
# Mastering Embedded System Diploma

# Learn-in-depth

## Pressure Controller Project Report

**by:**

**Ehab Mohamed Abdelhamed**

**Unit:**

**Unit 5 First Term Projects**

**Under the supervision of:**

**Eng. Keroles Shenouda**

# Table of contents

# Pressure Controller System

## • Description:

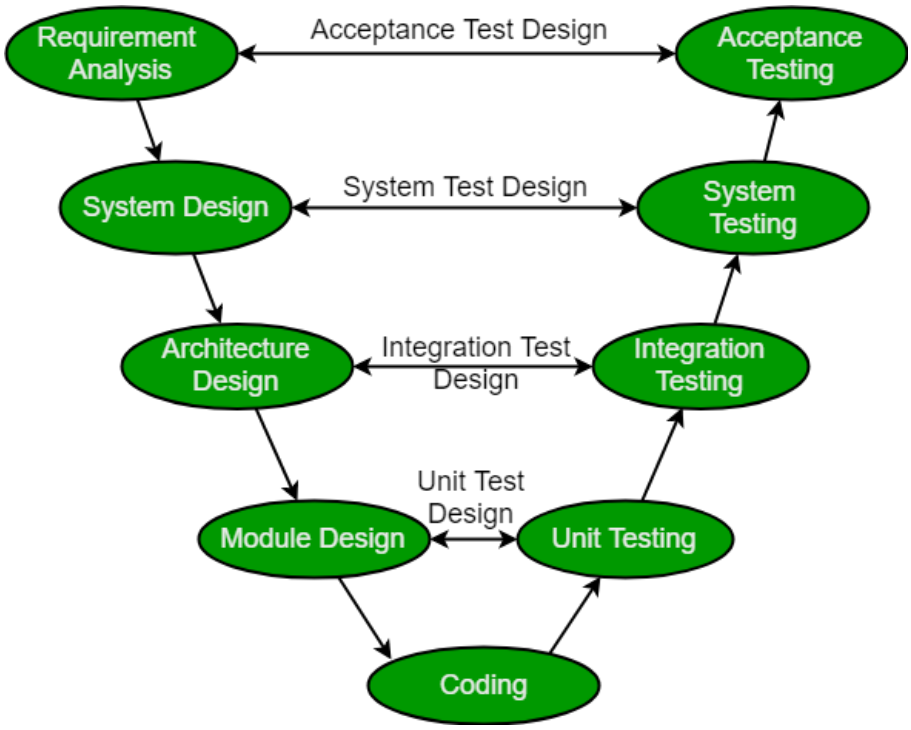
- A pressure controller informs the crew of a cabin with an alarm when the pressure exceeds 20 bars in the cabin.
- The alarm duration equals 60 seconds.
- keeps track of the measured values.

## • Case Study:

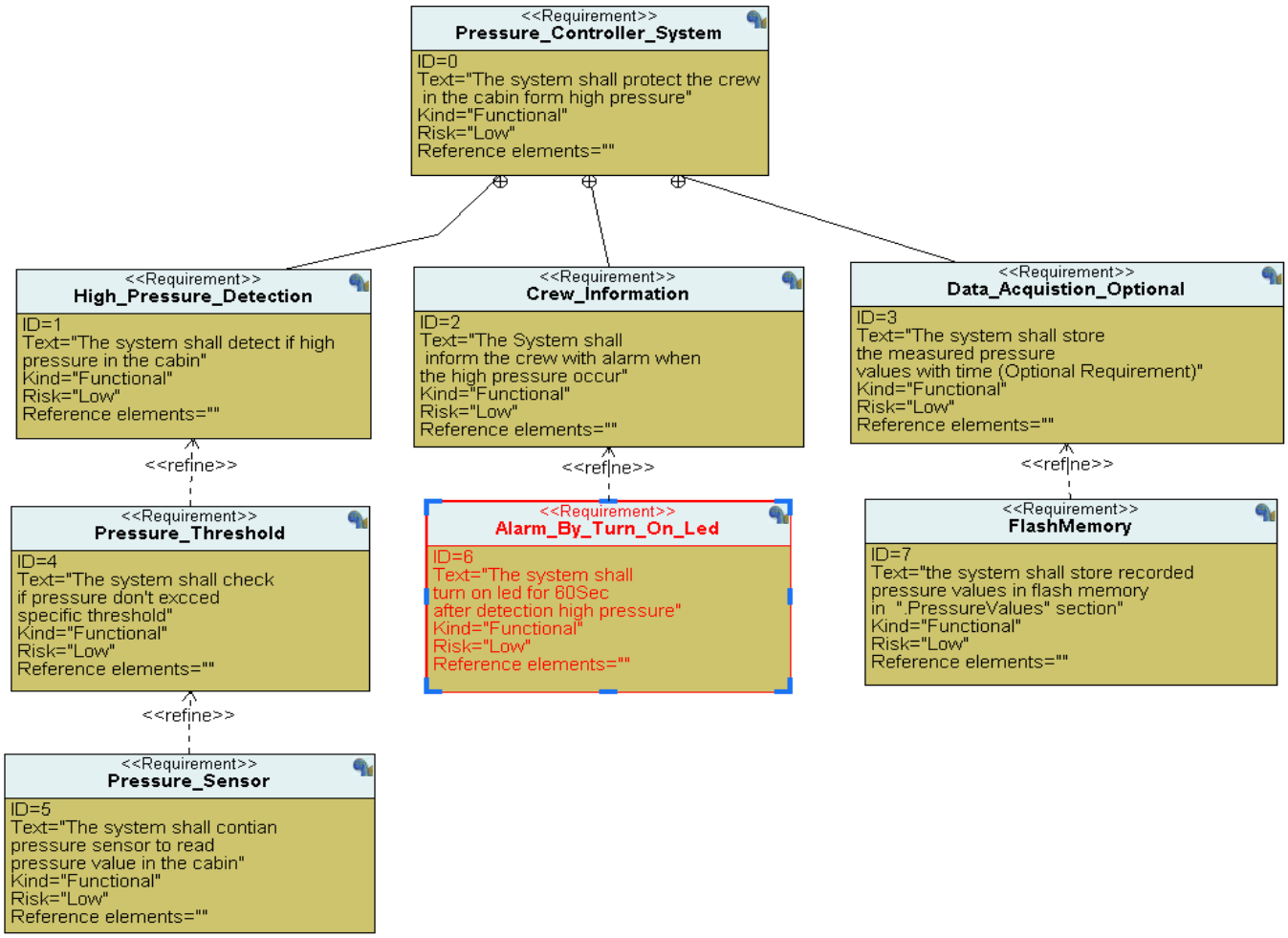
| | Assumptions | Acceptance |
|---|---|---|
| 2 | The controller set up and shutdown procedures are not modeled | ☑ |
| 3 | The controller maintenance is not modeled | ☑ |
| 4 | The pressure sensor never fails | ☑ |
| 5 | The alarm never fails | ☑ |
| 6 | The controller never faces power cut | ☑ |
| 7 | The "keep track of measured value" option is not modeled in the first version of the design | ☑ |
| 8 | The detection of errors when occurs in the system is not modeled | ☑ |

## • Method:

V-Model SDLC:



# • System Requirements:




| Requirments | ID | Customer Acceptance |
|---|---|---|
| Pressure Controller System | 0 | ☑ |
| High Pressure Detection | 1 | ☑ |
| Preusure Threshold | 2 | ☑ |
| Pressure Sensor | 3 | ☑ |
| Crew Information | 4 | ☑ |
| Alarm By Turn On Led | 5 | ☑ |
| Data acquisition (Optional) | 6 | ☑ |
| Flash Memory | 7 | ☑ |

## • Space Exploration/partitioning

Tasks:

| | |
|---|---|
| - Read form sensor | (30 % CPU load) |
| - Main algorithm | (40 % CPU load) |
| -Turn on alarm | (20 % CPU load) |

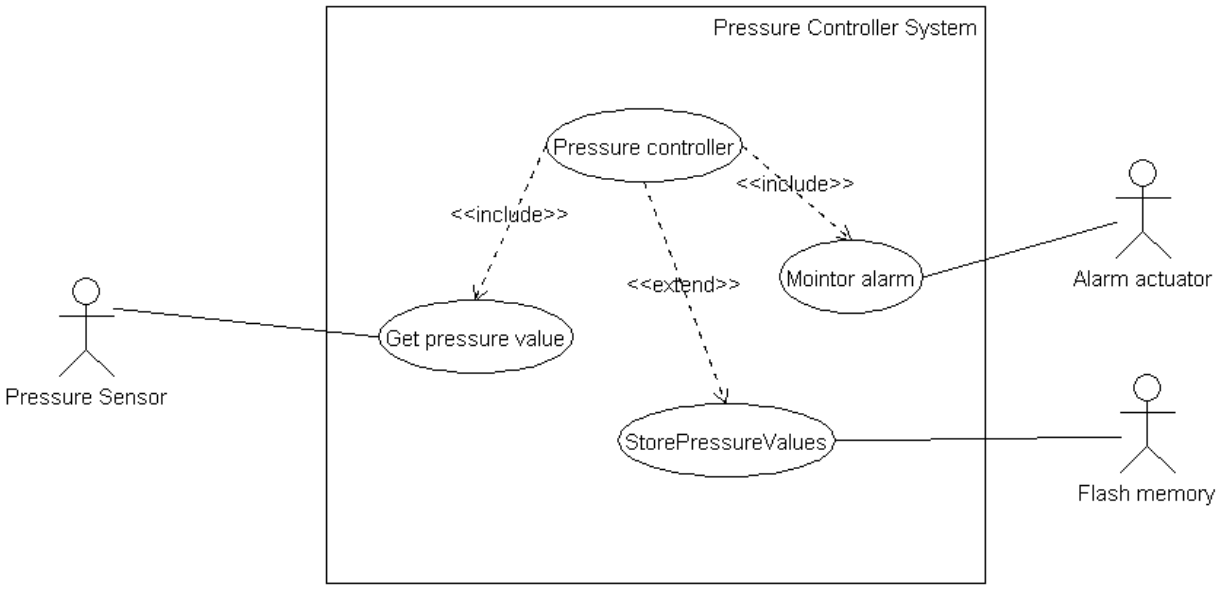
Options:

- Atmega32: 8-bit AVR 32 KB ISP flash memory with read-while-write capabilities, 1 KB EEPROM, 2 KB SRAM, 54/69 general purpose I/O lines.
- TM4C123GH6PM (Tica C): Arm Cortex-M4F based microcontrollers (MCUs). Featuring an 80-MHz Arm Cortex-M4F central processing unit, 256kB of flash and 32 kB
- STM32F103C8T6: RM® 32-bit Cortex® -M3 CPU, 72 MHz max CPU frequency ,64 KB Flash and 20 KB SRAM.

