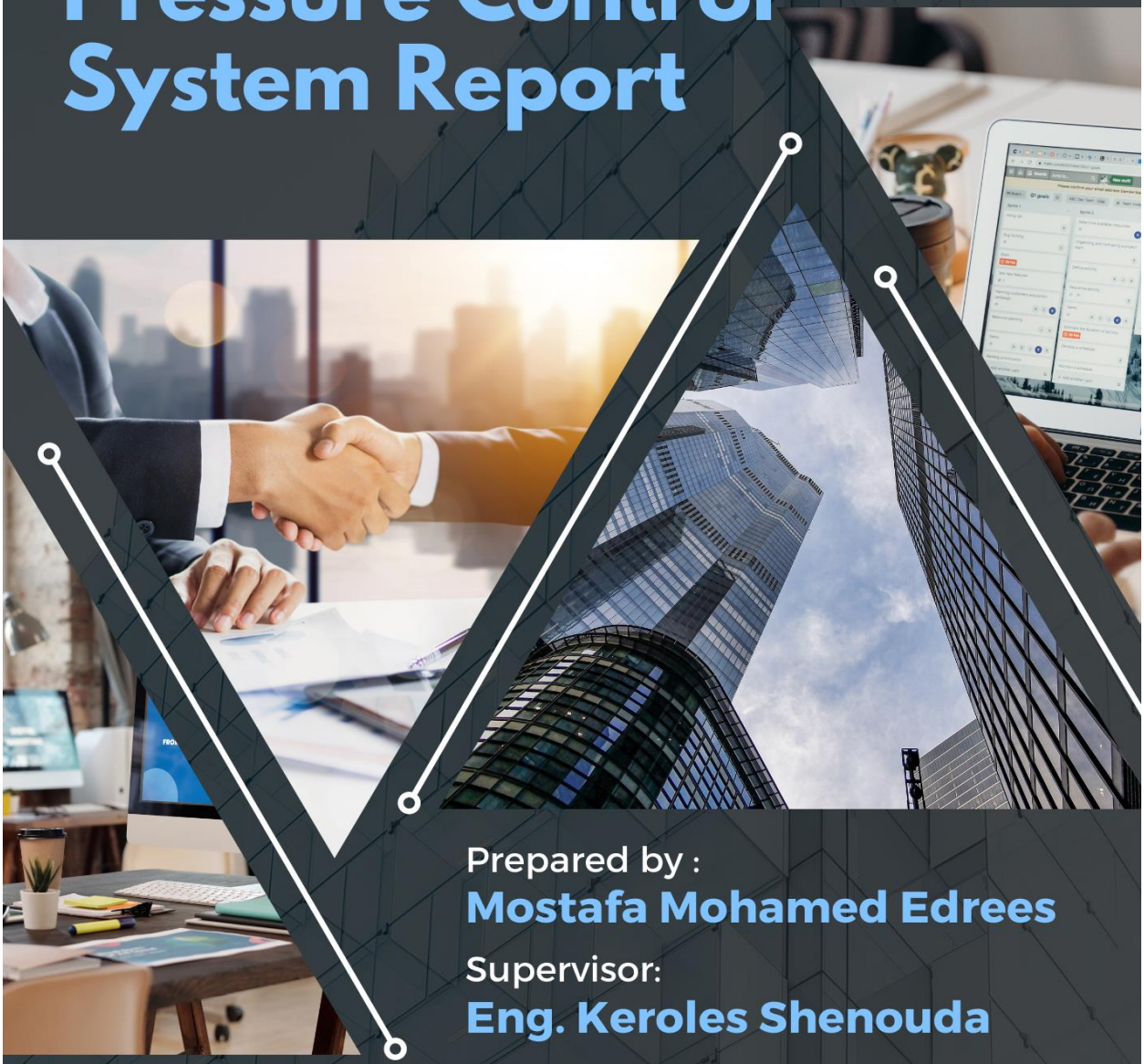
Learn-In_Depth

2023

Pressure Control System Report



Prepared by :

Mostafa Mohamed Edrees

Supervisor:

Eng. Keroles Shenouda

My Page

TIMELINE

System Architecting Sequence

1

CASE STUDY

<u>Pressure Control System</u>

Informs the crew of a cabin with an alarm when the pressure exceeds 20 bars in the cabin

2

METHOD

V MODEL - SDLC

Software Development Life Cycle

3

REQUIREMENT

- Pressure Sensor
- Alarm (LED)
- FLASH Memory

4

SPACE EXPLORATION

We choose STM32F103C6 after space exploration.

5

SYSTEM ANALYSIS

- Use Case Diagram
- Activity Diagram
- Sequence Diagram

6

SYSTEM DESIGN

Divide the project into some blocks.

Case Study:

Pressure Detection System

A client expects you to deliver the software of the following system:

Specification (from the client):

- A pressure controller informs the crew of a cabin with an alarm when the pressure exceeds 20 bars in the cabin.
- The alarm duration equals 60 seconds.
- Keeps track of the measured values.

Assumptions:

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

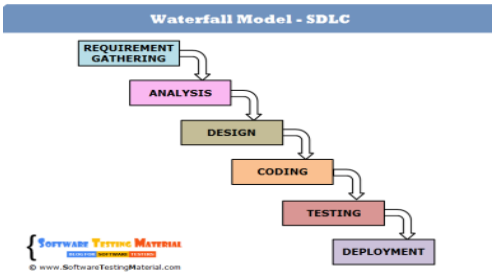
Version 1.0:

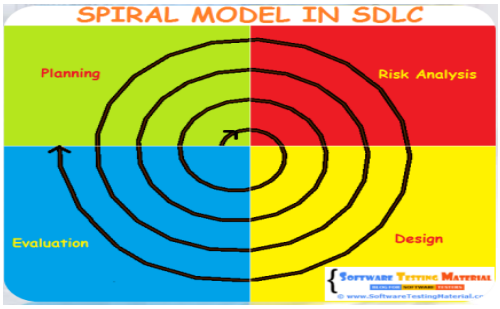
The keep track of measured value option is not modeled in the first version of the design.

Method:

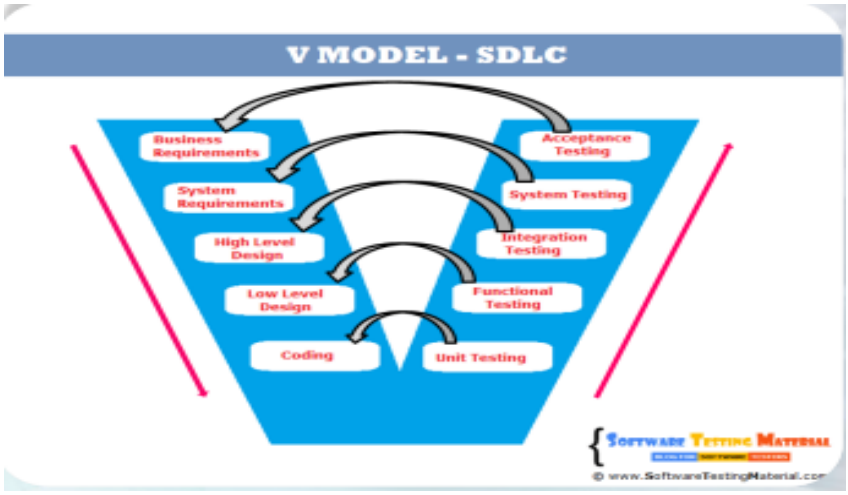
There are different methods like as:

- V Model-SDLC
- Waterfall Model-SDLC
- SPIRAL Model-SDLC



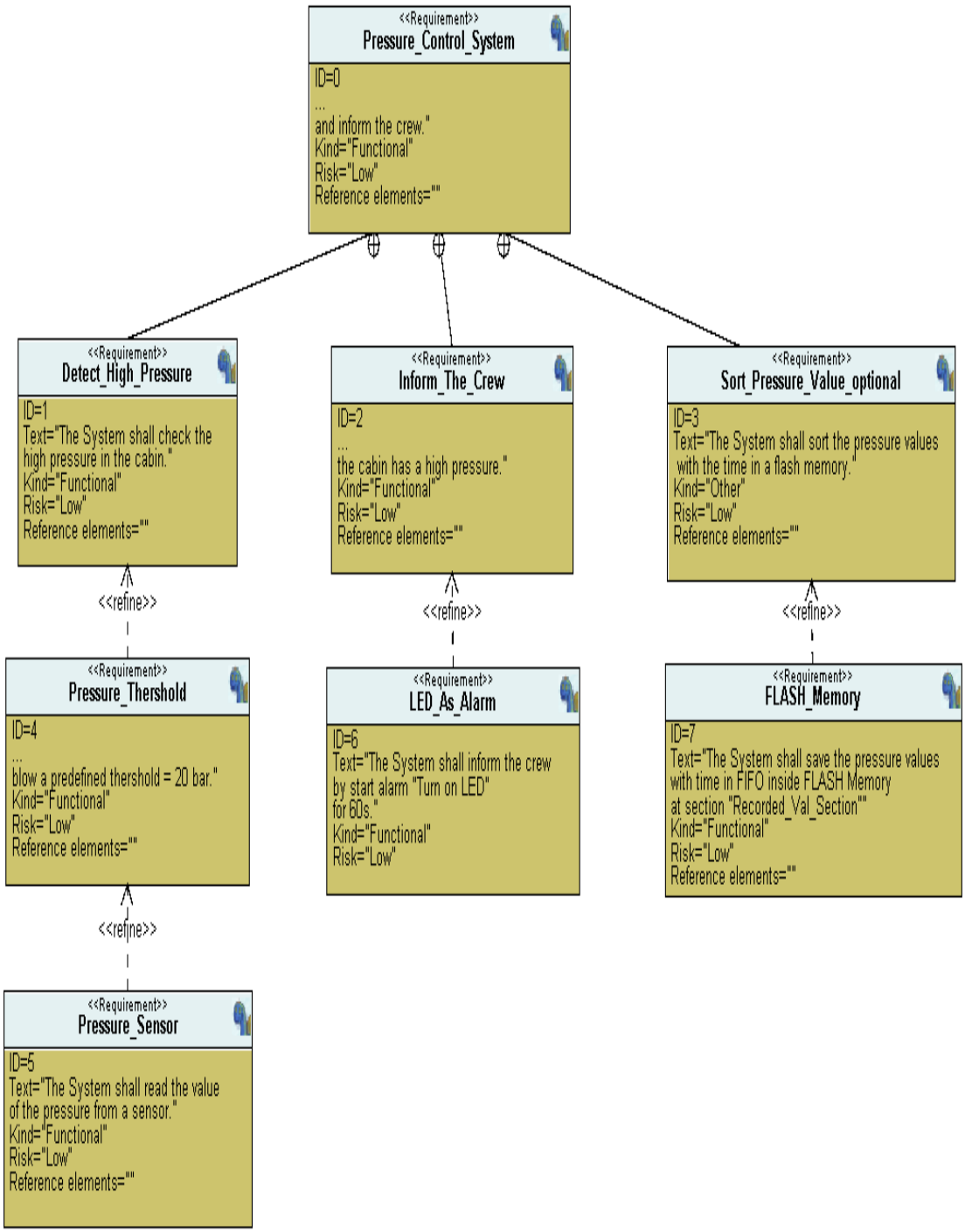


We will use **V Model-SDLC**



SDLC is Software Development Life Cycle.

Requirement Diagram:

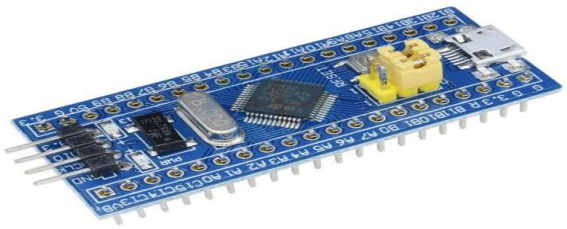


Space Exploration:

After space exploration we will use:

Microcontroller: STM32F10C6

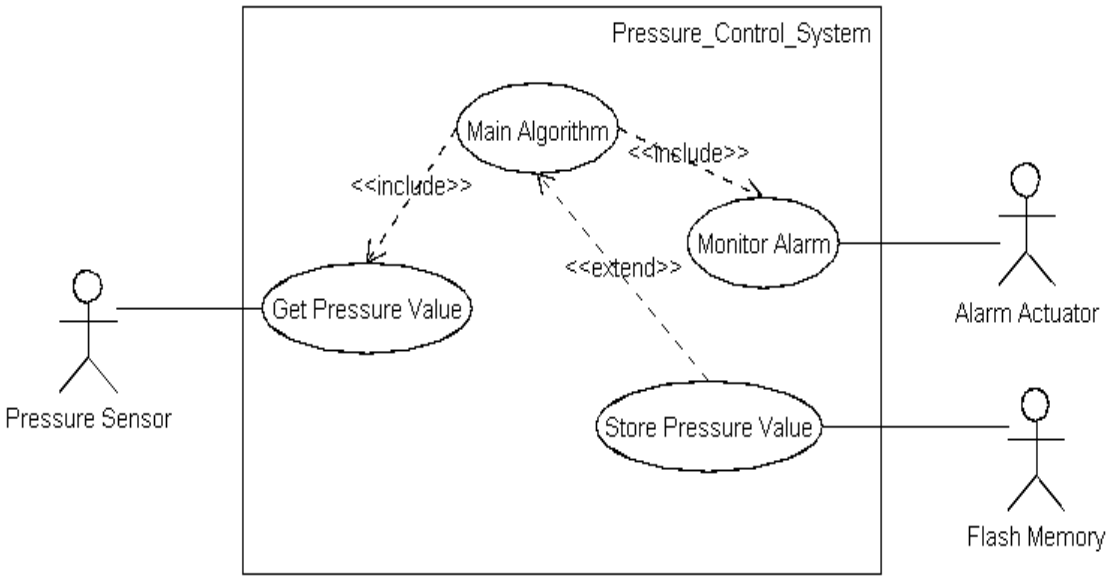
Processor: Arm-Cortex-M3



System Analysis:

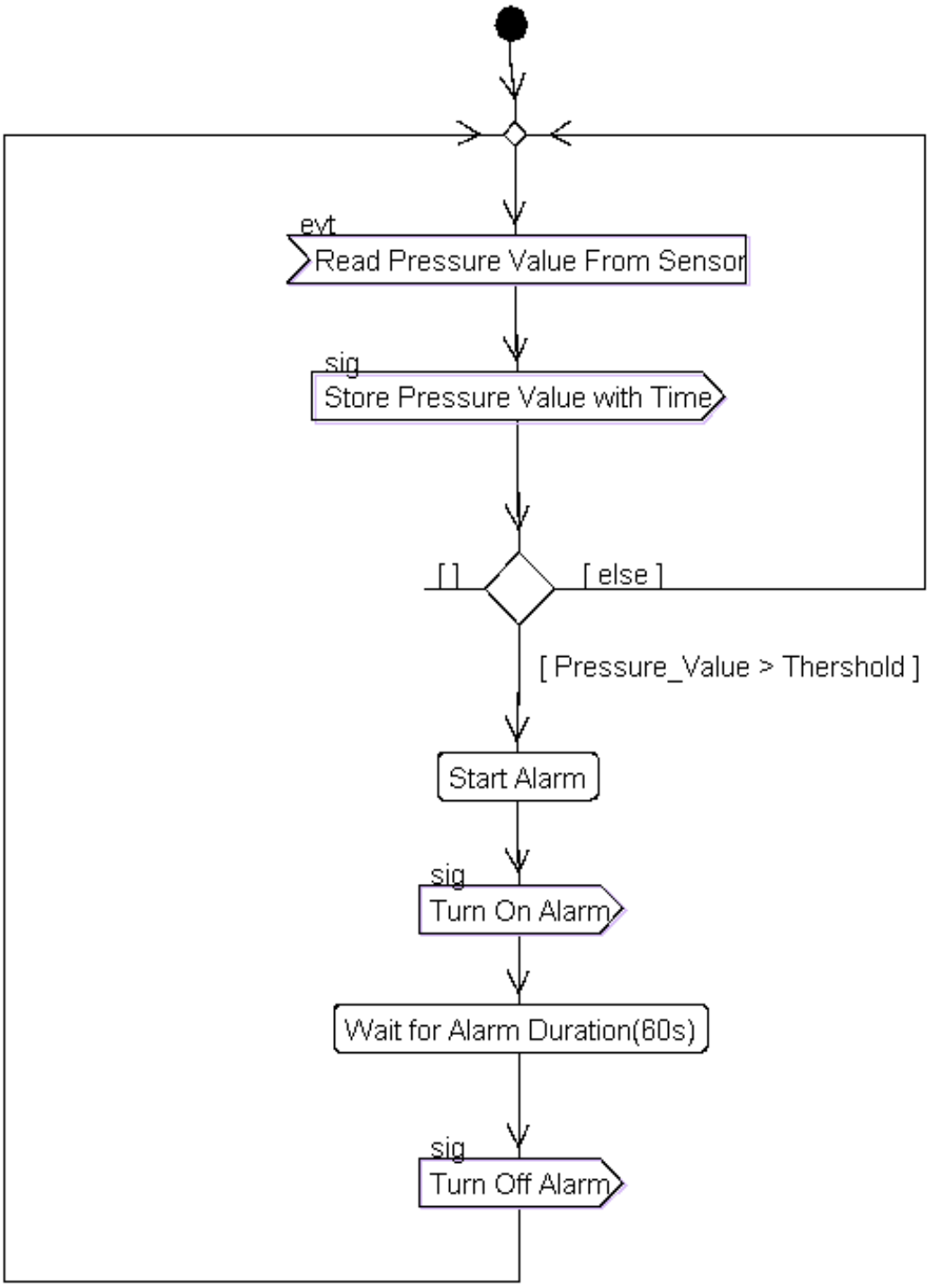
Use Case Diagram:

It will obtain System boundary and Main functions.



