Nov 2021

First Term (Final Project 1)

Eng. Mazen Talaat
My profile: https://bit.ly/3DUmnyz

Mastering embedded system online diploma by Eng. Keroles Shenouda
www.learn-in-depth.com

Table of contents

Case Study

A “client” expects to deliver the software of the following system:

- 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.

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

Method

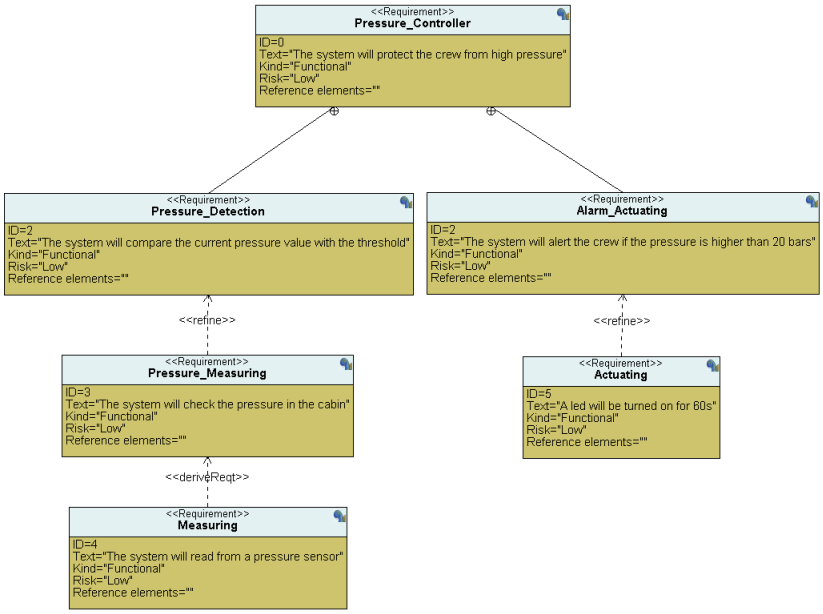
I will use the waterfall method which consists of

1. Requirements.
2. Analysis.
3. Design.
4. Coding.
5. Testing.
6. Deployment.

Idea

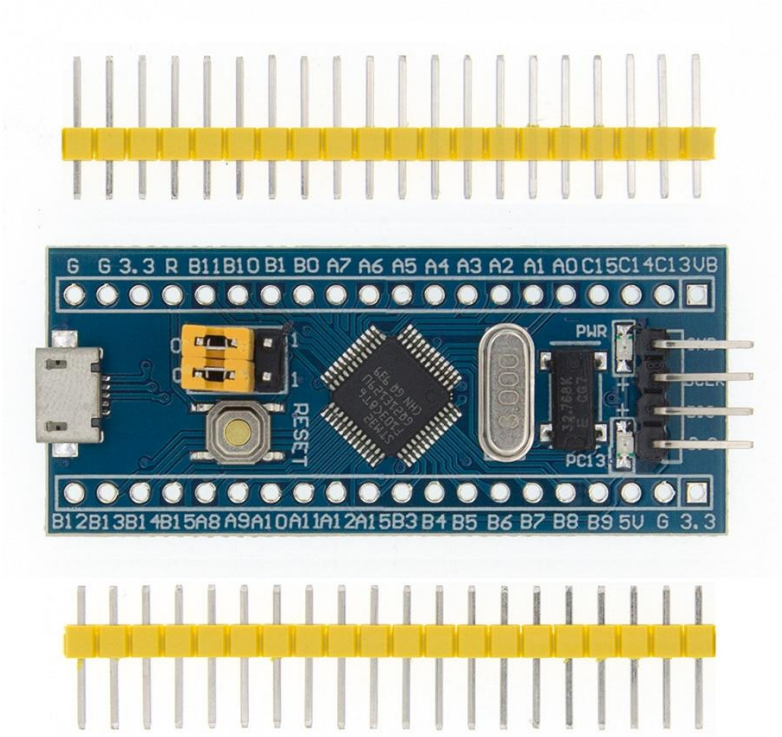
1. Get the pressure sensor data.
2. If the pressure is greater than or equal to the threshold = 20 bars.
    a. Start the alarm for 60 seconds.
3. If the pressure is less than the threshold = 20 bars.
    a. Continue.
4. Repeat.

Requirements Diagram



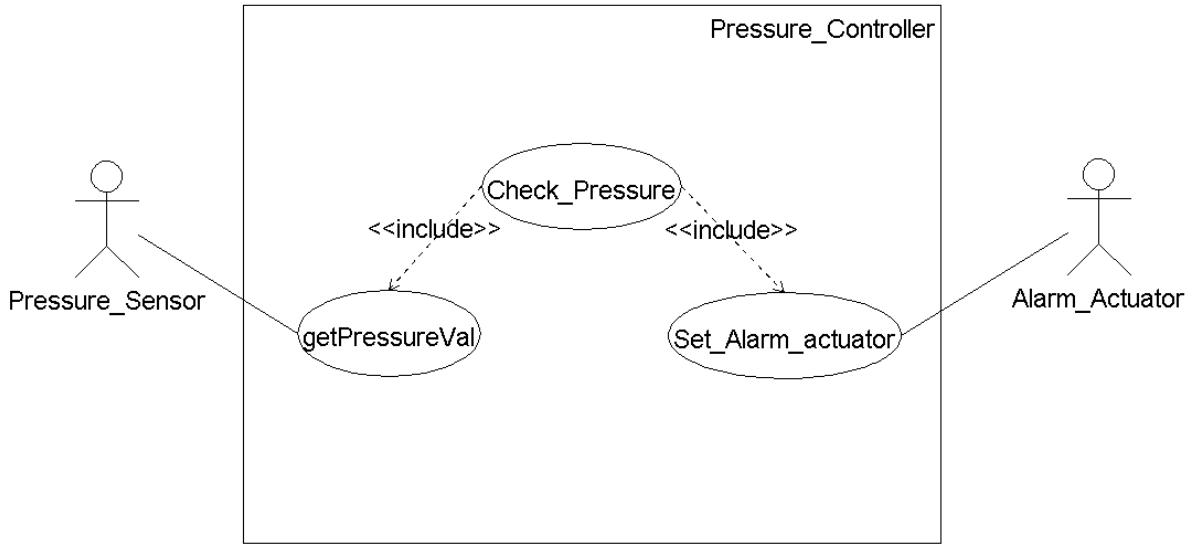
Space Exploration

We will use ARM STM32 board as it meets our requirements and available in stores.