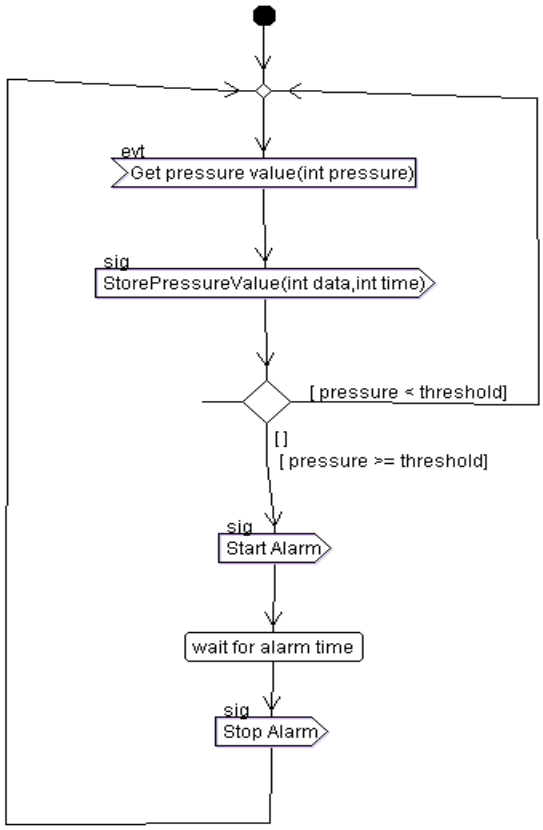
From performance analysis we found the optimal solution is STM32F103C8T6.

## • System Analysis

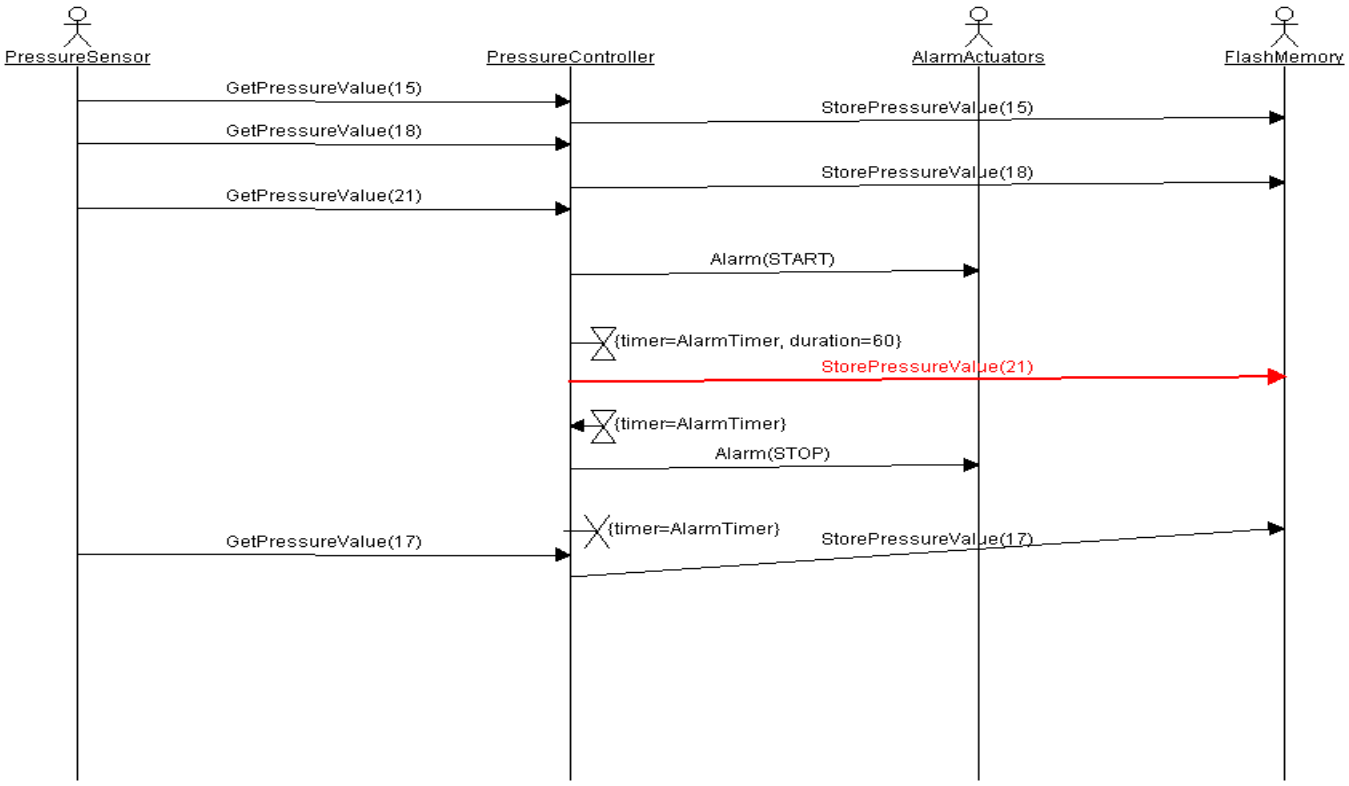
- **Use case diagram** → System boundary and main functions.



- **Activity Diagram** → Relations between main functions**.**

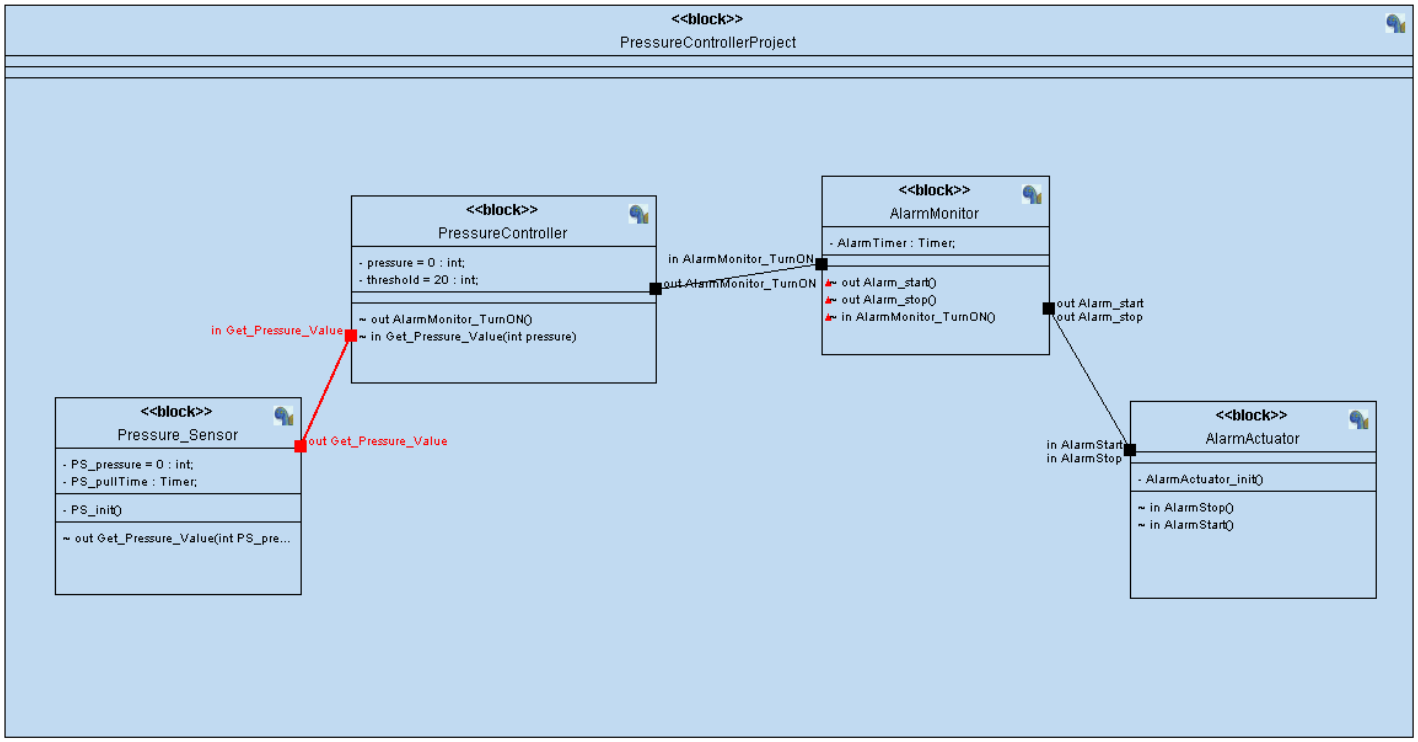



- **Sequence Diagram** → Communications between main system entities and actors.