Activity Diagram:

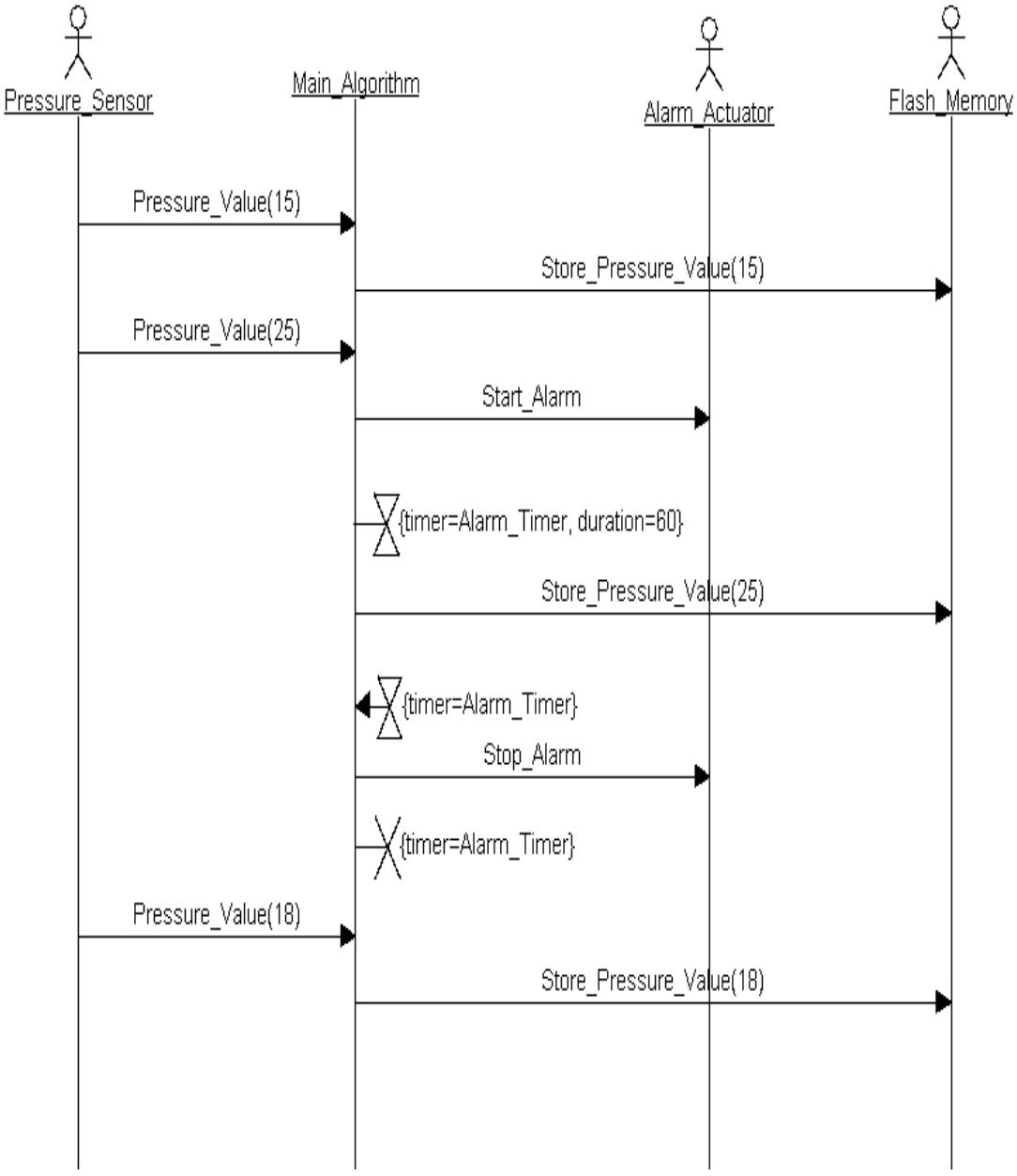
It’s used to describe the workflow behavior of a system.



Sequence Diagram:

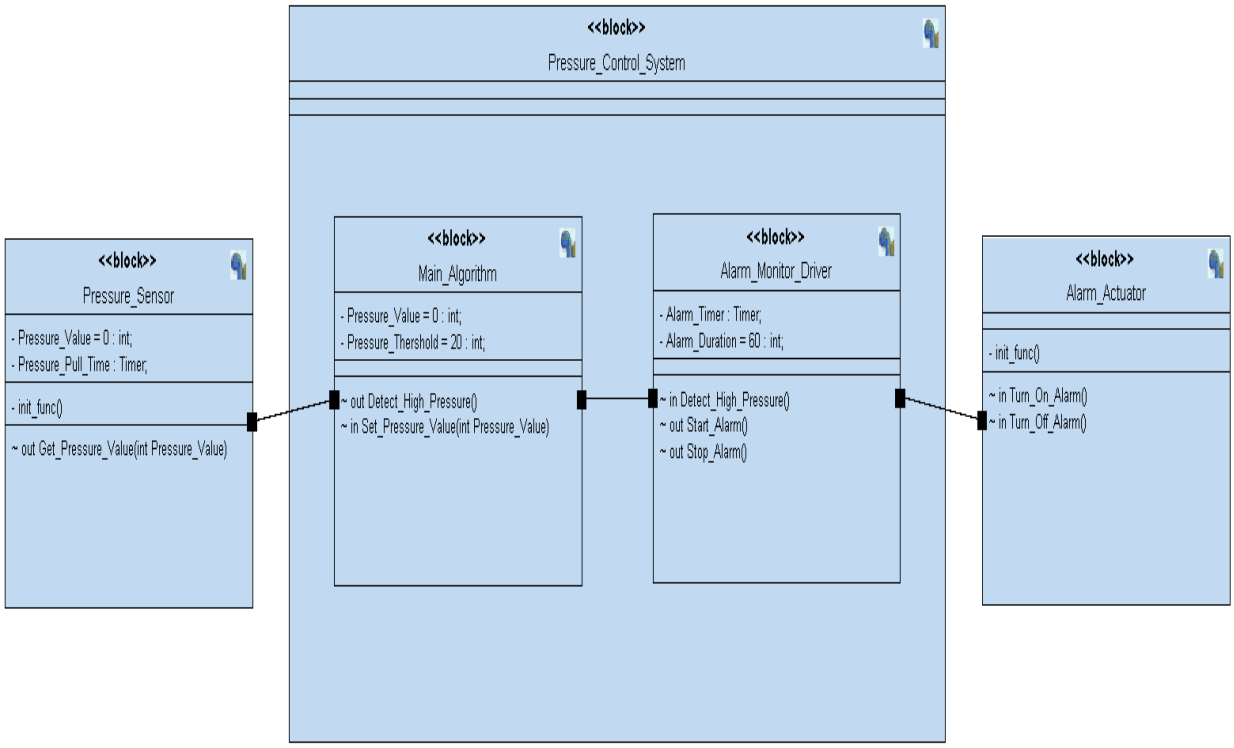
A sequence diagram is:

- An interaction diagram that details how operations are carried out.
- What messages are sent and when.
- Is organized according to time.



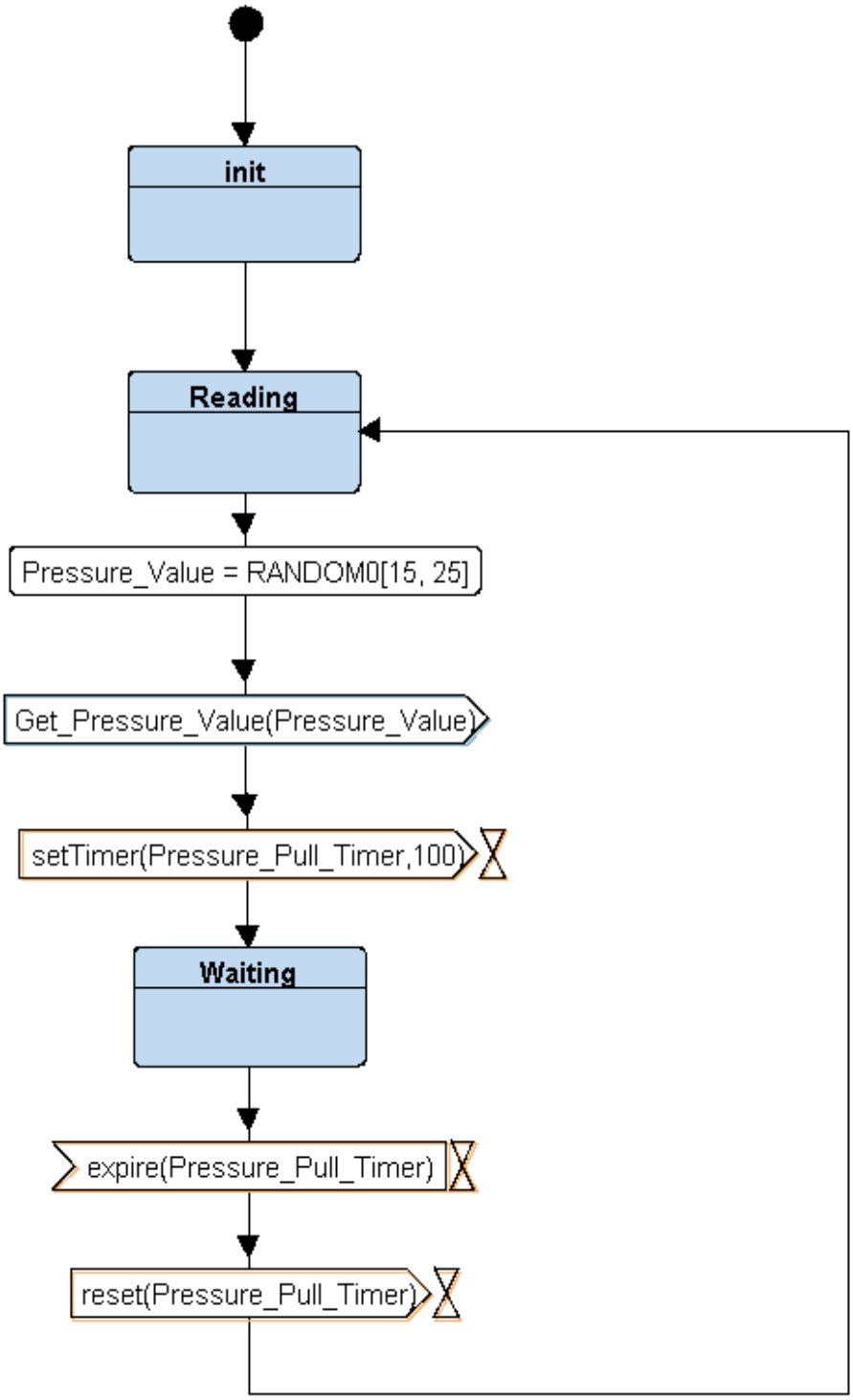
System Design:

Block Diagram:

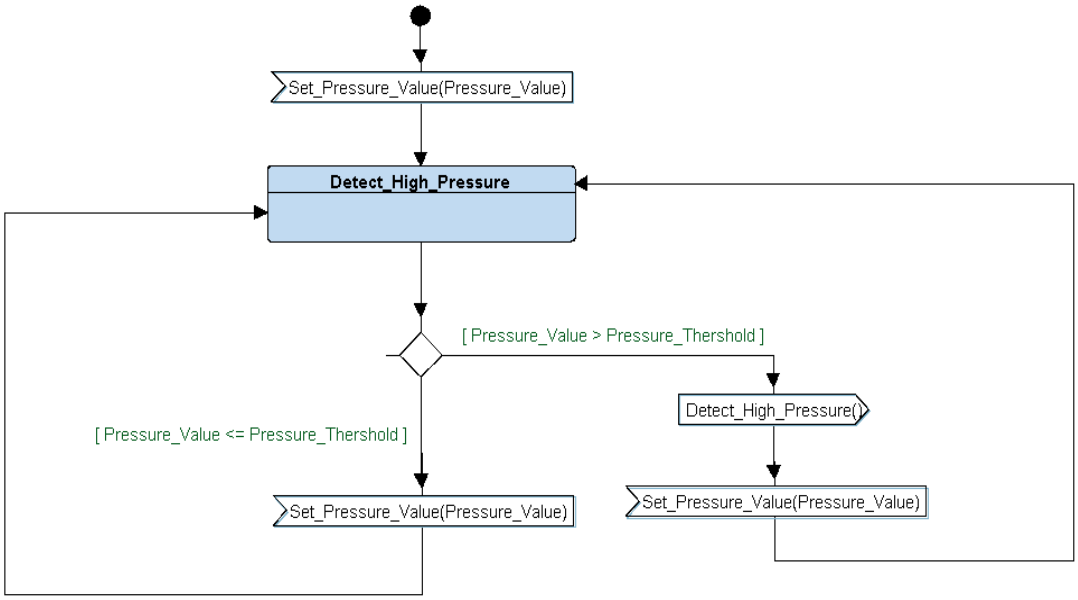


State Machine for each block:

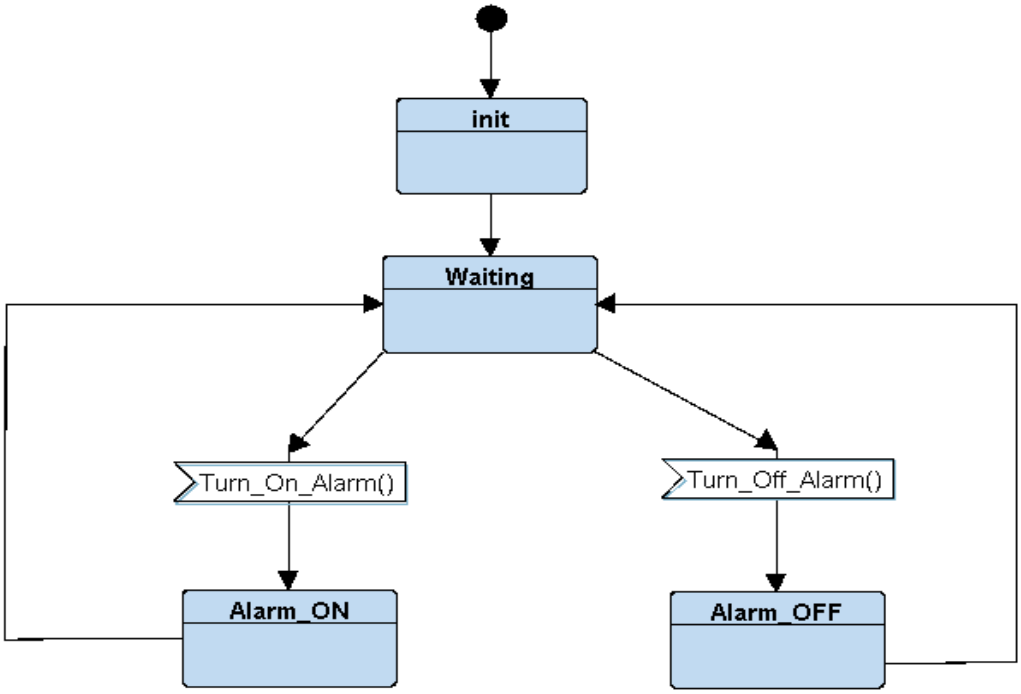
Pressure Sensor



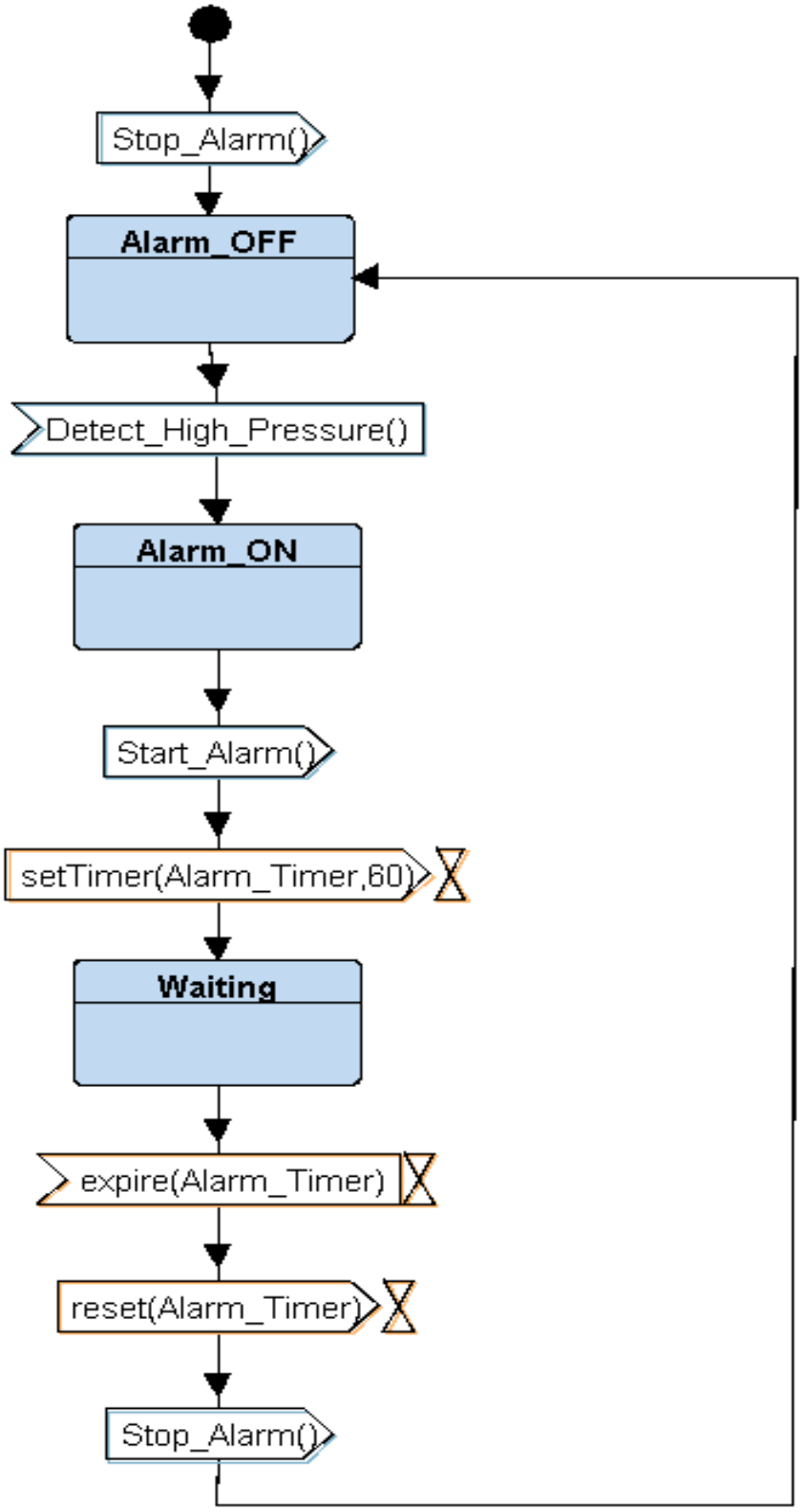
Main Algorithm



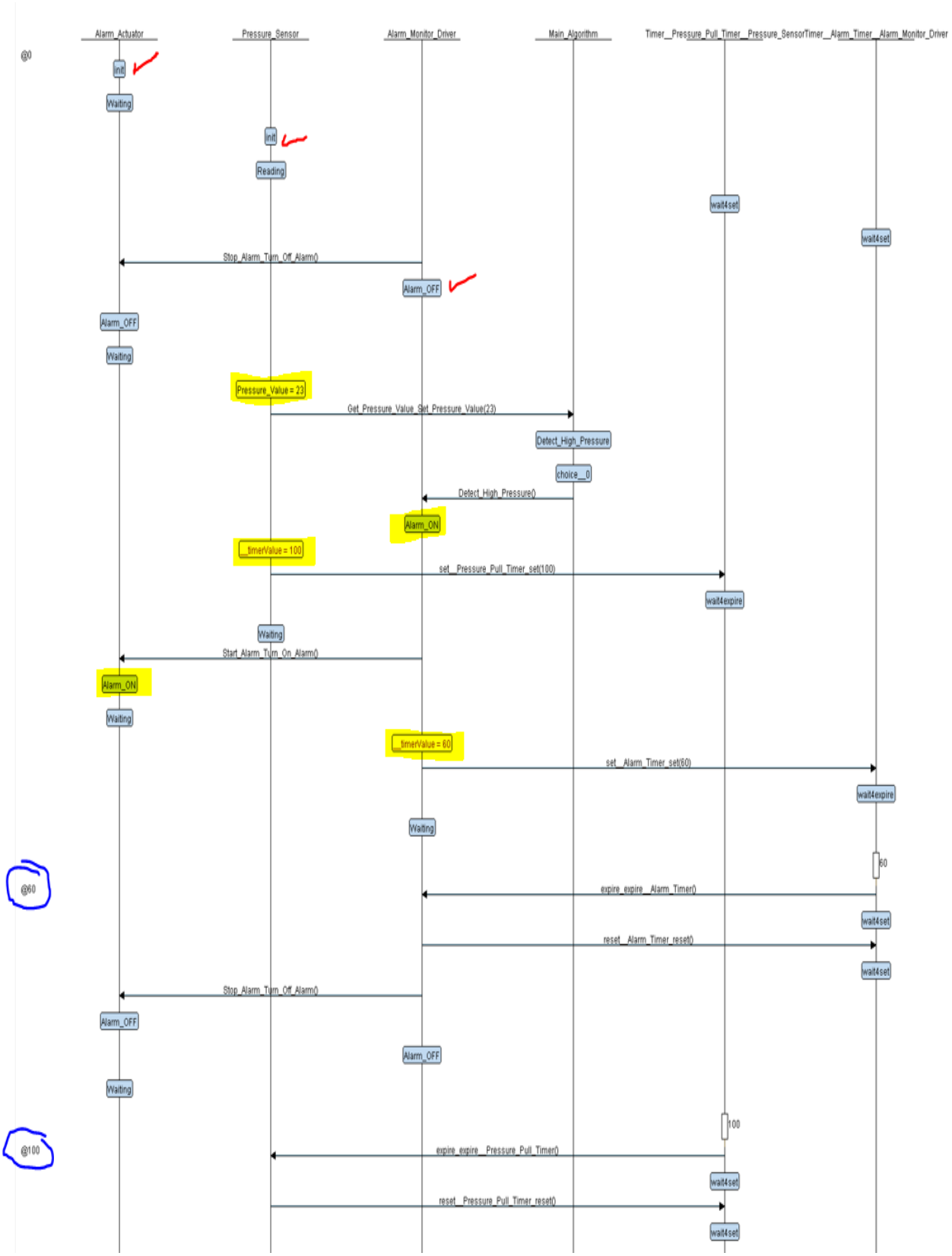
Alarm Actuator



Alarm Monitor Driver



Interactive Simulation State when pressure is bigger than 20:

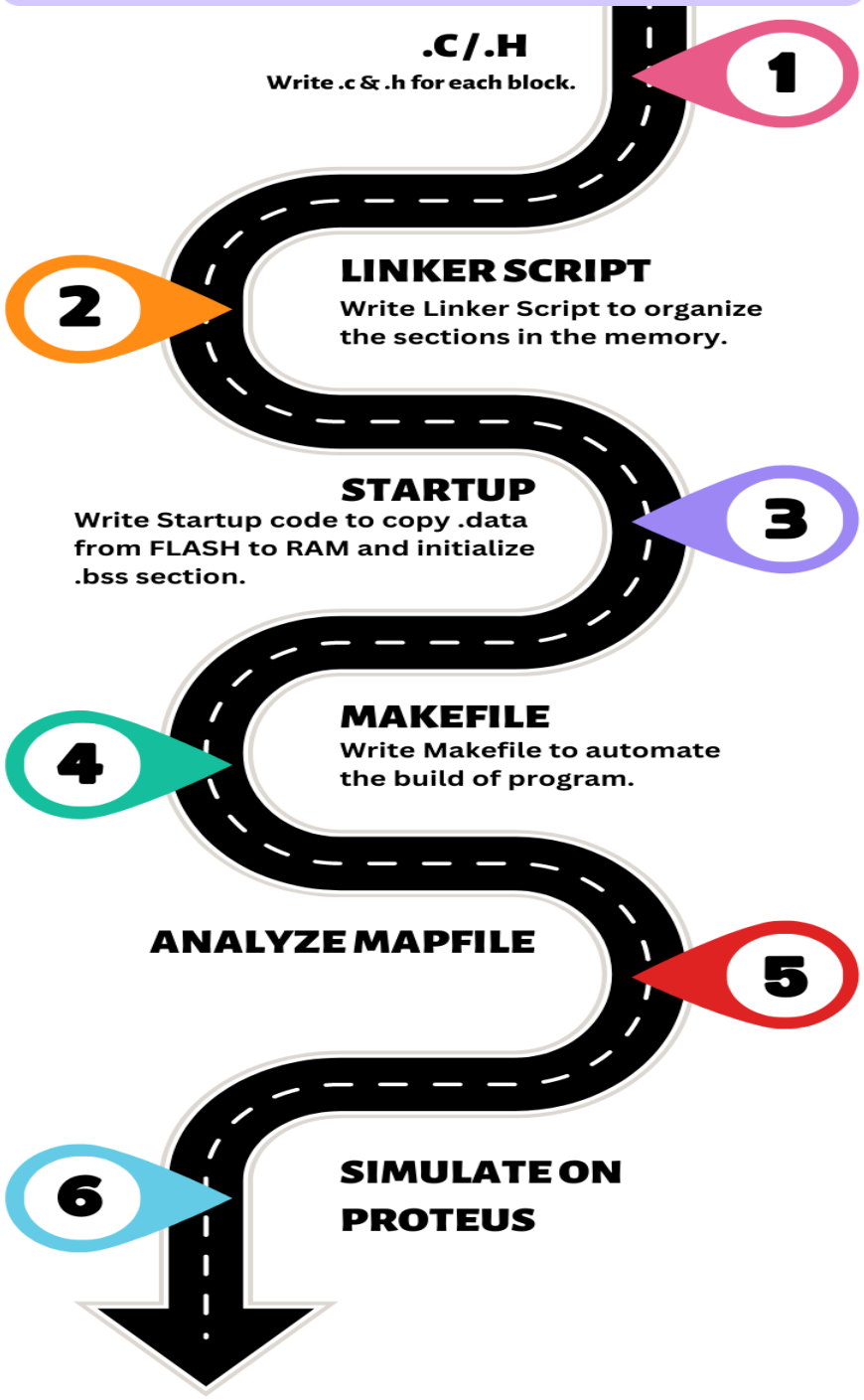


Interactive Simulation State when pressure is smaller than 20:

The Alarm is not turn on and we don’t use the timer of it because the pressure sensor read values less than 20 bar.