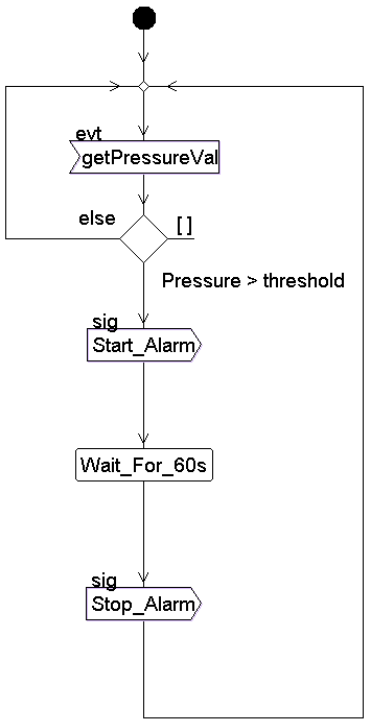


System Analysis

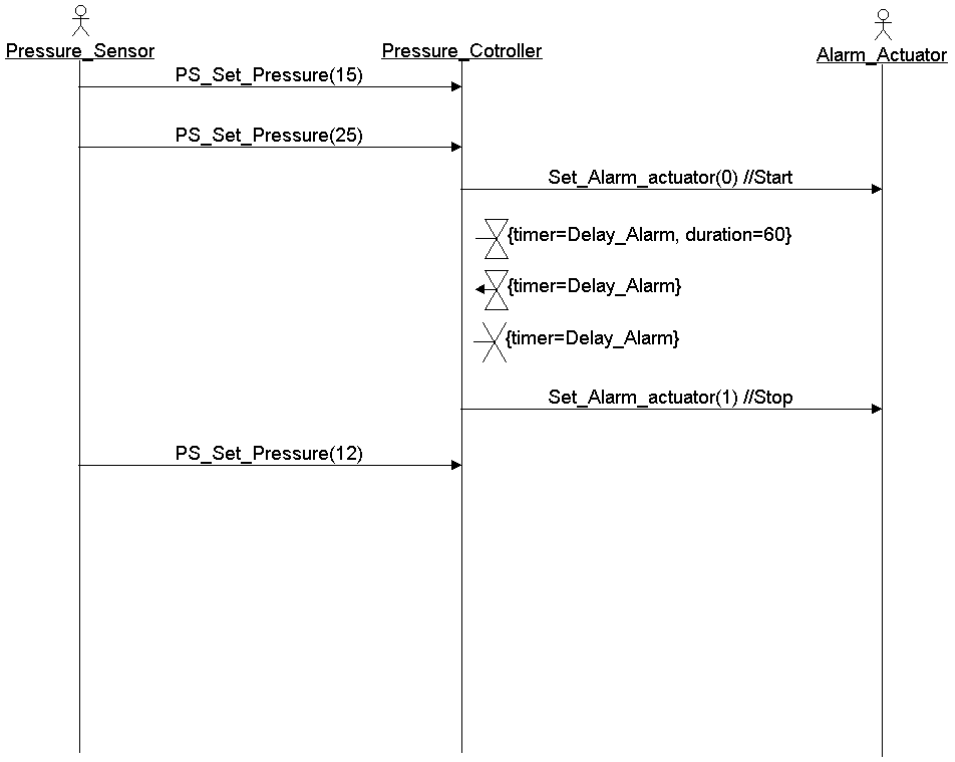
Use case diagram



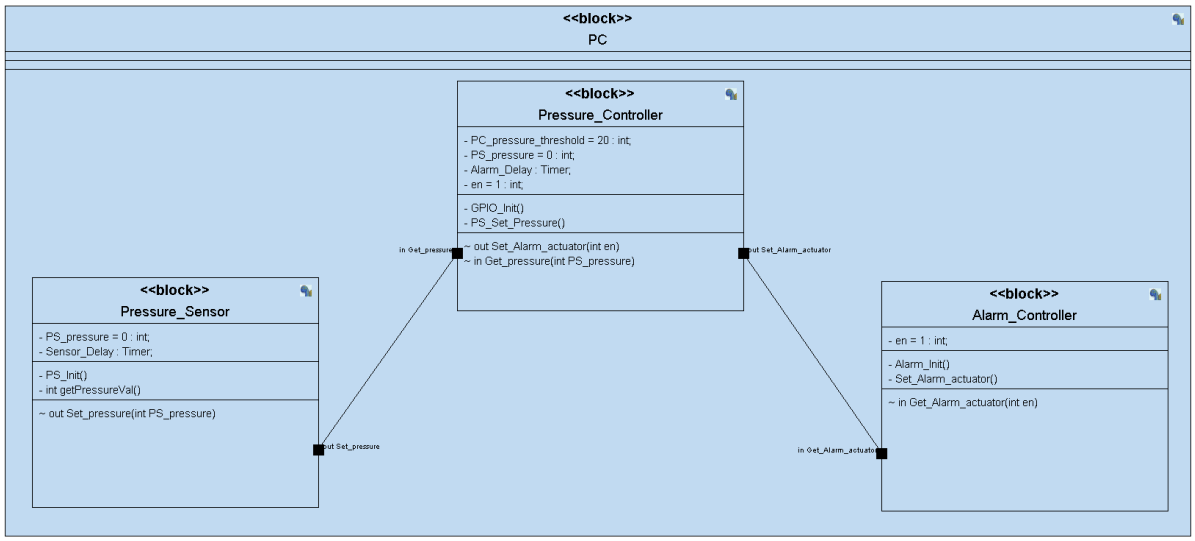
Activity diagram



Sequence diagram



Block Diagram



We will have 3 blocks:

1. Pressure sensor.
2. Pressure controller.
3. Alarm controller.

Pressure sensor

This block will have the pressure variable and will set it by the function getPressureVal.

