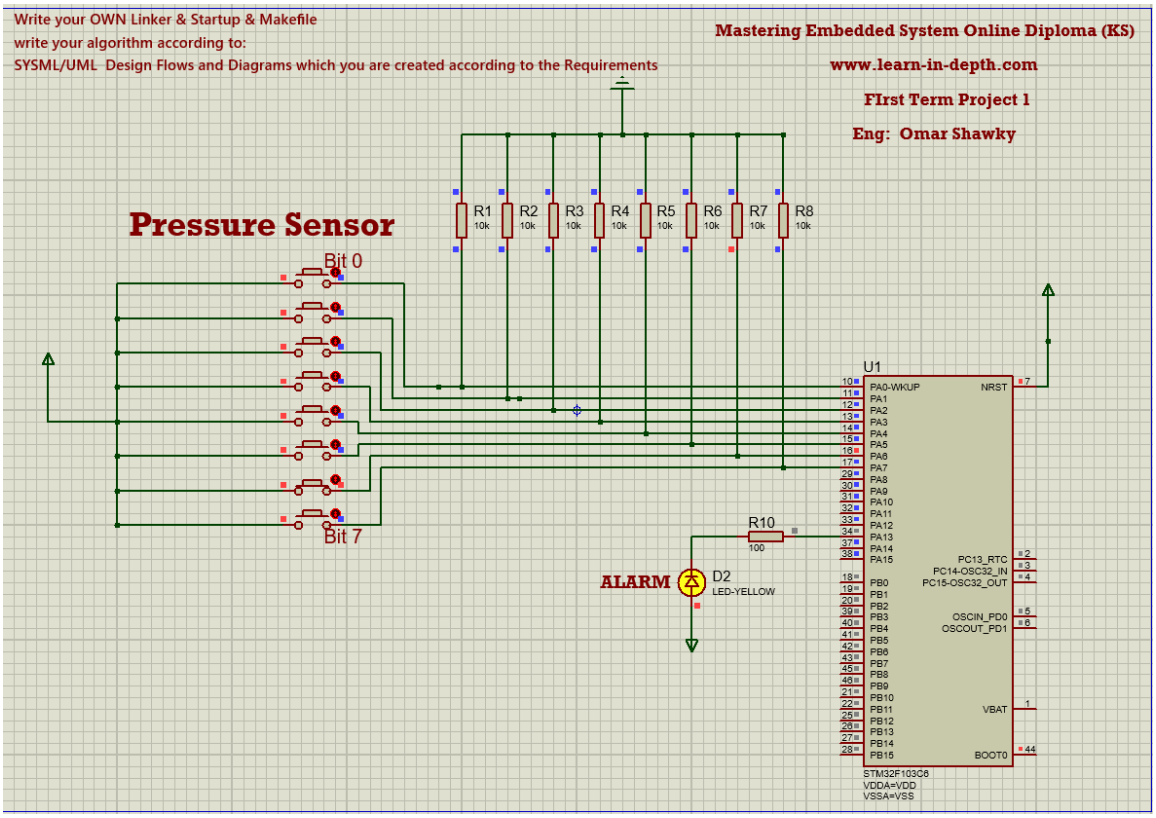
```
#Link all files with the linkerscript
$(Target_Name).elf:  $(OBJ) $(OBJASM)
    $(CC)ld.exe  -T  linkerscript.ld $(OBJ) $(OBJASM) $(LIBS) -o $@ -Map=Map_file

#Get the binary image of the .elf output
$(Target_Name).bin: $(Target_Name).elf
    $(CC)objcopy.exe  -O binary  $< $(Target_Name).bin
#-------------------------------------------------------#
#Clean the previous build
clean_all:
    rm *.o
    rm *.elf
    rm *.bin
clean:
    rm *.elf
    rm *.bin
```