TIMELINE

Implementation Sequence



.c / .h for each block:

Pressure Sensor.h

```c
#ifndef PRESSURE_SENSOR_H_
#define PRESSURE_SENSOR_H_

#include "state.h"
#include "driver.h"

//Pressure Sensor states
enum
{
    Pressure_Sensor_Reading,
    Pressure_Sensor_Waiting
}Pressure_Sensor_State_Id;

//states of pressure
enum
{
    Low_Pressure,
    High_Pressure
}Pressure_State;

//generate state functions
STATE_Define(Pressure_Sensor_Reading);
STATE_Define(Pressure_Sensor_Waiting);

//pointer to state function
extern void (*Pressure_Sensor_State)();

//function to initialize Pressure Sensor Driv
void Pressure_Sensor_init();

//function to Get Pressure Value
void Get_Pressure_Value();


#endif /* PRESSURE_SENSOR_H_ */
```

Pressure Sensor.c

```c
 * @file           : Pressure Sensor.c
#include "Pressure_Sensor.h"

//variables
int Pressure_Value = 0;
int Pressure_Pull_Timer = 100;

//pointer to state function
void (*Pressure_Sensor_State)();

 * @function_name   : Pressure_Sensor_init
void Pressure_Sensor_init()
{
    //initialize Pressure Sensor Drivers
    GPIO_INITIALIZATION();
}

 * @function_name   : Get_Pressure_Value
void Get_Pressure_Value()
{
    Pressure_Value = getPressureVal();
}


 * @function_name   : ST_Pressure_Sensor_Reading
STATE_Define(Pressure_Sensor_Reading)
{
    //State Name
    Pressure_Sensor_State_Id = Pressure_Sensor_Reading;

    //State Action
    Get_Pressure_Value();
    Set_Pressure_Value(Pressure_Value);

    //delay
    Delay(Pressure_Pull_Timer);
}

 * @function_name   : ST_Pressure_Sensor_Waiting
STATE_Define(Pressure_Sensor_Waiting)
{
    //State Name
    Pressure_Sensor_State_Id = Pressure_Sensor_Waiting;

    //go from Waiting to Reading state
    Pressure_Sensor_State = STATE(Pressure_Sensor_Reading);
}
```

Main Algorithm.h

```h
#ifndef MAIN_ALGORITHM_
#define MAIN_ALGORITHM_

#include "Alarm_Actuator.h"
#include "state.h"
#include "driver.h"
#include "Pressure_Sensor.h"

//Main Algorithm states
enum
{
    Detect_High_Pressure_state,
}Main_Algorithm_State_Id;

//generate state functions
STATE_Define(Detect_High_Pressure);

//pointer to state function
extern void (*Main_Algorithm_State)();


#endif  /* MAIN_ALGORITHM_ */
```

Main Algorithm.c

```c
 * @file            : Main_Algorithm.c

#include "Main_Algorithm.h"

//variables
int Pressure_Value;
int Pressure_Threshold = 20;

//pointer to state function
void (*Main_Algorithm_State)();

 * @function_name   : Set_Pressure_Value
void Set_Pressure_Value(int Pressure_Val)
{
    Pressure_Value = Pressure_Val;

    //go from Reading to Waiting state
    Pressure_Sensor_State = STATE(Pressure_Sensor_Waiting);
}

 * @function_name   : ST_Pressure_Sensor_Waiting
STATE_Define(Detect_High_Pressure)
{
    //State Name
    Main_Algorithm_State_Id = Detect_High_Pressure_state;

    //State Action
    if(Pressure_Value <= Pressure_Threshold)
    {
        Detect_High_Pressure(Low_Pressure);
    }
    else if(Pressure_Value > Pressure_Threshold)
    {
        Detect_High_Pressure(High_Pressure);
    }


    //go back to Detect High Pressure state.
    Main_Algorithm_State = STATE(Detect_High_Pressure);
}
```

Alarm Actuator.h

```c
 * @file            : Alarm Actuator.h

#ifndef ALARM_ACTUATOR_H_
#define ALARM_ACTUATOR_H_

#include "state.h"
#include "driver.h"
#include "Pressure_Sensor.h"

// Alarm Actuator states
enum
{
    Alarm_Actuator_ON,
    Alarm_Actuator_OFF,
    Alarm_Actuator_Waiting
}Alarm_Actuator_State_Id;

//states of Alarm
enum
{
    ON,
    OFF
}Alarm_State;

//generate state functions
STATE_Define(Alarm_Actuator_ON);
STATE_Define(Alarm_Actuator_OFF);
STATE_Define(Alarm_Actuator_Waiting);

//pointer to state function
extern void (*Alarm_Actuator_State)();

//function to initialize Alarm Actuator Driver
void Alarm_Actuator_init();


#endif /* ALARM_ACTUATOR_H_ */
```

Alarm Actuator.c

```c
 * @file            : Alarm Actuator.c

#include "Alarm_Actuator.h"

//variables
int Alarm_Duration = 10000;

//pointer to state function
void (*Alarm_Actuator_State)();

 * @function_name   : Alarm_Actuator_init
void Alarm_Actuator_init()
{
    //initialize Alarm Actuator Drivers
    //Alarm_INITIALIZATION();

    //Turn off alarm
    Set_Alarm_actuator(OFF);

    //go to Alarm OFF state
    Alarm_Actuator_State = STATE(Alarm_Actuator_OFF);
}
```

```
 * @function_name   : Detect_High_Pressure
void Detect_High_Pressure(int High_OR_Low)
{
    if(High_OR_Low == High_Pressure)
    {
        Set_Alarm_actuator(ON);

        //delay for alarm timer
        Delay(Alarm_Duration);

        //go to Alarm OFF state
        Alarm_Actuator_State = STATE(Alarm_Actuator_Waiting);

        //go to Alarm ON state
        Alarm_Actuator_State = STATE(Alarm_Actuator_ON);
    }
    else
    {
        //Turn off alarm
        Set_Alarm_actuator(OFF);

        //go to Alarm OFF state
        Alarm_Actuator_State = STATE(Alarm_Actuator_OFF);
    }
}

 * @function_name   : ST_Alarm_Actuator_ON
STATE_Define(Alarm_Actuator_ON)
{
    //State Name
    Alarm_Actuator_State_Id = Alarm_Actuator_ON;

    //waiting Alarm Duration
    Alarm_Actuator_State = STATE(Alarm_Actuator_Waiting);
    Alarm_Actuator_State();

    //Turn off alarm
    Set_Alarm_actuator(OFF);
    Delay(1000);

    //go to Alarm OFF state
    Alarm_Actuator_State = STATE(Alarm_Actuator_OFF);
}

 * @function_name   : ST_Alarm_Actuator_OFF
STATE_Define(Alarm_Actuator_OFF)
{
    //State Name
    Alarm_Actuator_State_Id = Alarm_Actuator_OFF;
}

 * @function_name   : ST_Alarm_Actuator_Waiting
STATE_Define(Alarm_Actuator_Waiting)
{
    //State Name
    Alarm_Actuator_State_Id = Alarm_Actuator_Waiting;
}
```

state.h

```
 * @file            : state.h

#ifndef STATE_H_
#define STATE_H_

#include "stdlib.h"

//state functions generated automatically
#define STATE_Define(_StateFun_)    void ST_##_StateFun_()
#define STATE(_StateFun_)           ST_##_StateFun_


//states connections
void Set_Pressure_Value(int Pressure_Val);
void Detect_High_Pressure(int High_OR_Low);


#endif /* STATE_H_ */
```

driver.h

```
 * @file            : driver.h

#ifndef DRIVER_H_
#define DRIVER_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 /* DRIVER_H_ */
```

driver.c

```
 * @file           : driver.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;
}
```

main.c

```c
 * @file           : main.c
#include "Alarm_Actuator.h"
#include "state.h"
#include "driver.h"
#include "Main_Algorithm.h"
#include "Pressure_Sensor.h"

void setup()
{
    //init Drivers
    //init Interrupts
    //init HAL like as Pressure Sensor Driver & Alarm Actuator Driver
    //init Blocks
    Pressure_Sensor_init();
    Alarm_Actuator_init();

    //Set state pointer for each block
    Pressure_Sensor_State = STATE(Pressure_Sensor_Reading);
    Alarm_Actuator_State = STATE(Alarm_Actuator_OFF);
    Main_Algorithm_State = STATE(Detect_High_Pressure);
}

int main()
{
    setup();

    while(1)
    {
        //Call state for each block
        Alarm_Actuator_State();
        Pressure_Sensor_State();
        Main_Algorithm_State();

        //delay
        Delay(1000);
    }

    return 0;
}
```

Linker Script:

```
MEMORY
{
    FLASH (rx) : ORIGIN = 0x08000000, LENGTH = 128k
    SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 20k
}

SECTIONS
{
    .text :
    {
        *(.vectors*)
        *(.text)
        . = ALIGN(4) ;
        _E_text = . ;
    }> FLASH

    .data :
    {
        . = ALIGN(4) ;
        _S_data = . ;
        *(.data*)
        . = ALIGN(4) ;
        _E_data = . ;
    }> SRAM AT> FLASH

    .rodata :
    {
        . = ALIGN(4) ;
        _S_rodata = . ;
        *(.rodata*)
        . = ALIGN(4) ;
        _E_rodata = . ;
    }> FLASH

    .bss :
    {
        . = ALIGN(4) ;
        _S_bss = . ;
        *(.bss*)
        . = ALIGN(4) ;
        *(COMMON*)
        . = ALIGN(4) ;
        _E_bss = . ;
    }> SRAM

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

Startup.c

```c
extern int main(void);
extern usint32_t _stack_top;