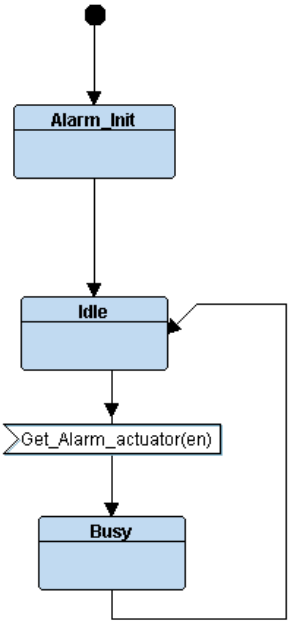
Alarm controller

This block will have the control over the alarm actuator using the function Set_Alarm_actuator.

Pressure controller

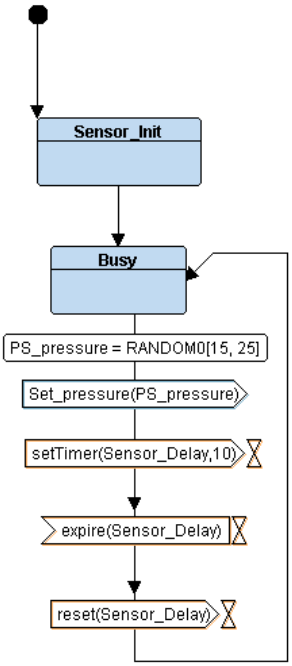
This block will have the control algorithm by retrieving, comparing and sending.

Alarm state machine



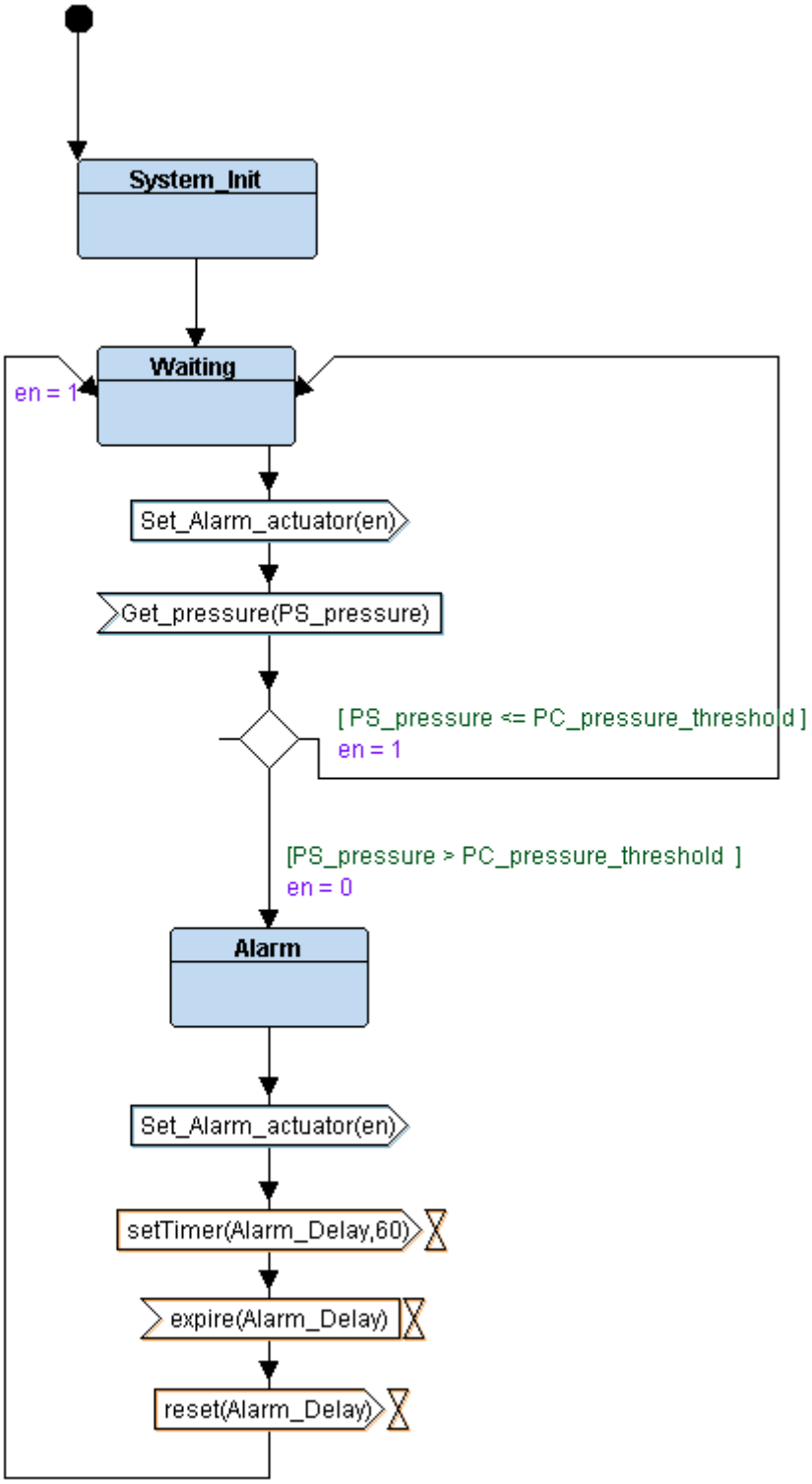
The alarm actuator will be initialized and begin at idle state and will start or stop depending on the "en" variable.