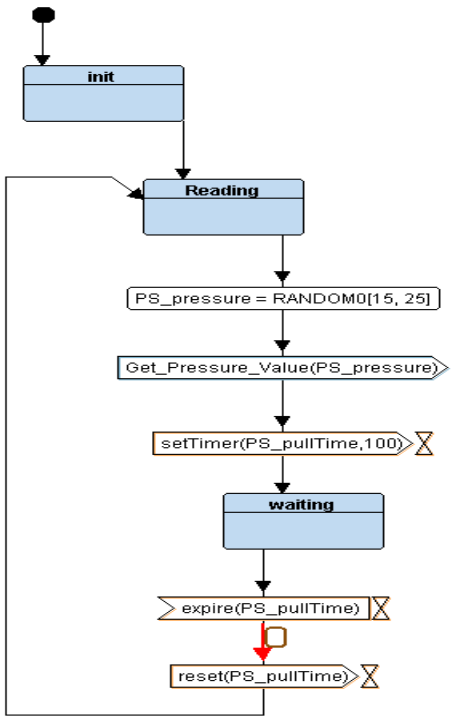
# • System Design

## - Block diagram:

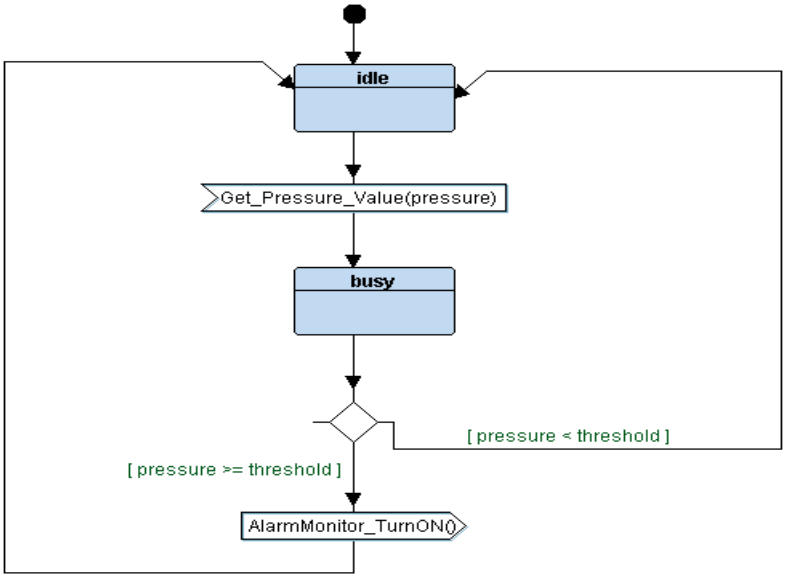



## - State Machine

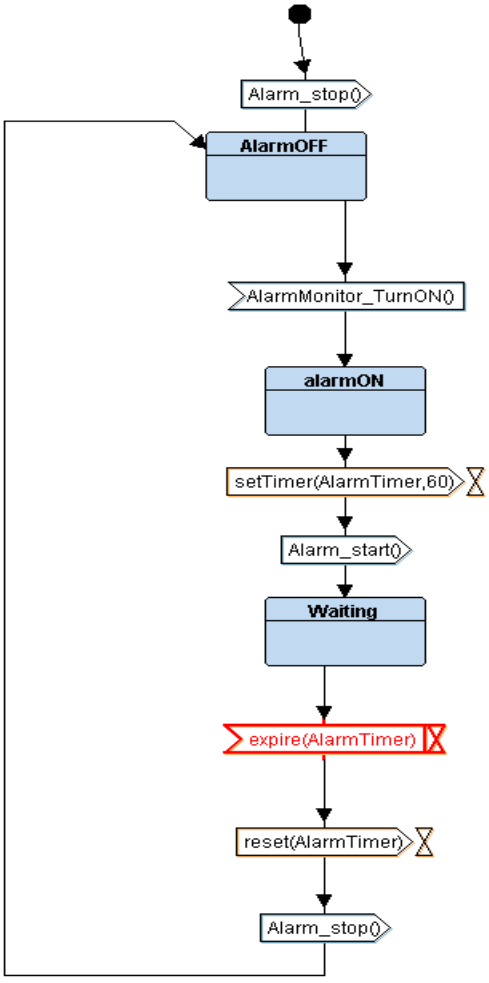
a) Pressure Sensor State Machine:



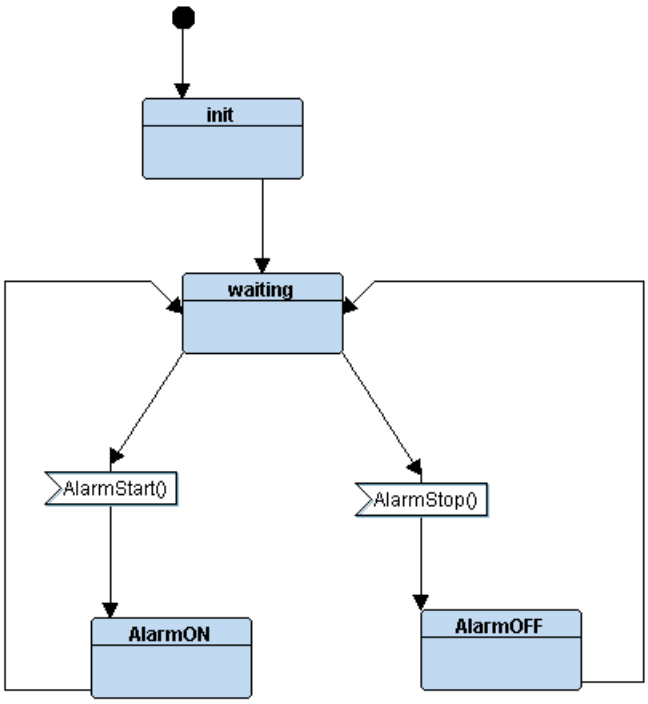
b) Pressure Controller State Machine:



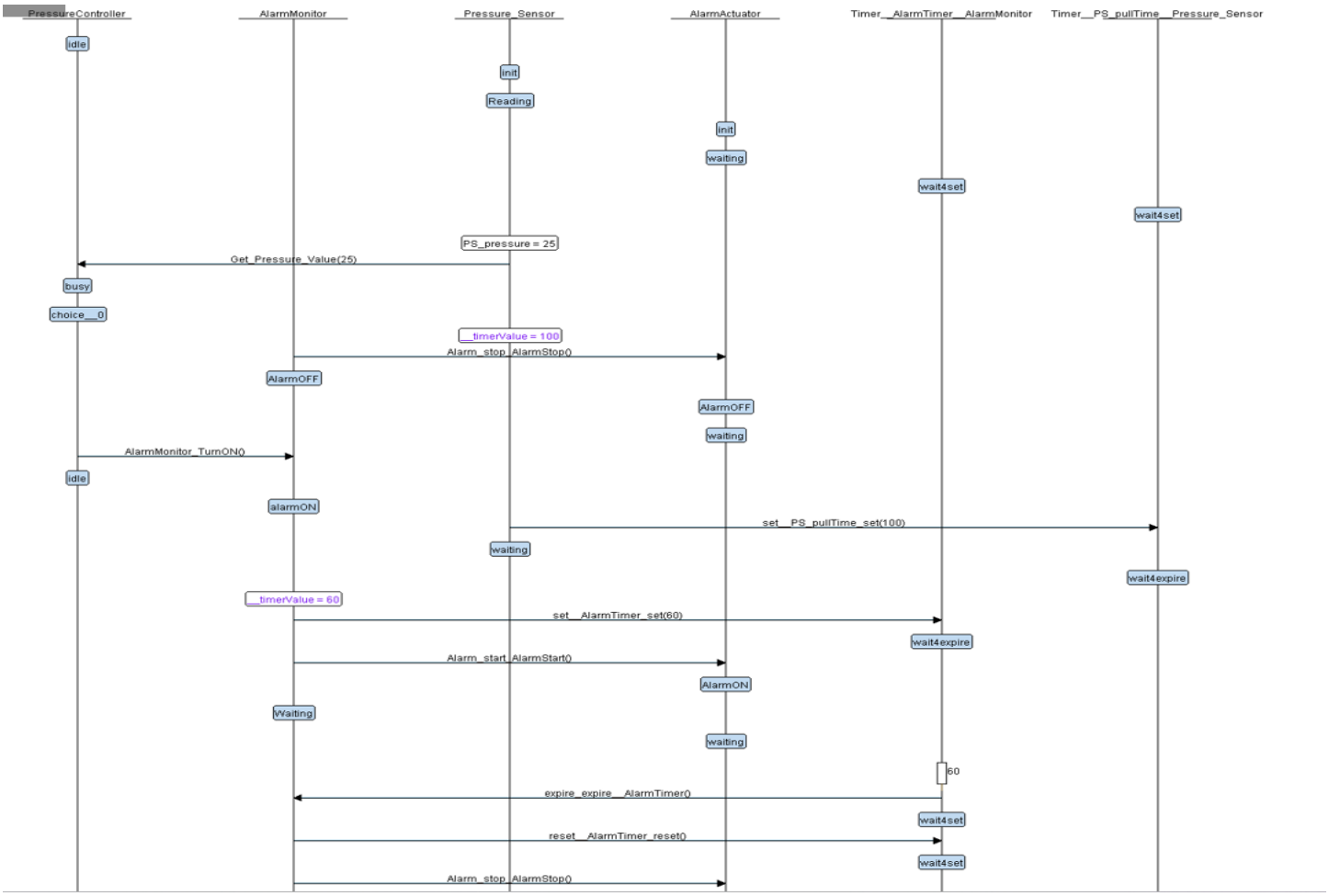

c) Alarm Monitor State Machine:

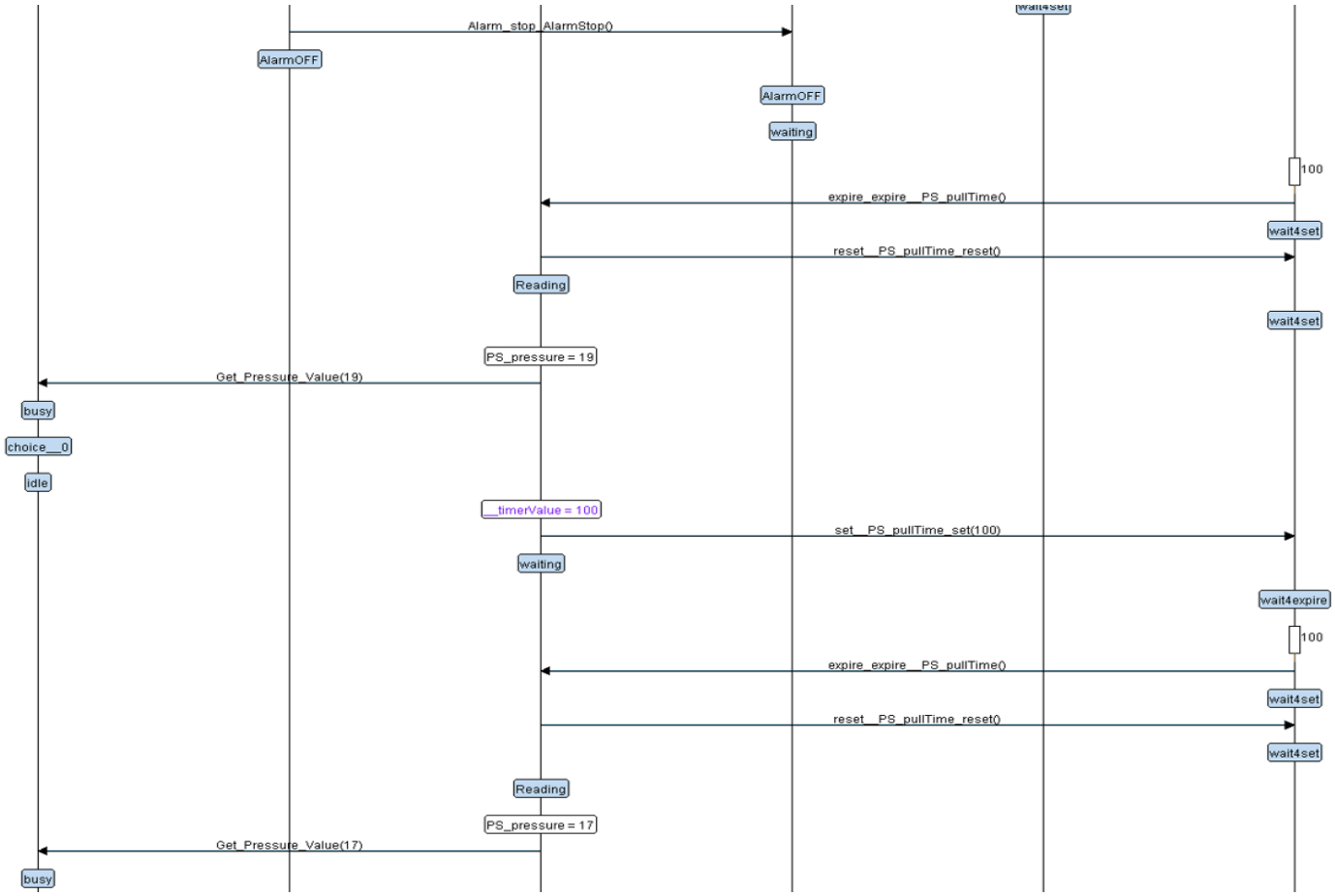


c) Alarm Actuator State Machine:




- **Verification logic output of state machines:**




Alarm_stop_AlarmStop()
AlarmOFF
AlarmOFF
waiting
100
expire_expire__PS_pullTime()
wait4set
reset__PS_pullTime_reset()
Reading
wait4set
PS_pressure = 19
Get_Pressure_Value(19)
busy
choice__0
idle
__timerValue = 100
set__PS_pullTime_set(100)
waiting
wait4expire
100
expire_expire__PS_pullTime()
wait4set
reset__PS_pullTime_reset()
wait4set
Reading
PS_pressure = 17
Get_Pressure_Value(17)
busy

- **Implementation stage**

  - main.c

```c
/*******************************************************************************
 * File Name: main.c
 * Description: Pressure controller system
 * Author: Ehab Mohamed Abdelhamed
 ******************************************************************************/
#include "pressure_controller.h"
#include "pressure_sensor.h"
#include "alarm_actuator.h"
#include "alarm_monitor.h"
#include "driver.h"

void App_init(){
    //Initialization all modules
    GPIO_INITIALIZATION ();
    PS_init();
    AlarmActuator_init();
}

//Running Application
void App_start(){
    /********* Super Loop************/
    while(1){
        //Call states;
        (*PS_state)();
        (*PressureController_state)();
        (*AlarmMonitor_state)();
        (*AlarmActuator_state)();
    }
}

//Main Function
void main(void){
    App_init();
    App_start();
}
```

  - state.h

```c
/*******************************************************************************
 * File Name: state.h
 * Description: Definition some macros for create states
 * Author: Ehab Mohamed Abdelhamed
 ******************************************************************************/

#ifndef STATE_H_
#define STATE_H_

/*******************************************************************************
 *                          Global Macros for states                           *
 *******************************************************************************/
#define CREATE_STATE(STATE_NAME) void STATE_##STATE_NAME()
#define STATE(STATE_NAME) STATE_##STATE_NAME

/*******************************************************************************
 *                          Module Connection                                  *
 *******************************************************************************/
void GetPressureValue(int Pval);
void AlarmMointorTurnON();
void AlarmStart();
void AlarmStop();



#endif /* STATE_H_ */
```

- driver

```c
/*******************************************************************************
 * File Name: drive.c
 * Description: device drivers for Pressure controller system
 * Author: Ehab Mohamed Abdelhamed
 ******************************************************************************/
#include "driver.h"
#include <stdint.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;
}

```

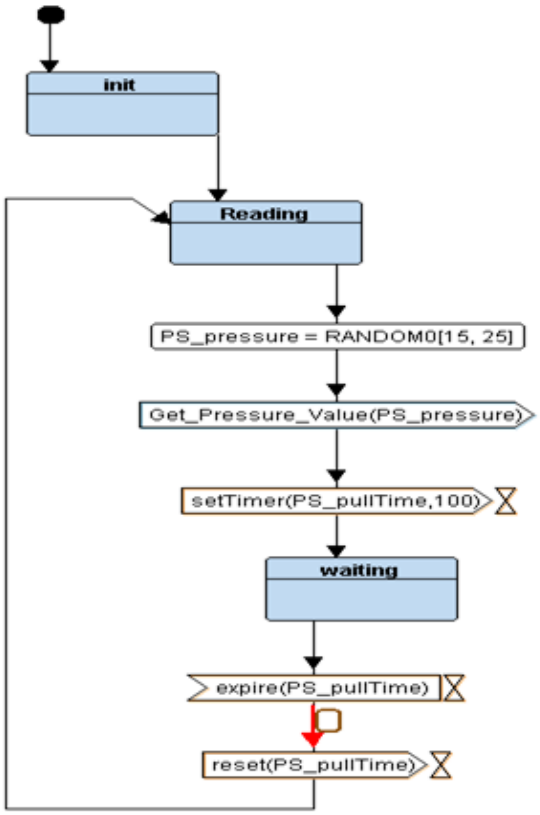
```c
/*******************************************************************************
 * File Name: drive.h
 * Description: Header file for device drivers for Pressure controller system
 * Author: Ehab Mohamed Abdelhamed
 ******************************************************************************/
#include <stdint.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 ();

```

- Pressure Sensor

```c
/*****************************************************************************
 * File Name: pressure_sesnor.c
 * Description: source file for pressure sensor driver to read pressure value
 * Author: Ehab Mohamed Abdelhamed
 *****************************************************************************/
#include "pressure_sensor.h"

/*****************************************************************************
 *                      Module Global Variables                              *
 *****************************************************************************/

uint32 PS_pressure=0;
void(*PS_state)()=STATE(PS_reading);
ePS_states_t PS_state_id=PS_READING;

/*****************************************************************************
 *                           Functions Definition                            *
 *****************************************************************************/

/*******************************************************************************
 * Function Name: PS_init()
 * Description: Function to intialize pressure sensor driver
 *******************************************************************************/

void PS_init(){
    /*intializtion of GPIO Pins of the module*/
}

/*******************************************************************************
 * Function Name: STATE_PS_reading
 * Description: Function to  measure pressure value
 *******************************************************************************/
CREATE_STATE(PS_reading){
    //state name
    PS_state_id=PS_READING;
    //state action
    /*
    * Read from pressure sensor
    * Send the measured value to pressure controller module
    */
    PS_pressure=getPressureVal();
    GetPressureValue(PS_pressure);
    //Jump to waiting state
    PS_state=STATE(PS_waiting);
}
```

```c
/*******************************************************************************
 * Function Name: STATE_PS_waiting
 * Description: Function to wait fo pulling time
 *******************************************************************************/
CREATE_STATE(PS_waiting){
    //state name
    PS_state_id=PS_WAITING;
    //state action
    /*
    * waiting pull time
    * jump to reading state
    */
    Delay(10000);
    PS_state=STATE(PS_reading);
}
```