//Vector Handler
void Reset_Handler(void);
void NMI_Handler(void) __attribute__((weak,alias("Default_Handler")));
void Hard_Fault_Handler(void) __attribute__((weak,alias("Default_Handler")));
void MM_Fault_Handler(void) __attribute__((weak,alias("Default_Handler")));

vusint32_t vectors[] __attribute__((section(".vectors"))) =
{
    (vusint32_t) &_stack_top,
    (vusint32_t) &Reset_Handler,
    (vusint32_t) &NMI_Handler,
    (usint32_t)  &Hard_Fault_Handler,
    (usint32_t)  &MM_Fault_Handler,
    (usint32_t)  &Bus_Fault_Handler,
    (usint32_t)  &Usage_Fault_Handler
};

extern usint32_t _E_text;
extern usint32_t _S_data;
extern usint32_t _E_data;
extern usint32_t _S_bss;
extern usint32_t _E_bss;

void Reset_Handler (void)
{
    int i;

    //copy .data section from flash to ram
    usint32_t DATA_size = (usint8_t *)&_E_data - (usint8_t *)&_S_data ;
    usint8_t *P_src = (usint8_t *)&_E_text ;
    usint8_t *P_dst = (usint8_t *)&_S_data ;
    for(i = 0; i < DATA_size; i++)
    {
        *((usint8_t *)P_dst++) = *((usint8_t *)P_src++);
    }

    //locate .bss section in ram and initalize it with zero
    usint32_t BSS_size = (usint8_t *)&_E_bss - (usint8_t *)&_S_bss ;
    P_dst = (usint8_t *)&_S_bss ;
    for(i = 0; i < BSS_size; i++)
    {
        *((usint8_t *)P_dst++) = (usint8_t)0;
    }

    //jump to main()
    main();
}

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

Makefile:

```
CC = arm-none-eabi-
CFLAGS = -gdwarf-2 -mcpu=cortex-m3 -mthumb
INCS = -I .
LIBS =
SRC = $(wildcard *.c)
OBJ = $(SRC:.c=.o)
As = $(wildcard *.s)
AsOBj = $(As:.s=.o)

Project_Name = Pressure_Control_System
Copyrights = Mostafa Edrees
Board = STM32F103C8T6
Arm_Processor = cortex-m3
date = 1/8/2023

all: $(Project_Name).bin
    @echo -e "\n*****************************************"
    @echo -e "\tBuild is Done"
    @echo -e "Project Name:" $(Project_Name)
    @echo -e "Copyrights:" $(Copyrights)
    @echo -e "date:" $(date)
    @echo -e "Board:" $(Board)
    @echo -e "Arm Processor:" $(Arm_Processor)
    @echo -e "*****************************************\n"


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


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

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


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

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

Build the project by using Makefile:

```
Mostafa Edrees@MostafaEdrees MINGW32 /d/Mastering Embedded System/GitHub_Repo/Mastering_Embedded_System_Online_Diploma/Unit5_First Term Projects/Pressure Cont
rol System/Project_Implementation (master)
$ mingw32-make.exe
arm-none-eabi-gcc.exe -c -I . -gdwarf-2 -mcpu=cortex-m3 -mthumb startup.c -o startup.o
arm-none-eabi-gcc.exe -c -I . -gdwarf-2 -mcpu=cortex-m3 -mthumb Main_Algorithm.c -o Main_Algorithm.o
arm-none-eabi-gcc.exe -c -I . -gdwarf-2 -mcpu=cortex-m3 -mthumb Pressure_Sensor.c -o Pressure_Sensor.o
arm-none-eabi-gcc.exe -c -I . -gdwarf-2 -mcpu=cortex-m3 -mthumb main.c -o main.o
arm-none-eabi-gcc.exe -c -I . -gdwarf-2 -mcpu=cortex-m3 -mthumb Alarm_Actuator.c -o Alarm_Actuator.o
arm-none-eabi-gcc.exe -c -I . -gdwarf-2 -mcpu=cortex-m3 -mthumb driver.c -o driver.o
arm-none-eabi-ld.exe -T linker_script.ld  startup.o Main_Algorithm.o Pressure_Sensor.o main.o Alarm_Actuator.o driver.o  -Map=Map_file.map -o Pressure_Control
_System.elf
arm-none-eabi-objcopy.exe -O binary Pressure_Control_System.elf Pressure_Control_System.bin

****************************************
       Build is Done
Project Name: Pressure_Control_System
Copyrights: Mostafa Edrees
date: 1/8/2023
Board: STM32F103C8T6
Arm Processor: cortex-m3
****************************************
```

Link of GitHub Repository:

GitHub

Mapfile:

```
Memory Configuration

Name             Origin             Length             Attributes
FLASH            0x08000000         0x00020000         xr
SRAM             0x20000000         0x00005000         xrw
*default*        0x00000000         0xffffffff

Linker script and memory map


.text           0x08000000      0x48c
 *(.vectors*)
 .vectors       0x08000000       0x14 startup.o
                0x08000000                vectors
 *(.text)
 .text          0x08000014       0xbc startup.o
                0x08000014                Reset_Handler
                0x080000c4                MM_Fault_Handler
                0x080000c4                Default_Handler
                0x080000c4                Hard_Fault_Handler
                0x080000c4                NMI_Handler
 .text          0x080000d0       0x98 Main_Algorithm.o
                0x080000d0                Set_Pressure_Value
                0x08000100                ST_Detect_High_Pressure
 .text          0x08000168       0x88 Pressure_Sensor.o
                0x08000168                Pressure_Sensor_init
                0x08000174                Get_Pressure_Value
                0x0800018c                ST_Pressure_Sensor_Reading
                0x080001c4                ST_Pressure_Sensor_Waiting
 .text          0x080001f0       0x7c main.o
                0x080001f0                setup
                0x08000234                main
 .text          0x0800026c      0x114 Alarm_Actuator.o
                0x0800026c                Alarm_Actuator_init
                0x0800028c                Detect_High_Pressure
                0x080002fc                ST_Alarm_Actuator_ON
                0x08000350                ST_Alarm_Actuator_OFF
                0x08000368                ST_Alarm_Actuator_Waiting
 .text          0x08000380      0x10c driver.o
                0x08000380                Delay
                0x080003a4                getPressureVal
                0x080003bc                Set_Alarm_actuator
                0x0800040c                GPIO_INITIALIZATION
                0x0800048c                . = ALIGN (0x4)
                0x0800048c                _E_text = .
```

```
.data           0x20000000        0xc load address 0x0800048c
                0x20000000                . = ALIGN (0x4)
                0x20000000                _S_data = .
 *(.data*)
 .data          0x20000000        0x0 startup.o
 .data          0x20000000        0x4 Main_Algorithm.o
                0x20000000                Pressure_Threshold
 .data          0x20000004        0x4 Pressure_Sensor.o
                0x20000004                Pressure_Pull_Timer
 .data          0x20000008        0x0 main.o
 .data          0x20000008        0x4 Alarm_Actuator.o
                0x20000008                Alarm_Duration
 .data          0x2000000c        0x0 driver.o
                0x2000000c                . = ALIGN (0x4)
                0x2000000c                _E_data = .

.igot.plt       0x2000000c        0x0 load address 0x08000498
 .igot.plt      0x00000000        0x0 startup.o

.rodata         0x08000498        0x0
                0x08000498                . = ALIGN (0x4)
                0x08000498                _S_rodata = .
 *(.rodata*)
                0x08000498                . = ALIGN (0x4)
                0x08000498                _E_rodata = .

.bss            0x2000000c       0x18
                0x2000000c                . = ALIGN (0x4)
                0x2000000c                _S_bss = .
 *(.bss*)
 .bss           0x2000000c        0x0 startup.o
 .bss           0x2000000c        0x0 Main_Algorithm.o
 .bss           0x2000000c        0x4 Pressure_Sensor.o
                0x2000000c                Pressure_Value
 .bss           0x20000010        0x0 main.o
 .bss           0x20000010        0x0 Alarm_Actuator.o
 .bss           0x20000010        0x0 driver.o
                0x20000010                . = ALIGN (0x4)
```

Symbols Table of executable file:

```
$ arm-none-eabi-nm.exe Pressure_Control_System.elf
20000024 B _E_bss
2000000c D _E_data
08000498 D _E_rodata
0800048c T _E_text
2000000c B _S_bss
20000000 D _S_data
08000498 D _S_rodata
20001024 B _stack_top
0800026c T Alarm_Actuator_init
20000020 B Alarm_Actuator_State
20000010 B Alarm_Actuator_State_Id
20000008 D Alarm_Duration
20000018 B Alarm_State
080000c4 T Default_Handler
08000380 T Delay
0800028c T Detect_High_Pressure
08000174 T Get_Pressure_Value
080003a4 T getPressureVal
0800040c T GPIO_INITIALIZATION
080000c4 W Hard_Fault_Handler
08000234 T main
20000014 B Main_Algorithm_State
2000001a B Main_Algorithm_State_Id
080000c4 W MM_Fault_Handler
080000c4 W NMI_Handler
20000004 D Pressure_Pull_Timer
08000168 T Pressure_Sensor_init
2000001c B Pressure_Sensor_State
20000019 B Pressure_Sensor_State_Id
20000011 B Pressure_State
20000000 D Pressure_Threshold
2000000c B Pressure_Value
08000014 T Reset_Handler
080003bc T Set_Alarm_actuator
080000d0 T Set_Pressure_Value
080001f0 T setup
08000350 T ST_Alarm_Actuator_OFF
080002fc T ST_Alarm_Actuator_ON
08000368 T ST_Alarm_Actuator_Waiting
08000100 T ST_Detect_High_Pressure
0800018c T ST_Pressure_Sensor_Reading
080001c4 T ST_Pressure_Sensor_Waiting
08000000 T vectors
```

Simulate on eclipse by using printf:

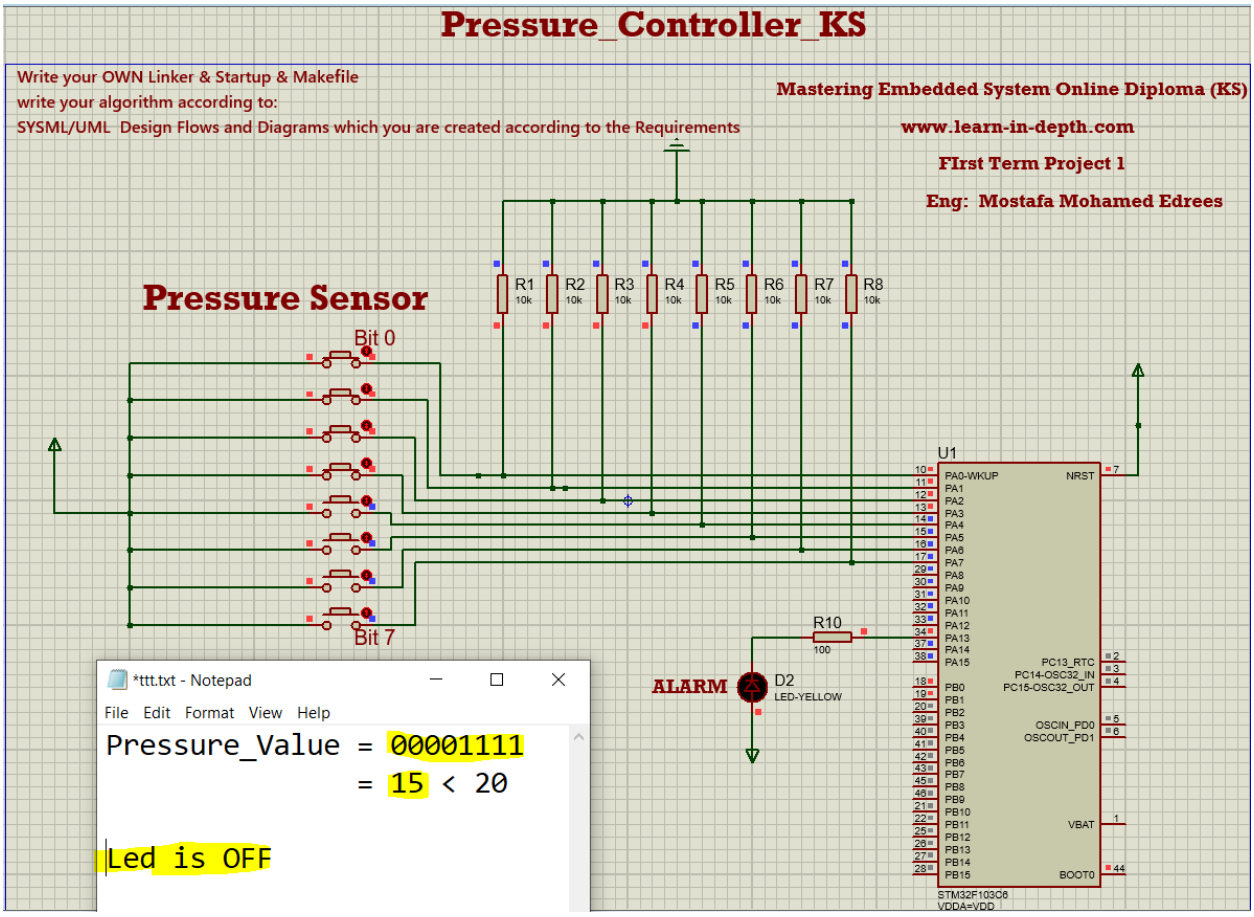
```
        Pressure_Sensor_init
        Alarm_Actuator_init
        ---------Led OFF------------

Alarm_Actuator_OFF State
Pressure_Sensor_Reading State: Pressure = 23
Pressure Sensor ---------------- Pressure = 23 ---------------> Main Algorithm
Pressure_Sensor_Waiting State
        ----------High Pressure------------
Main Algorithm ---------------- Start Alarm ---------------> Alarm Actuator
        ----------Led ON------------
Alarm_Actuator_ON State
Alarm_Actuator_Waiting State
        ----------Led OFF-----------
Pressure_Sensor_Reading State: Pressure = 16
Pressure Sensor ---------------- Pressure = 16 ---------------> Main Algorithm
Pressure_Sensor_Waiting State
        ----------Low Pressure------------
Main Algorithm ---------------- Stop Alarm ---------------> Alarm Actuator
        ----------Led OFF------------
Alarm_Actuator_OFF State
Pressure_Sensor_Reading State: Pressure = 22
Pressure Sensor ---------------- Pressure = 22 ---------------> Main Algorithm
Pressure_Sensor_Waiting State
        ----------High Pressure------------
Main Algorithm ---------------- Start Alarm ---------------> Alarm Actuator
        ----------Led ON------------
Alarm_Actuator_ON State
Alarm_Actuator_Waiting State
        ----------Led OFF------------
```

Simulate on Proteus:

We will enter the pressure value by 8 switch buttons and these implement the binary number of pressure value.

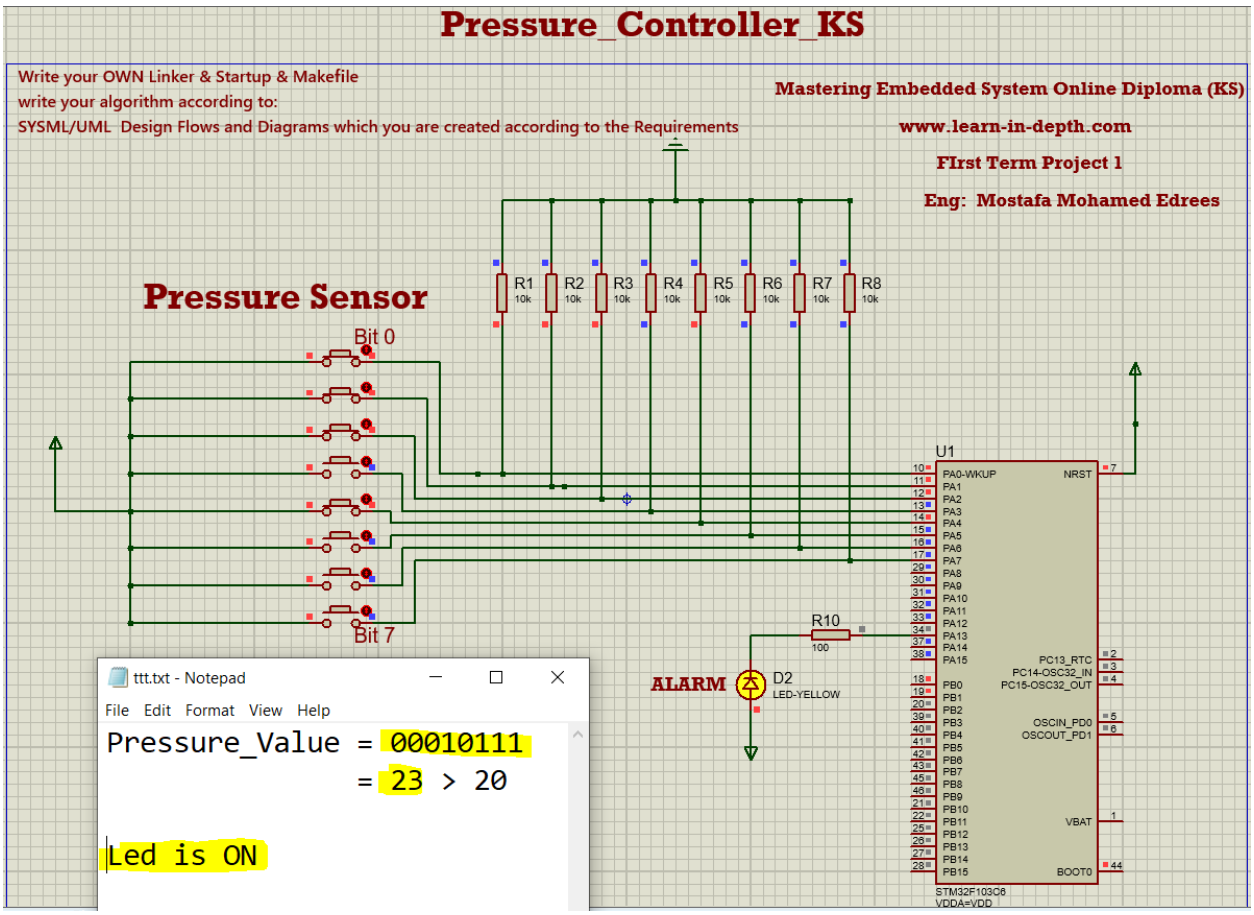
Case1:

The Pressure Value is less than Threshold (20).



Case2:

The Pressure Value is more than Threshold (20).



You can see run of the project from here:

[Running Video]