Pressure sensor state machine



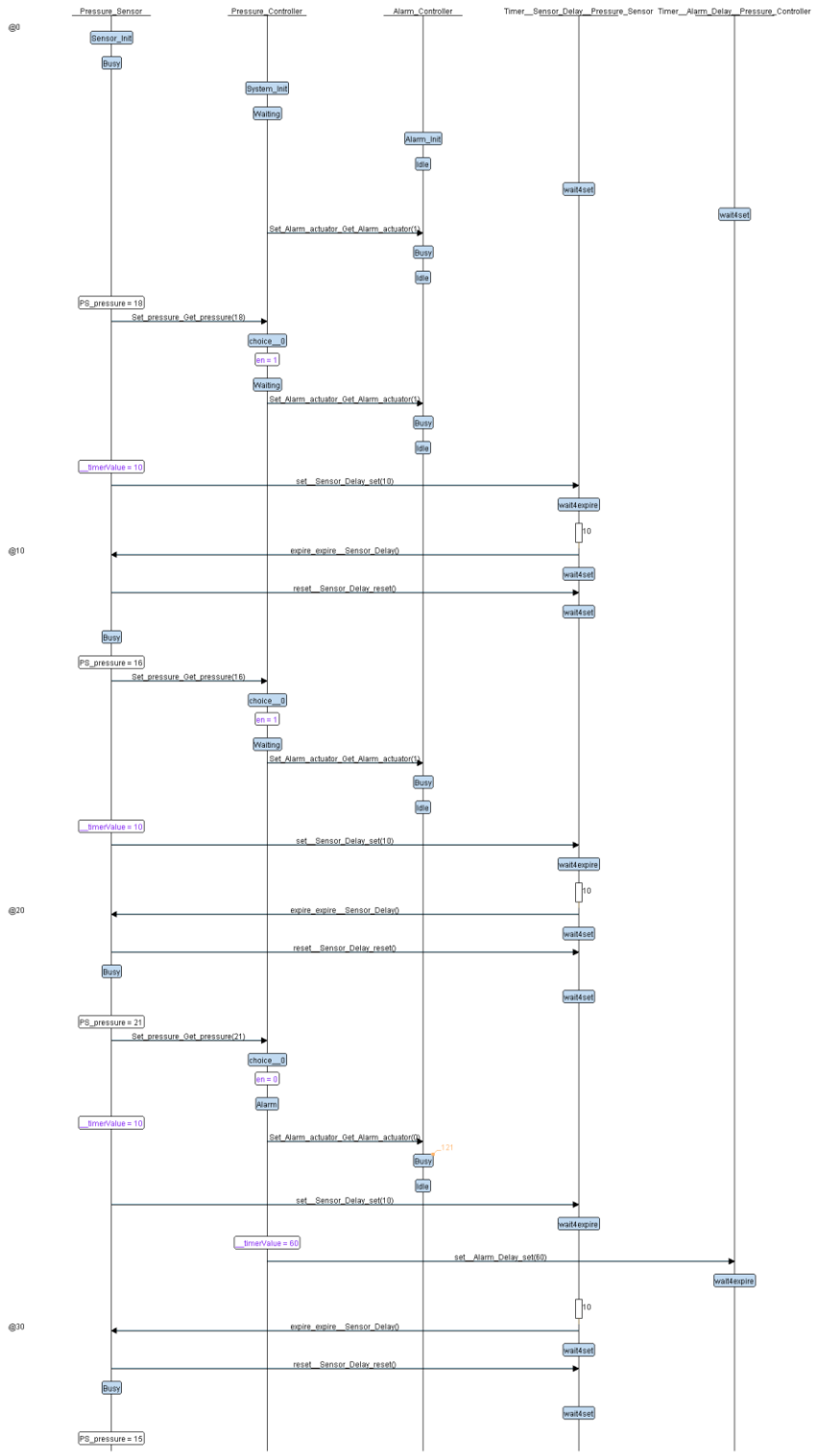
The pressure sensor will be initialized and begin at busy state and keep sending the pressure value every 10 sec.

Pressure controller state machine



The system will begin at the waiting state then sets the alarm off and get the pressure from the sensor then checks whether the pressure is greater than the threshold or not.
If it is greater, it will switch to the alarm state and set the alarm on for 60 seconds then repeat the process.

Final Simulation



All blocks started with initializing, then busy for sensor, waiting for the controller and idle for the alarm.
Then, the controller sets the alarm off and get the pressure from the sensor which is equal to 18. It waits for 10 seconds to get the new pressure from sensor.
When the pressure is equal to 21, it switches to the alarm state and set the alarm on for 60 seconds, after that it will read from the sensor again.

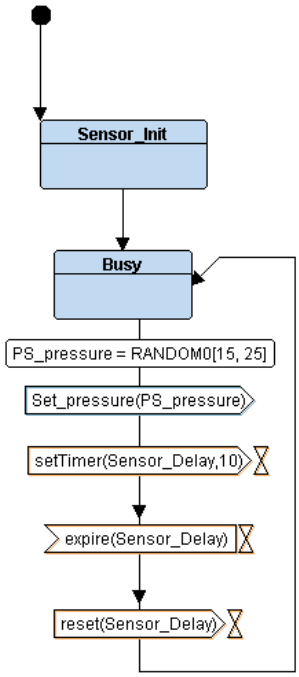
Code

driver.h/c

```c
#define uint32_t unsigned int

#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();
```

```c
#include "driver.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;
}
```