Figure 27 - Makefile

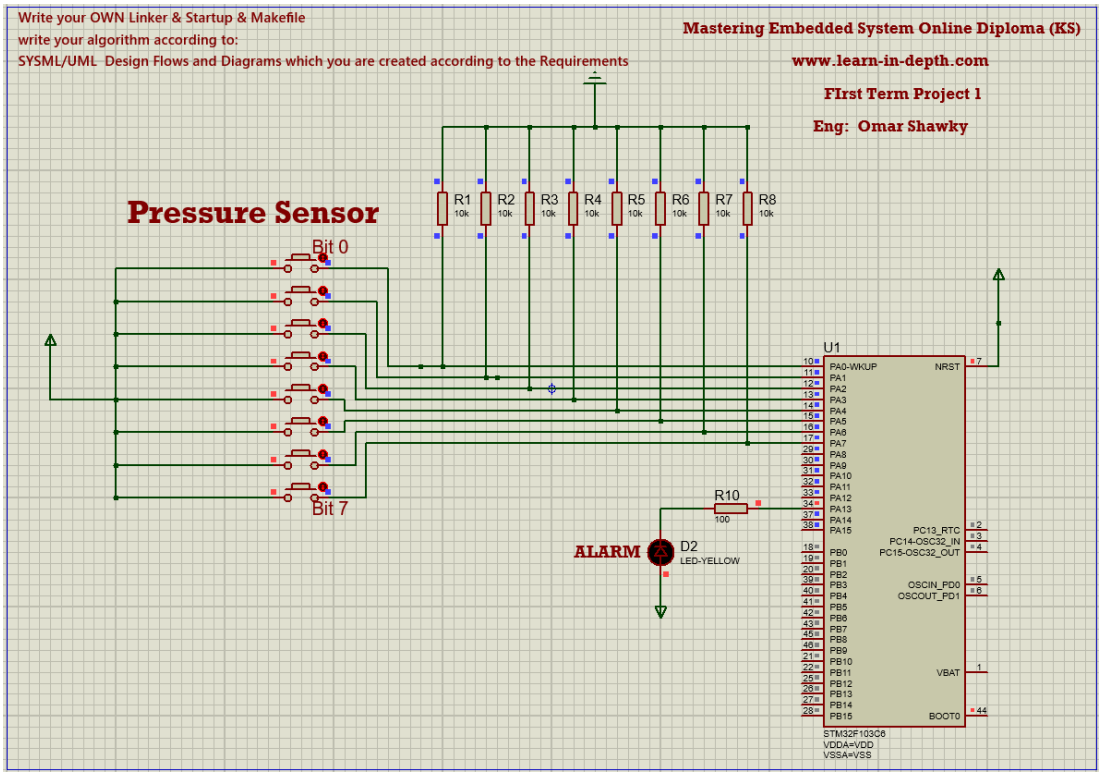
3.2 Output

3.2.1 Simulation Screenshots

Pressure is above threshold.



Pressure is below threshold.