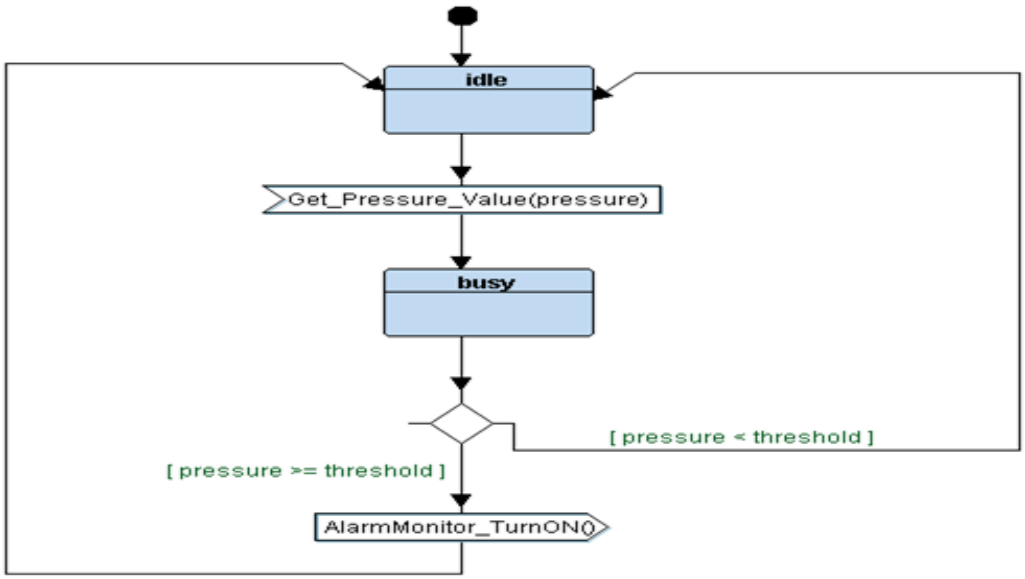
```c
/******************************************************************************
 * File Name: pressure_sesnor.h
 * Description: source file for pressure sensor driver to read pressure value
 * Author: Ehab Mohamed Abdelhamed
 ******************************************************************************/
#ifndef PRESSURE_SENSOR_H_
#define PRESSURE_SENSOR_H_

#include "Platform_Types.h"
#include "driver.h"
#include "state.h"
/*******************************************************************************
 *                                Extern Variables                             *
 *******************************************************************************/

extern void(*PS_state)(); //Poninter to function pressure sensor state

/*******************************************************************************
 *                                Module Data Types                            *
 *******************************************************************************/
typedef enum{
    PS_READING,PS_WAITING
}ePS_states_t;
/*******************************************************************************
 *                                Module States                                *
 *******************************************************************************/
 //Function to intiailzation the pressure sensor
void PS_init();
//Reading state for pressure sensor
CREATE_STATE(PS_reading);
//waiting state for pressure sensor
CREATE_STATE(PS_waiting);


#endif /* PRESSURE_SENSOR_H_ */
```



- Pressure controller

```c
/*******************************************************************************
 * File Name: pressure_controller.c
 * Description: soruce file for pressure controller to detect high pressure
 * Author: Ehab Mohamed Abdelhamed
 ******************************************************************************/

#include "pressure_controller.h"

/*******************************************************************************
 *                           Module Global Variables                           *
 *******************************************************************************/

uint32 pressure=0;
uint32 threshold=PRESSURE_THRESHOLD;
void(*PressureController_state)()=STATE(PressureController_busy);
PressureController_states_t PressureController_state_id=PressureController_idle;

/*******************************************************************************
 *                                Functions Definition                         *
 *******************************************************************************/

 /*******************************************************************************
 * Function Name: GetPressureValue
 * Description: Function to pressure value from pressure sensor
 *******************************************************************************/

void GetPressureValue(int Pval){
    pressure=Pval;
    PressureController_state=STATE(PressureController_busy);
}

 /*******************************************************************************
 * Function Name: CREATE_STATE(PressureController_idle)
 * Description: Function to waiting untill sensor send pressure value
 *******************************************************************************/

CREATE_STATE(PressureController_idle){
    //state name
    PressureController_state_id=PressureController_idle;
    //state action
    //no thing , waiting untill sensor send pressure value
}
 /*******************************************************************************
 * Function Name: CREATE_STATE(PressureController_busy)
 * Description: Function to check if pressure value larger than threshold
 *******************************************************************************/

CREATE_STATE(PressureController_busy){
    //state name
    PressureController_state_id=PressureController_busy;
    //state action
    if(pressure>=threshold){
        AlarmMointorTurnON();
    }
    PressureController_state=STATE(PressureController_idle);
}

```

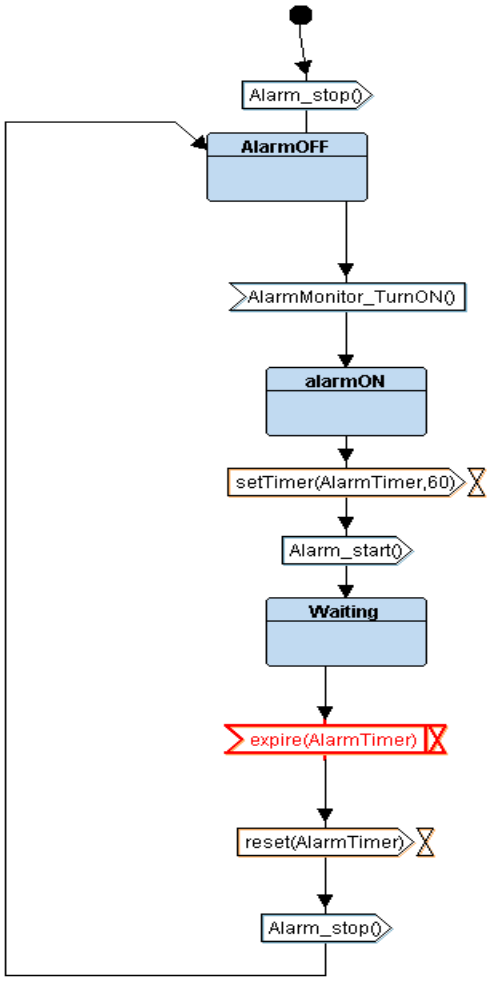
```c
/*******************************************************************************
 * File Name: pressure_controller.h
 * Description: Header file for pressure controller
 * Author: Ehab Mohamed Abdelhamed
 ******************************************************************************/

#ifndef PRESSURE_CONTROLLER_H_
#define PRESSURE_CONTROLLER_H_

#include "Platform_Types.h"
#include "state.h"

/*******************************************************************************
 *                                User configuration                           *
 *******************************************************************************/

#define PRESSURE_THRESHOLD (20u)

/*******************************************************************************
 *                                Extern Variables                             *
 *******************************************************************************/

extern void(*PressureController_state)();


/*******************************************************************************
 *                                Module Data Types                            *
 *******************************************************************************/

typedef enum{
    PressureController_idle,PressureController_busy
}PressureController_states_t;

/*******************************************************************************
 *                                Module States                                *
 *******************************************************************************/

/*State for waiting the pressure value from pressure signal */
CREATE_STATE(PressureController_idle);

/*State for check if the pressure value larger than threshold
* if it larger than threshold turn on alarm mointor
* if not back to idle state
*/
CREATE_STATE(PressureController_busy);

#endif /* PRESSURE_CONTROLLER_H_ */
```



- Alarm mointor

```c
/*******************************************************************************
 * File Name: alarm_monitor.c
 * Description: soruce file for alarm monitor to give alarm when high pressure detected
 * Author: Ehab Mohamed Abdelhamed
 *******************************************************************************/
#include "alarm_monitor.h"

/*******************************************************************************
 *                           Module Global Variables                           *
 *******************************************************************************/

void(*AlarmMonitor_state)()=STATE(AlarmMonitor_alarmOFF);
AlarmMonitor_states_t AlarmMonitor_state_id=AlarmMonitor_alarmOFF;


/*******************************************************************************
 *                                Functions Definition                         *
 *******************************************************************************/

 /*******************************************************************************
 * Function Name: STATE_AlarmMonitor_alarmOFF
 * Description: Function to make alarm mointor on state when high pressure detection
 *******************************************************************************/
 void AlarmMointorTurnON(){
    AlarmMonitor_state=STATE(AlarmMonitor_alarmON);
}

/*******************************************************************************
 * Function Name: STATE_AlarmMonitor_alarmOFF
 * Description: intial state of alarm mointor and actutors is turned off
 *******************************************************************************/

CREATE_STATE(AlarmMonitor_alarmOFF){
    //state name
    AlarmMonitor_state_id=AlarmMonitor_alarmOFF;
}

/*******************************************************************************
 * Function Name: STATE_AlarmMonitor_alarmON
 * Description: state for send signal to alarm actuator to turn on
 *******************************************************************************/

CREATE_STATE(AlarmMonitor_alarmON){
    //state name
    AlarmMonitor_state_id=AlarmMonitor_alarmON;
    //state action
    //start alarm actuators then move to waiting state
    AlarmStart();
    AlarmMonitor_state=STATE(AlarmMonitor_waiting);
}

/*******************************************************************************
 * Function Name: STATE_AlarmMonitor_waiting
 * Description: state for keep alarm on for alarm time period then turn off
 *******************************************************************************/

CREATE_STATE(AlarmMonitor_waiting){
    //State name
    AlarmMonitor_state_id=AlarmMonitor_waiting;
    //state action
    //Keep alarm on for alarm time period
    Delay(ALARM_TIME);
    AlarmStop();
    //move to alarm off state
    AlarmMonitor_state=STATE(AlarmMonitor_alarmOFF);
}
```