Pressure_Senor.h/c

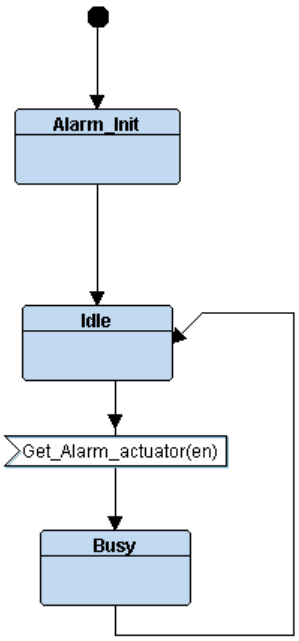



```c
#include "Pressure_Sensor.h"

int PS_pressure = 0;

void (*P_PS_state)() = 0;

void PS_Init(){
    /* Do some stuff */
}

STATE_Fn_Define(PS_Busy){
    PS_State_ID = PS_Busy;
    PS_pressure = getPressureVal();
    PS_Set_Pressure(PS_pressure);
    P_PS_state = STATE(PS_Busy);
    Delay(100000);
}
```

```c
#ifndef PRESSURE_SENSOR_H_
#define PRESSURE_SENSOR_H_

#define STATE_Fn_Define(_StatFunction_) void ST_##_StatFunction_()
#define STATE(_StatFunction_) ST_##_StatFunction_

/* States */
enum{
    PS_Busy
}PS_State_ID;

/* State functions */
STATE_Fn_Define(PS_Busy);

/* Global pointer to fn */
extern void (*P_PS_state)();

void PS_Init();

#endif /* PRESSURE_SENSOR_H_ */
```

Alarm_Controller.h/c

```c
#include "Alarm_Controller.h"

void (*P_AC_state)() = 0;

void AC_Init(){
    Set_Alarm_actuator(1);
}

STATE_Fn_Define(AC_Idle){
    AC_State_ID = AC_Idle;
    P_AC_state = STATE(AC_Idle);
}

STATE_Fn_Define(AC_Busy){
    AC_State_ID = AC_Busy;
    P_AC_state = STATE(AC_Idle);
}
```

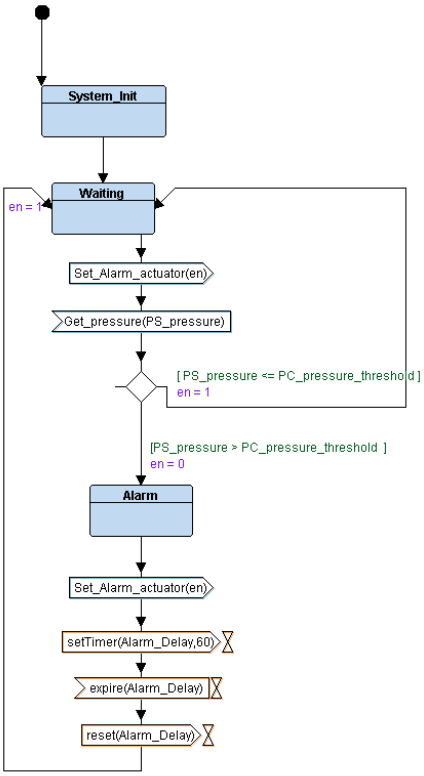



```c
#ifndef ALARM_CONTROLLER_H_
#define ALARM_CONTROLLER_H_

#define STATE_Fn_Define(_StatFunction_) void ST_##_StatFunction_()
#define STATE(_StatFunction_) ST_##_StatFunction_

/* States */
enum{
    AC_Idle,
    AC_Busy
}AC_State_ID;

/* State functions */
STATE_Fn_Define(AC_Idle);
STATE_Fn_Define(AC_Busy);

/* Global pointer to fn */
extern void (*P_AC_state)();

void AC_Init();

#endif /* ALARM_CONTROLLER_H_ */
```

Pressure_Controller.h/c

```c
#include "Pressure_Controller.h"

/* Variables */
int PC_pressure = 0;
int PC_pressure_threshold = 20;

void (*P_PC_state)() = 0;

void PS_Set_Pressure(int p){
    PC_pressure = p;
    (PC_pressure <= PC_pressure_threshold) ? (P_PC_state = STATE(PC_Waiting)) : (P_PC_state = STATE(PC_Alarm));
}

STATE_Fn_Define(PC_Waiting){
    PC_State_ID = PC_Waiting;
    Set_Alarm_actuator(1);
}

STATE_Fn_Define(PC_Alarm){
    PC_State_ID = PC_Alarm;
    Set_Alarm_actuator(0);
    /* Wait 60s */
    Delay(6000000);
    Set_Alarm_actuator(1);
}
```

```c
#ifndef PRESSURE_CONTROLLER_H_
#define PRESSURE_CONTROLLER_H_

#define STATE_Fn_Define(_StatFunction_) void ST_##_StatFunction_()
#define STATE(_StatFunction_) ST_##_StatFunction_

/* States */
enum{
    PC_Waiting,
    PC_Alarm
}PC_State_ID;

/* State functions */
STATE_Fn_Define(PC_Waiting);
STATE_Fn_Define(PC_Alarm);

/* Global pointer to fn */
extern void (*P_PC_state)();

#endif /* PRESSURE_CONTROLLER_H_ */
```



main.c

```c
/*
 * main.c
 *
 * Created on: Nov 26, 2021
 * Author: Mazen Talaat
 */

#include "driver.h"
#include "Pressure_Controller.h"
#include "Alarm_Controller.h"
#include "Pressure_Sensor.h"

/* Pressure_Controller states: waiting, alarm *
 * Pressure_Sensor     states: busy           *
 * Alarm_Controller    states: idle, busy     */

void setup(){
    AC_Init();
    PS_Init();
    P_PC_state = STATE(PC_Waiting);
    P_AC_state = STATE(AC_Idle);
    P_PS_state = STATE(PS_Busy);
}

int main (){
    GPIO_INITIALIZATION();
    setup();
    while (1)
    {
        P_PS_state();
        P_PC_state();
        P_AC_state();
        /* wait 10s */
        Delay(100000);
    }
    return 0;
}
```