3.2.2 Output Mapfile

```
Allocating common symbols
Common symbol       size              file

AA_state            0x4               AlarmActuator.o
AM_state_id         0x1               AlarmMonitor.o
AM_state            0x4               AlarmMonitor.o
AA_state_id         0x1               AlarmActuator.o
Al_state            0x4               Algorithm.o
PS_state            0x4               PressureSensor.o
Al_state_id         0x1               Algorithm.o
PS_state_id         0x1               main.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      0x544
 *(.vectors*)
 .vectors       0x08000000       0x1c startup.o
                0x08000000                vectors
 *(.rodata)
 .rodata        0x0800001c        0x4 AlarmMonitor.o
                0x0800001c                ATimer
 .rodata        0x08000020        0x4 Algorithm.o
                0x08000020                PVal_threshold
 .rodata        0x08000024        0x4 PressureSensor.o
                0x08000024                PSD_Timer
 *(.text*)
 .text          0x08000028       0xb8 AlarmActuator.o
                0x08000028                AA_init
```

```
*(.text*)
.text          0x08000028       0xb8 AlarmActuator.o
               0x08000028                AA_init
               0x08000034                ST_AA_waiting
               0x08000080                ST_AA_alarmON
               0x080000b0                ST_AA_alarmOFF
.text          0x080000e0       0xe0 AlarmMonitor.o
               0x080000e0                alarmON
               0x0800010c                alarmOFF
               0x08000138                AM_init
               0x08000144                ST_AM_alarmOFF
               0x0800015c                ST_AM_alarmON
               0x08000188                ST_AM_waiting
.text          0x080001c0       0x94 Algorithm.o
               0x080001c0                setPressureVal
               0x080001e0                ST_Al_waiting
               0x0800020c                ST_Al_comparing
.text          0x08000254      0x10c driver.o
               0x08000254                Delay
               0x08000278                getPressureVal
               0x08000290                Set_Alarm_actuator
               0x080002e0                GPIO_INITIALIZATION
.text          0x08000360       0x9c main.o
               0x08000360                setup
               0x080003c0                main
.text          0x080003fc       0x8c PressureSensor.o
               0x080003fc                PS_init
               0x08000408                ST_PS_reading
               0x08000450                ST_PS_waiting
.text          0x08000488       0xbc startup.o
               0x08000488                Reset_Handler
               0x08000538                Usage_fault_Handler
               0x08000538                MM_fault_Handler
               0x08000538                Default_Handler
               0x08000538                BUS_fault_Handler
               0x08000538                H_fault_Handler
```

```
                0x08000538                NMI_Handler
                0x08000544                _E_TEXT = .

.glue_7         0x08000544        0x0
 .glue_7        0x00000000        0x0 linker stubs

.glue_7t        0x08000544        0x0
 .glue_7t       0x00000000        0x0 linker stubs

.vfp11_veneer   0x08000544        0x0
 .vfp11_veneer  0x00000000        0x0 linker stubs

.v4_bx          0x08000544        0x0
 .v4_bx         0x00000000        0x0 linker stubs

.iplt           0x08000544        0x0
 .iplt          0x00000000        0x0 AlarmActuator.o

.rel.dyn        0x08000544        0x0
 .rel.iplt      0x00000000        0x0 AlarmActuator.o

.data           0x20000000        0x4 load address 0x08000544
                0x20000000                _S_DATA = .
 *(.data)
 .data          0x20000000        0x0 AlarmActuator.o
 .data          0x20000000        0x0 AlarmMonitor.o
 .data          0x20000000        0x0 Algorithm.o
 .data          0x20000000        0x0 driver.o
 .data          0x20000000        0x0 main.o
 .data          0x20000000        0x4 PressureSensor.o
                0x20000000                CA_threshold
 .data          0x20000004        0x0 startup.o
                0x20000004                . = ALIGN (0x4)
                0x20000004                _E_DATA = .

.igot.plt       0x20000004        0x0 load address 0x08000548
 .igot.plt      0x00000000        0x0 AlarmActuator.o
```

```
.igot.plt       0x00000000        0x0 AlarmActuator.o

.bss            0x20000004     0x1028 load address 0x08000548
                0x20000004                _S_BSS = .
 *(.bss)
 .bss           0x20000004        0x0 AlarmActuator.o
 .bss           0x20000004        0x4 AlarmMonitor.o
                0x20000004                AlarmFlag
 .bss           0x20000008        0x4 Algorithm.o
                0x20000008                PVal
 .bss           0x2000000c        0x0 driver.o
 .bss           0x2000000c        0x0 main.o
 .bss           0x2000000c        0x4 PressureSensor.o
                0x2000000c                PSD_pVal
 .bss           0x20000010        0x0 startup.o
                0x20000010                _E_BSS = .
                0x20000010                . = ALIGN (0x4)
                0x20001010                . = (. + 0x1000)
 *fill*         0x20000010     0x1000
                0x20001010                _STACK_TOP = .
 COMMON         0x20001010        0x5 AlarmActuator.o
                0x20001010                AA_state
                0x20001014                AA_state_id
 *fill*         0x20001015        0x3
 COMMON         0x20001018        0x8 AlarmMonitor.o
                0x20001018                AM_state_id
                0x2000101c                AM_state
 COMMON         0x20001020        0x5 Algorithm.o
                0x20001020                Al_state
                0x20001024                Al_state_id
 COMMON         0x20001025        0x1 main.o
                0x20001025                PS_state_id
 *fill*         0x20001026        0x2
 COMMON         0x20001028        0x4 PressureSensor.o
                0x20001028                PS_state
LOAD AlarmActuator.o
LOAD AlarmMonitor.o
```

```
LOAD AlarmMonitor.o
LOAD Algorithm.o
LOAD driver.o
LOAD main.o
LOAD PressureSensor.o
LOAD startup.o
OUTPUT(learn_in_depth_cortex_m3.elf elf32-littlearm)

.debug_info     0x00000000      0x92b
 .debug_info    0x00000000      0x127 AlarmActuator.o
 .debug_info    0x00000127      0x170 AlarmMonitor.o
 .debug_info    0x00000297      0x13a Algorithm.o
 .debug_info    0x000003d1      0x103 driver.o
 .debug_info    0x000004d4      0x191 main.o
 .debug_info    0x00000665      0x13b PressureSensor.o
 .debug_info    0x000007a0      0x18b startup.o

.debug_abbrev   0x00000000      0x4c7
 .debug_abbrev  0x00000000       0xa3 AlarmActuator.o
 .debug_abbrev  0x000000a3       0xb1 AlarmMonitor.o
 .debug_abbrev  0x00000154       0xc2 Algorithm.o
 .debug_abbrev  0x00000216       0x9d driver.o
 .debug_abbrev  0x000002b3       0xa5 main.o
 .debug_abbrev  0x00000358       0x9b PressureSensor.o
 .debug_abbrev  0x000003f3       0xd4 startup.o

.debug_loc      0x00000000      0x450
 .debug_loc     0x00000000       0xb0 AlarmActuator.o
 .debug_loc     0x000000b0      0x108 AlarmMonitor.o
 .debug_loc     0x000001b8       0x90 Algorithm.o
 .debug_loc     0x00000248       0xc8 driver.o
 .debug_loc     0x00000310       0x58 main.o
 .debug_loc     0x00000368       0x84 PressureSensor.o
 .debug_loc     0x000003ec       0x64 startup.o
```