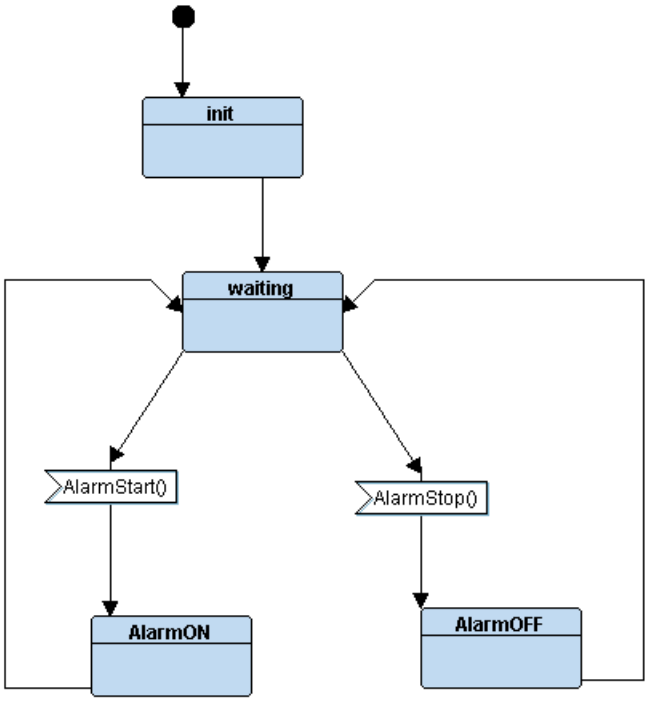
```c
/*******************************************************************************************
 * File Name: alarm_monitor.h
 * Description: Header file for alarm monitor
 * Author: Ehab Mohamed Abdelhamed
 *******************************************************************************************/

#ifndef ALARM_MONITOR_H_
#define ALARM_MONITOR_H_

#include "Platform_Types.h"
#include "driver.h"
#include "state.h"
/*******************************************************************************************
 *                                User configuration                                       *
 *******************************************************************************************/

#define ALARM_TIME (6000000u)

/*******************************************************************************************
 *                                Extern Variables                                         *
 *******************************************************************************************/

extern void(*AlarmMonitor_state)();
/*******************************************************************************************
 *                                Module Data Types                                        *
 *******************************************************************************************/

typedef enum{
    AlarmMonitor_waiting,AlarmMonitor_alarmON,AlarmMonitor_alarmOFF
}AlarmMonitor_states_t;

/*******************************************************************************************
 *                                Module States                                            *
 *******************************************************************************************/
//state for waiting untill pressure controll send signal to turn on alarm
CREATE_STATE(AlarmMonitor_waiting);
//state to send signal to alarm actuator to turn on
CREATE_STATE(AlarmMonitor_alarmON);
//state to send signal to alarm actuator to turn off
CREATE_STATE(AlarmMonitor_alarmOFF);

#endif /* ALARM_MONITOR_H_ */

```



- Alarm actuator

```c
/*******************************************************************************
 * File Name: alarm_actuator.c
 * Description: Source file for alarm actuator driver
 * Author: Ehab Mohamed Abdelhamed
 *******************************************************************************/

#include "alarm_actuator.h"
/*******************************************************************************
 *                           Module Global Variables                           *
 *******************************************************************************/

void(*AlarmActuator_state)()=STATE(AlarmActuator_waiting);
AlarmActuator_states_t AlarmActuator_state_id=AlarmActuator_waiting;

/*******************************************************************************
 *                                Functions Definition                         *
 *******************************************************************************/

/*********************************************************************************************
 * Function Name: AlarmStart
 * Description: Signal to make alarm actuator in on state
 *********************************************************************************************/

void AlarmStart(){
    AlarmActuator_state=STATE(AlarmActuator_alarmON);
}
/*********************************************************************************************
 * Function Name: AlarmStop
 * Description: Signal to make alarm actuator in off state
 *********************************************************************************************/

void AlarmStop(){
    AlarmActuator_state=STATE(AlarmActuator_alarmOFF);
}
/*********************************************************************************************
 * Function Name: AlarmActuator_init
 * Description: Function to initailze actuator driver
 *********************************************************************************************/

void AlarmActuator_init(){
    /*intialization*/
    //intiatilze actuators by off (Active Low)
    Set_Alarm_actuator(TRUE);
}

/*********************************************************************************************
 * Function Name: STATE_AlarmActuator_waiting
 * Description: waiting signal to action
 *********************************************************************************************/

CREATE_STATE(AlarmActuator_waiting){
    //State Name
    AlarmActuator_state_id=AlarmActuator_waiting;
}
/*********************************************************************************************
 * Function Name: STATE_AlarmActuator_alarmON
 * Description: turn on alarm actuator then back to waiting state
 *********************************************************************************************/

CREATE_STATE(AlarmActuator_alarmON){
    //State Name
    AlarmActuator_state_id=AlarmActuator_alarmON;
    //State action
    Set_Alarm_actuator(FALSE);
    AlarmActuator_state=STATE(AlarmActuator_waiting);

}
/*********************************************************************************************
 * Function Name: STATE_AlarmActuator_alarmOFF
 * Description: turn off alarm actuator then back to waiting state
 *********************************************************************************************/
CREATE_STATE(AlarmActuator_alarmOFF){
    //State Name
    AlarmActuator_state_id=AlarmActuator_alarmOFF;
    //State action
    Set_Alarm_actuator(TRUE);
    AlarmActuator_state=STATE(AlarmActuator_waiting);
}
```

```c
/******************************************************************************
 * File Name: alarm_actuator.h
 * Description: Header file for alarm actuator driver
 * Author: Ehab Mohamed Abdelhamed
 ******************************************************************************/

#ifndef ALARM_ACTUATOR_H_
#define ALARM_ACTUATOR_H_

#include "Platform_Types.h"
#include "driver.h"
#include "state.h"
/*******************************************************************************
 *                                Extern Variables                             *
 *******************************************************************************/
extern void(*AlarmActuator_state)();

/*******************************************************************************
 *                                Module Data Types                            *
 *******************************************************************************/
typedef enum{
    AlarmActuator_waiting,AlarmActuator_alarmON,AlarmActuator_alarmOFF
}AlarmActuator_states_t;

/*******************************************************************************
 *                                Module States                                *
 *******************************************************************************/
//function to intialize actuator driver
void AlarmActuator_init();
//waiting state for receive signal
CREATE_STATE(AlarmActuator_waiting);
//turn on alarm actuator
CREATE_STATE(AlarmActuator_alarmON);
//turn off alarm actuator
CREATE_STATE(AlarmActuator_alarmOFF);

#endif /* ALARM_ACTUATOR_H_ */
```