startup.c

```c
#include <stdint.h>

/* Functions declaration */
extern int main();
void Reset_Handler();
void Default_Handler()
{
    Reset_Handler();
}
void NMI_Handler        () __attribute__((weak, alias ("Default_Handler")));;
void H_Fault_Handler    () __attribute__((weak, alias ("Default_Handler")));;

/* Variables declaration */
static unsigned long Stack_top[256]; /*256*4 = 1024B */

extern unsigned int _S_data;
extern unsigned int _S_bss;
extern unsigned int _E_data;
extern unsigned int _E_bss;
extern unsigned int _E_text;

/* Vector table declaration */
/* Array of pointers to a function returns nothing */
void (* const g_p_fn_Vectors[])() __attribute__((section(".vectors"))) =
{
    (void (*)()) ((unsigned long)Stack_top + sizeof(Stack_top)),
    &Reset_Handler,
    &NMI_Handler,
    &H_Fault_Handler
};

/* Reset function body */
void Reset_Handler(void)
{
    unsigned int data_size = (unsigned char*)&_E_data - (unsigned char*)&_S_data;
    unsigned char* P_src = (unsigned char*)&_E_text;
    unsigned char* P_dst = (unsigned char*)&_S_data;
    int i;
    /* copying data from ROM to RAM */
    for(i=0; i<data_size; i++){
        *((unsigned char*)P_dst++) = *((unsigned char*)P_src++);
    }
    unsigned int bss_size = (unsigned char*)&_E_bss - (unsigned char*)&_S_bss;
    /*Initializing .bss with zeros */
    for(i=0; i<bss_size; i++){
        *((unsigned char*)P_dst++) = (unsigned char)0;
    }
    main();
}
```

linker_script.ld

```
MEMORY
{
    FLASH(RX) : o = 0x08000000 , l = 128K
    SRAM(RWX) : o = 0x20000000 , l = 20K
}

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

    .data :
    {
            _S_data = . ;
            *(.data)
            . = ALIGN(4);
            _E_data = . ;

    }> SRAM AT > FLASH

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

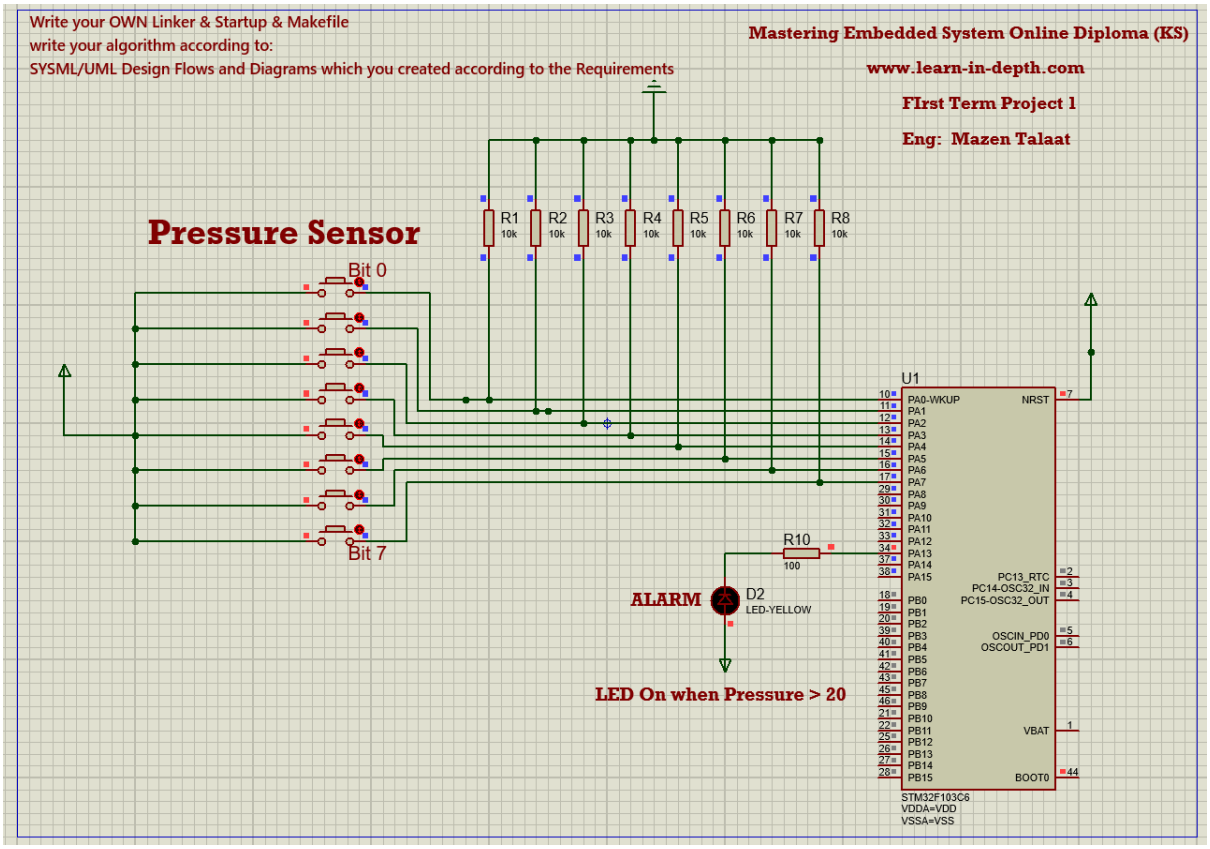
map_file.map

```
Memory Configuration

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

Linker script and memory map


.text           0x08000000      0x3b8
 *(.vectors*)
 .vectors       0x08000000       0x10 obj/startup.o
                0x08000000                g_p_fn_Vectors
 *(.text*)
 .text          0x08000010       0x68 obj/Alarm_Controller.o
                0x08000010                AC_Init
                0x08000020                ST_AC_Idle
                0x0800004c                ST_AC_Busy
 .text          0x08000078      0x10c obj/driver.o
                0x08000078                Delay
                0x0800009c                getPressureVal
                0x080000b4                Set_Alarm_actuator
                0x08000104                GPIO_INITIALIZATION
 .text          0x08000184       0x84 obj/main.o
                0x08000184                setup
                0x080001c8                main
 .text          0x08000208       0xa8 obj/Pressure_Controller.o
                0x08000208                PS_Set_Pressure
                0x08000264                ST_PC_Waiting
                0x08000280                ST_PC_Alarm
 .text          0x080002b0       0x54 obj/Pressure_Sensor.o
                0x080002b0                PS_Init
                0x080002bc                ST_PS_Busy
 .text          0x08000304       0xb4 obj/startup.o
                0x08000304                H_Fault_Handler
                0x08000304                Default_Handler
                0x08000304                NMI_Handler
                0x08000310                Reset_Handler
 *(.rodata)
                0x080003b8                _E_text = .

.data           0x20000000        0x4 load address 0x080003b8
                0x20000000                _S_data = .
 *(.data)
 .data          0x20000000        0x0 obj/Alarm_Controller.o
 .data          0x20000000        0x0 obj/driver.o
 .data          0x20000000        0x0 obj/main.o
 .data          0x20000000        0x4 obj/Pressure_Controller.o
                0x20000000                PC_pressure_threshold
 .data          0x20000004        0x0 obj/Pressure_Sensor.o
 .data          0x20000004        0x0 obj/startup.o
                0x20000004                . = ALIGN (0x4)
                0x20000004                _E_data = .

.igot.plt       0x20000004        0x0 load address 0x080003bc
 .igot.plt      0x00000000        0x0 obj/Alarm_Controller.o

.bss            0x20000004      0x417 load address 0x080003bc
                0x20000004                _S_bss = .
 *(.bss*)
 .bss           0x20000004        0x4 obj/Alarm_Controller.o
                0x20000004                P_AC_state
 .bss           0x20000008        0x0 obj/driver.o
 .bss           0x20000008        0x0 obj/main.o
 .bss           0x20000008        0x8 obj/Pressure_Controller.o
                0x20000008                PC_pressure
                0x2000000c                P_PC_state
 .bss           0x20000010        0x8 obj/Pressure_Sensor.o
                0x20000010                PS_pressure
                0x20000014                P_PS_state
 .bss           0x20000018      0x400 obj/startup.o
                0x20000418                . = ALIGN (0x4)
                0x20000418                _E_bss = .
 COMMON         0x20000418        0x1 obj/Alarm_Controller.o
                0x20000418                AC_State_ID
 COMMON         0x20000419        0x2 obj/main.o
                0x20000419                PC_State_ID
                0x2000041a                PS_State_ID
```

MakeFile

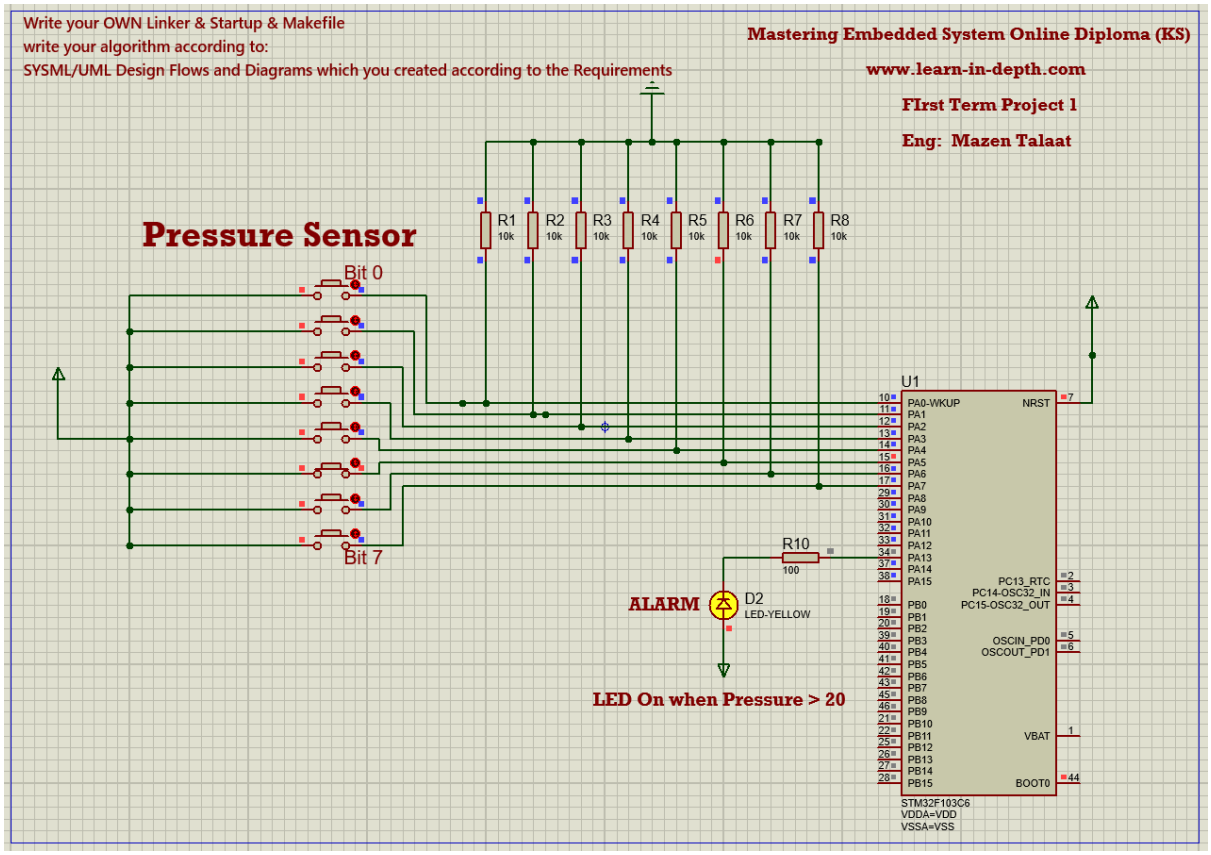
```
CC     = arm-none-eabi-
CFLAGS = -mcpu=cortex-m3 -mthumb -gdwarf-2