- **startup.c**

```c
/*******************************************************************************
 * File Name: startup.c
 * Description: startup file for STM32F103C8T6 (32-bit Cortex-M3 CPU)
 * Author: Ehab Mohamed Abdelhamed
 *******************************************************************************/

/* Enable Exceptions ... This Macro enable IRQ interrupts, by clearing the I-bit in the PRIMASK. */
#define Enable_Exceptions()     __asm("CPSIE I")

/* Disable Exceptions ... This Macro disable IRQ interrupts,by clearing the I-bit in the PRIMASK. */
#define Disable_Exceptions()    __asm("CPSID I")


extern unsigned long _TEXT_END;
extern unsigned long _DATA_START;
extern unsigned long _DATA_END;
extern unsigned long _BSS_START;
extern unsigned long _BSS_END;
extern unsigned long _STACK_POINTER_TOP;

 void main(void);
 void Defualt_handler(void);

 /*Create interrupt vector table*/
 void _reset(void);                                                                  //2-Reset_handler
 void NMI_handler(void)__attribute__((weak, alias ("Defualt_handler")));             //3-NMI_handler
 void HardFault_handler(void)__attribute__((weak, alias ("Defualt_handler")));       //4-HardFault_handler
 void MemManage_handler(void)__attribute__((weak, alias ("Defualt_handler")));       //5-MemManage_handlr
 void BusFault_handler(void)__attribute__((weak, alias ("Defualt_handler")));        //6-BusFault_handler
 /*Reserved*/                                                                        //7-Reserved
 void SVCall_handler(void)__attribute__((weak, alias("Defualt_handler")));           //8-SVCall_handler
 void DebugMointor_handler(void)__attribute__((weak, alias ("Defualt_handler")));    //9-DebugMointor_handler
 /*Reserved*/                                                                        //10-Reserved
 void PendSV_handler(void)__attribute__((weak, alias ("Defualt_handler")));          //11-PendSV_handlr
 void SysTick_handler(void)__attribute__((weak, alias ("Defualt_handler")));         //12-SysTick_handler
 void WWDG_handler(void)__attribute__((weak, alias ("Defualt_handler")));            //13-WWDG_handlr
 void PVD_handler(void)__attribute__((weak, alias ("Defualt_handler")));             //14-PVD_handler
 void TAMPER_handler(void)__attribute__((weak, alias ("Defualt_handler")));          //15-TAMPER_handler
 void FLASH_handler(void)__attribute__((weak, alias ("Defualt_handler")));           //16-FLASH_handler
 void RCC_handler(void)__attribute__((weak, alias ("Defualt_handler")));             //17-RCC_handler
 void ECTI0_handler(void)__attribute__((weak, alias ("Defualt_handler")));           //18-ECTI0_handler
 void ECTI1_handler(void)__attribute__((weak, alias ("Defualt_handler")));           //19-ECTI1_handler
 void ECTI2_handler(void)__attribute__((weak, alias ("Defualt_handler")));           //20-ECTI2_handler
 //IRQ21 .........
 //IRQ22 .........
```

```c
 const unsigned long vectors[] __attribute__((section(".vectors")))=
 {
     (unsigned long)&_STACK_POINTER_TOP,
     (unsigned long)&_reset,
     (unsigned long)&NMI_handler,
     (unsigned long)&HardFault_handler,
     (unsigned long)&MemManage_handler,
     (unsigned long)&BusFault_handler,
     (unsigned long)0,
     (unsigned long)&SVCall_handler,
     (unsigned long)&DebugMointor_handler,
     (unsigned long)0,
     (unsigned long)&PendSV_handler,
     (unsigned long)&SysTick_handler,
     (unsigned long)&WWDG_handler,
     (unsigned long)&PVD_handler,
     (unsigned long)&TAMPER_handler,
     (unsigned long)&FLASH_handler,
     (unsigned long)&RCC_handler,
     (unsigned long)&ECTI0_handler,
     (unsigned long)&ECTI1_handler,
     (unsigned long)&ECTI2_handler
     //(unsigned long)&IRQ_21 .........
     //(unsigned long)&IRQ_22 .........
 };
```

```c
 //Defination of reset handler
 void _reset(void){
     //Disable all interrupt
     Disable_Exceptions();
     unsigned long i;
     const unsigned char *_Ptr_Source=(unsigned char *)&_TEXT_END;
     unsigned char  *_Ptr_Destination=(unsigned char *)&_DATA_START;
     unsigned long _SIZE_=((unsigned long)&_DATA_END - (unsigned long)&_DATA_START);
     //copy .data section that contian intialized data from flash to sram
     for(i=0;i<_SIZE_;i++){
         *(_Ptr_Destination)=*(_Ptr_Source);
         (unsigned char *)_Ptr_Destination++;
         (const unsigned char *)_Ptr_Source++;
     }
     //reserve .bss section in sram and intialize it by zero
    _Ptr_Destination=(unsigned char *)&_BSS_START;
     _SIZE_=((unsigned long)&_BSS_END - (unsigned long)&_BSS_START);
     for(i=0;i<_SIZE_;i++){
         *(_Ptr_Destination)=0;
         (unsigned char *)_Ptr_Destination++;
     }
     //Enble all interrupt
     Enable_Exceptions();
     //Jump to main function
     main();
     while(1){}
 }

 void Defualt_handler(void){
     while(1){}
 }
```

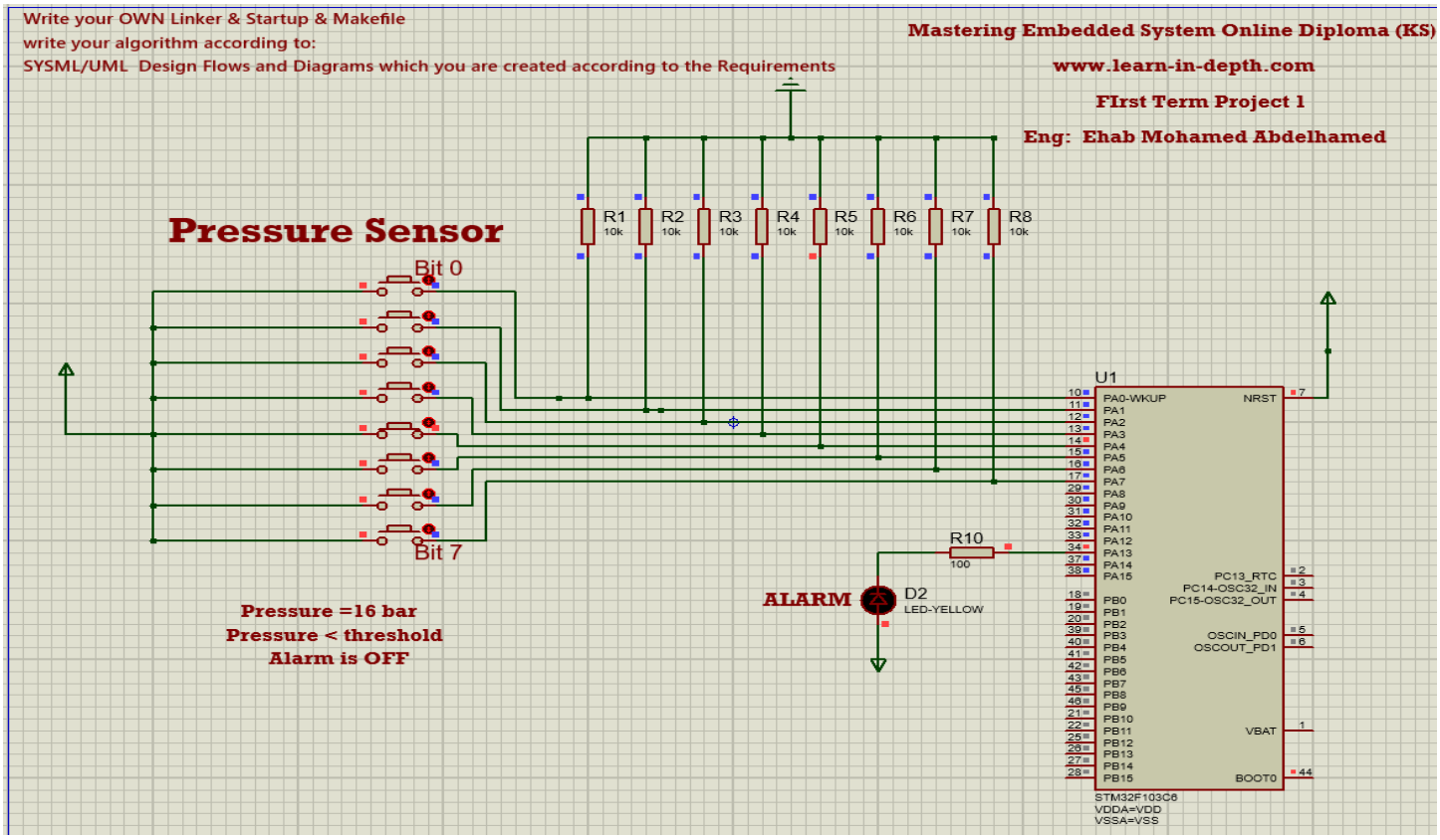
- Linker script

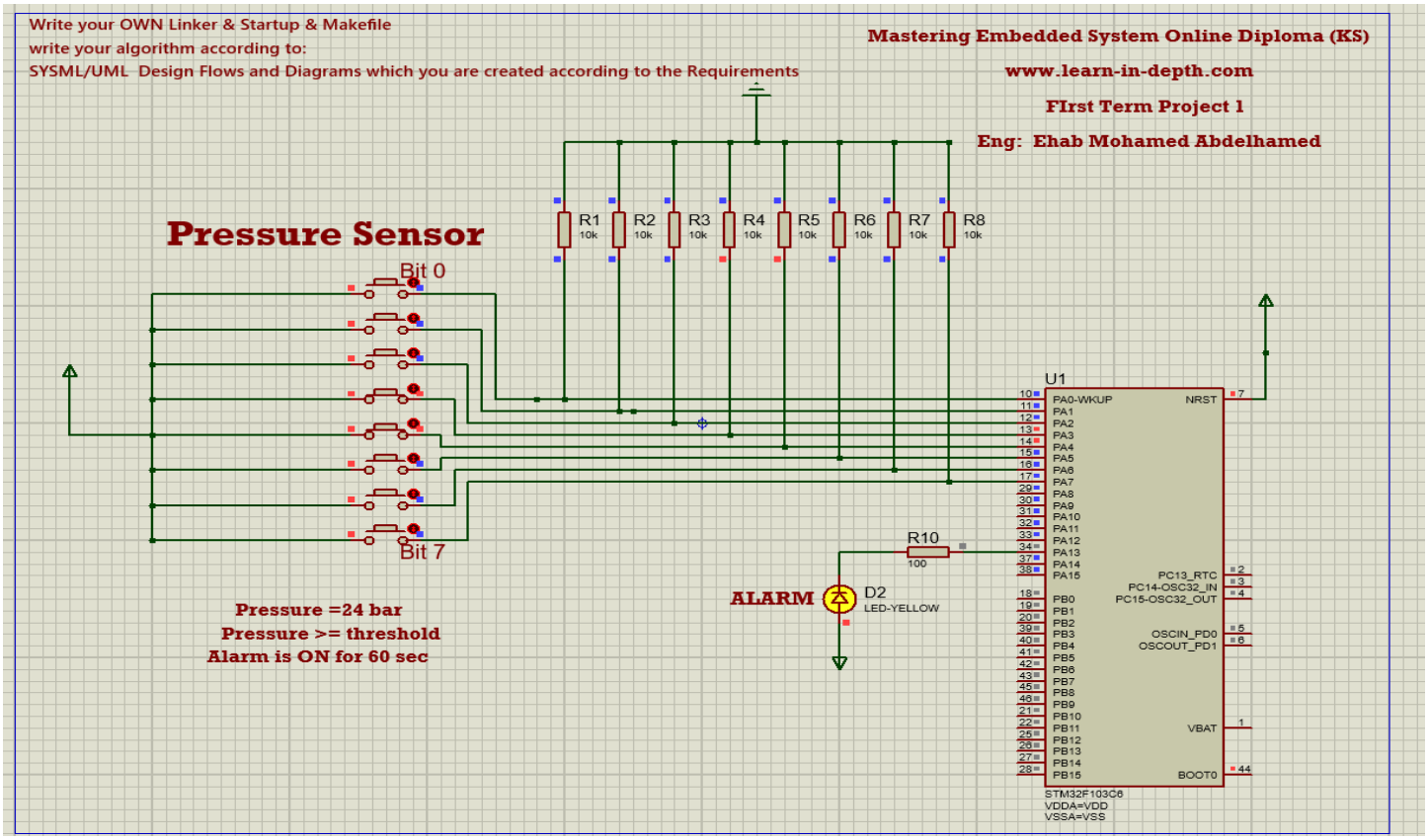
```
/*****************************************************************************
* File Name: linker_scriot.ld
* Description: linker script file for STM32F103C8T6 (32-bit Cortex-M3 CPU)
* Author: Ehab Mohamed Abdelhamed
*****************************************************************************/


ENTRY(_reset)

MEMORY
{
    flash (RX) : ORIGIN = 0x08000000 , LENGTH = 64K
    sram  (RWX): ORIGIN = 0x20000000 , LENGTH = 20K
}

SECTIONS
{
    .text :
    {
            startup.o(.vectors)
            *(.text*)
            *(.rodata*)
            . = ALIGN(4) ;
            _TEXT_END = . ;
    }> flash

    .data :
    {
        _DATA_START = . ;
        *(.data*)
        . = ALIGN(4) ;
        _DATA_END = . ;
    }> sram AT> flash

    .bss :
    {
        _BSS_START = . ;
        *(.bss*)
        . = ALIGN(4) ;
        _BSS_END = . ;
        . = . + 0x2048 ;
        _STACK_POINTER_TOP = . ;
    }> sram

    /*Section for store measured pressure values*/
    .pressureRecords :
    {
        *(.pressureRecords);
    }> flash
}
```

- Make file

```
#Makefile prabared by Eng.Ehab Moahmed Abdelhamed
CC= arm-none-eabi
CF= -g -gdwarf-2 -mcpu=cortex-m3
INCS=-I .
SRC= $(wildcard *.c)
ASM= $(wildcard *.s)
OBJ= $(SRC:.c=.o)  $(ASM:.s=.o)
ProjectName=PressureControllerSystem

all: $(ProjectName).bin
        @echo "******* Building is done *******"

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

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

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

%.o : %.s
    $(CC)-as.exe  -c $(CF) $< -o $@

clean_all:
    rm *.o *.elf *.bin
    @echo "**** Everything is clean ****"

clean:
    rm *.elf *.bin

```

- **Simulation:**


Write your OWN Linker & Startup & Makefile
write your algorithm according to:
SYSML/UML Design Flows and Diagrams which you are created according to the Requirements
Mastering Embedded System Online Diploma (KS)
www.learn-in-depth.com
FIrst Term Project 1
Eng: Ehab Mohamed Abdelhamed
Pressure Sensor
Bit 0
Bit 7
R1 10k
R2 10k
R3 10k
R4 10k
R5 10k
R6 10k
R7 10k
R8 10k
R10 100
ALARM
D2 LED-YELLOW
U1
PA0-WKUP
NRST
PC13_RTC
PC14-OSC32_IN
PC15-OSC32_OUT
OSCIN_PD0
OSCOUT_PD1
VBAT
BOOT0
STM32F103C6
VDDA=VDD
VSSA=VSS
Pressure =16 bar
Pressure < threshold
Alarm is OFF


Write your OWN Linker & Startup & Makefile
write your algorithm according to:
SYSML/UML Design Flows and Diagrams which you are created according to the Requirements
Mastering Embedded System Online Diploma (KS)
www.learn-in-depth.com
FIrst Term Project 1
Eng: Ehab Mohamed Abdelhamed
Pressure Sensor
Bit 0
Bit 7
R1 10k
R2 10k
R3 10k
R4 10k
R5 10k
R6 10k
R7 10k
R8 10k
R10 100
ALARM
D2 LED-YELLOW
U1
PA0-WKUP
NRST
PC13_RTC
PC14-OSC32_IN
PC15-OSC32_OUT
OSCIN_PD0
OSCOUT_PD1
VBAT
BOOT0
STM32F103C6
VDDA=VDD
VSSA=VSS
Pressure =24 bar
Pressure >= threshold
Alarm is ON for 60 sec

## • Misra rules checking

```
Ehab Mohamed@LAPTOP-OQ38O3RQ MINGW64 /e/Embedded_System_KS/4-Unit 5 First Term P
$ cppcheck.exe --dump main.c pressure_sensor.c pressure_controller.c alarm_monit
Checking alarm_actuator.c ...
1/5 files checked 28% done
Checking alarm_monitor.c ...
2/5 files checked 52% done
Checking main.c ...
3/5 files checked 59% done
Checking pressure_controller.c ...
4/5 files checked 80% done
Checking pressure_sensor.c ...
5/5 files checked 100% done
```

```
Ehab Mohamed@LAPTOP-OQ38O3RQ MINGW64 /e/Embedded_System_KS/4-Unit 5 First Term Projects/Pressure Cont
$ python.exe /e/Embedded_System_KS/Tools/Cppcheck/addons/misra.py --suppress-rules 20.10  main.c.dump
Checking main.c.dump...
Checking main.c.dump, config ...
Checking pressure_sensor.c.dump...
Checking pressure_sensor.c.dump, config ...
Checking pressure_controller.c.dump...
Checking pressure_controller.c.dump, config ...
Checking alarm_monitor.c.dump...
Checking alarm_monitor.c.dump, config ...
Checking alarm_actuator.c.dump...
Checking alarm_actuator.c.dump, config ...
```

## • SW analysis

- Sections

```
Ehab Mohamed@LAPTOP-OQ38O3RQ MINGW64 /e/Embedded_System_KS/4-Unit 5 F
$ arm-none-eabi-objdump.exe -h PressureControllerSystem.elf

PressureControllerSystem.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000418  08000000  08000000  00010000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000018  20000000  08000418  00020000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          0000205c  20000018  08000430  00020018  2**2
                  ALLOC
  3 .debug_info   0000082b  00000000  00000000  00020018  2**0
                  CONTENTS, READONLY, DEBUGGING
  4 .debug_abbrev 00000508  00000000  00000000  00020843  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    000005c0  00000000  00000000  00020d4b  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 000000e0  00000000  00000000  0002130b  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_line   000004cc  00000000  00000000  000213eb  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .debug_str    000004f1  00000000  00000000  000218b7  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      0000007b  00000000  00000000  00021da8  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  00021e23  2**0
                  CONTENTS, READONLY
 11 .debug_frame  00000374  00000000  00000000  00021e58  2**2
                  CONTENTS, READONLY, DEBUGGING
```

- Symbols

```
Ehab Mohamed@LAPTOP-OQ38O3RQ MINGW64 /e/Embedded_System_KS/4-Unit 5 First Term Pr
$ arm-none-eabi-nm.exe PressureControllerSystem.elf
2000002c B _BSS_END
20000018 B _BSS_START
20000018 D _DATA_END
20000000 D _DATA_START
08000050 T _reset
20002074 B _STACK_POINTER_TOP
08000418 T _TEXT_END
08000118 T AlarmActuator_init
20000000 D AlarmActuator_state
20000018 B AlarmActuator_state_id
0800018c T AlarmMointorTurnON
20000004 D AlarmMonitor_state
20000008 D AlarmMonitor_state_id
080000e0 T AlarmStart
080000fc T AlarmStop
080002d8 T App_init
080002ec T App_start
080000d8 W BusFault_handler
080000d8 W DebugMointor_handler
080000d8 T Defualt_handler
08000214 T Delay
```

```
08000214 T Delay
080000d8 W ECTI0_handler
080000d8 W ECTI1_handler
080000d8 W ECTI2_handler
080000d8 W FLASH_handler
08000234 T getPressureVal
0800032c T GetPressureValue
08000288 T GPIO_INITIALIZATION
080000d8 W HardFault_handler
0800031c T main
080000d8 W MemManage_handler
080000d8 W NMI_handler
080000d8 W PendSV_handler
2000001c B pressure
20000010 D PressureController_state
20000020 B PressureController_state_id
080003a8 T PS_init
20000024 B PS_pressure
20000014 D PS_state
20000028 B PS_state_id
080000d8 W PVD_handler
080000d8 W RCC_handler
0800024c T Set_Alarm_actuator
08000164 T STATE_AlarmActuator_alarmOFF
0800013c T STATE_AlarmActuator_alarmON
08000126 T STATE_AlarmActuator_waiting
080001a8 T STATE_AlarmMonitor_alarmOFF
080001c0 T STATE_AlarmMonitor_alarmON
080001e4 T STATE_AlarmMonitor_waiting
08000370 T STATE_PressureController_busy
08000358 T STATE_PressureController_idle
080003b4 T STATE_PS_reading
080003f0 T STATE_PS_waiting
080000d8 W SVCall_handler
080000d8 W SysTick_handler
080000d8 W TAMPER_handler
2000000c D threshold
08000000 T vectors
080000d8 W WWDG_handler
```