SRCDIR = src
OBJDIR = obj
INCDIR = include
BINDIR = bin

LIBS=

INCS = -I $(INCDIR)
SRC  = $(wildcard $(SRCDIR)/*.c)
OBJ  = $(SRC:$(SRCDIR)/%.c=$(OBJDIR)/%.o)

Project   = Project1_Pressure_Controller
P_BIN_DIR = $(BINDIR)/$(Project)

all: $(P_BIN_DIR).bin $(P_BIN_DIR).hex
    @echo "========= Everything Done =========="

$(OBJDIR)/%.o: $(SRCDIR)/%.c
    $(CC)gcc.exe -c $(CFLAGS) $(INCS) $< -o $@
    @echo "========= .C to .O Done =========="

$(P_BIN_DIR).elf: $(OBJ)
    $(CC)ld.exe -T linker_script.ld $(LIBS) $^ -o $@ -Map=$(BINDIR)/map_file.map
    @echo "========= Linking Done =========="

$(P_BIN_DIR).bin: $(P_BIN_DIR).elf
    $(CC)objcopy.exe -O binary $< $@
    @echo "========= Binary Out Done =========="

$(P_BIN_DIR).hex: $(P_BIN_DIR).elf
    $(CC)objcopy.exe -O ihex $< $@
    @echo "========= Hex Out Done =========="

clean_all:
    rm -rf $(OBJDIR)/*.o $(BINDIR)/*.elf $(BINDIR)/*.bin $(BINDIR)/*.map $(BINDIR)/*.hex
    @echo "========= Cleaned Everything =========="
clean:
    rm -rf $(BINDIR)/*.elf $(BINDIR)/*.bin
    @echo "========= Cleaned .elf and .bin =========="
```

Proteus Simulation



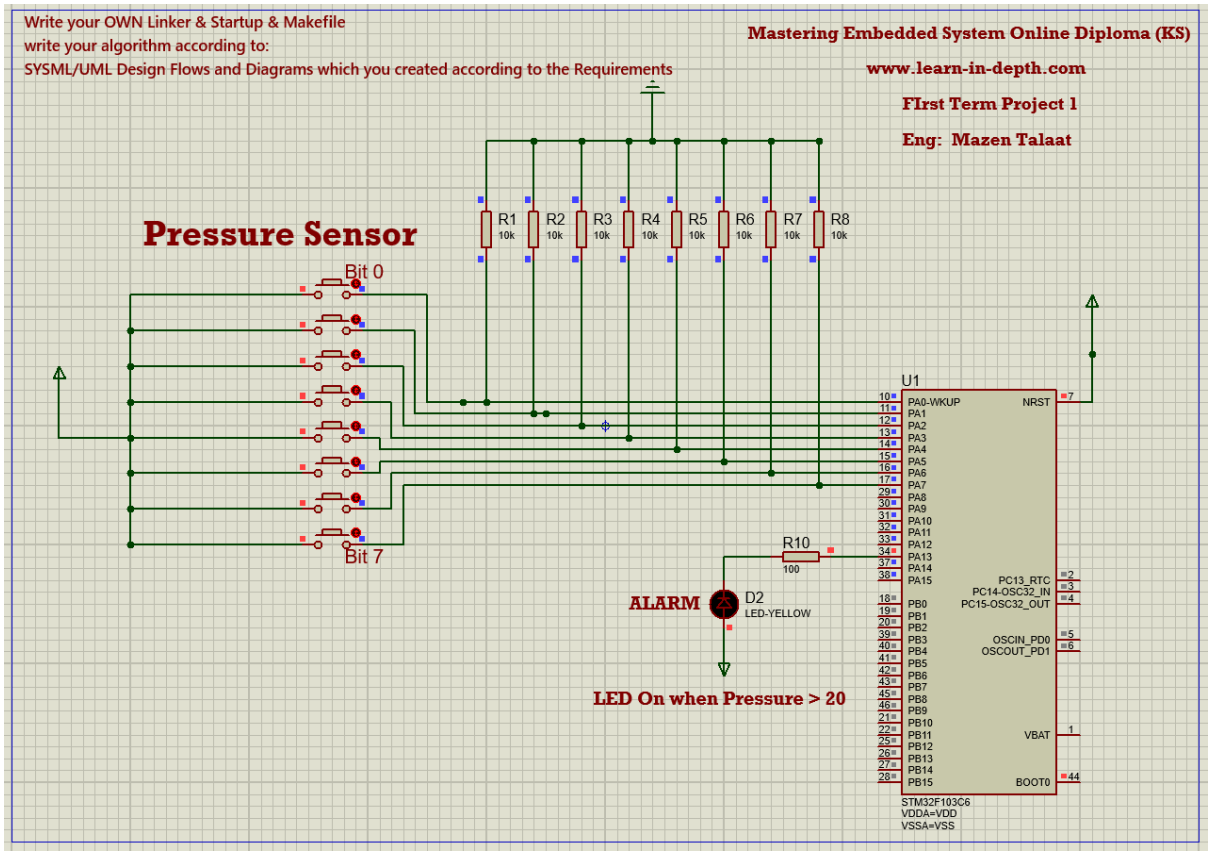
Name	Address	Value
PC_State_ID	20000419	PC_Waiting (0)
AC_State_ID	20000418	AC_Idle (0)
PS_State_ID	2000041A	PS_Busy (0)
PC_State_ID	20000419	PC_Waiting (0)
PC_pressure	20000008	0
PC_pressure_threshold	20000000	20
PS_State_ID	2000041A	PS_Busy (0)
PS_pressure	20000010	0
⊞Stack_top	20000018	dword[256]
AC_State_ID	20000418	AC_Idle (0)
nCount	BP+12 = @200003E4	3827

Pressure = 0, LED Off



Name	Address	Value
PC_State_ID	20000419	PC_Alarm (1)
AC_State_ID	20000418	AC_Idle (0)
PS_State_ID	2000041A	PS_Busy (0)
PC_State_ID	20000419	PC_Alarm (1)
PC_pressure	20000008	32
PC_pressure_threshold	20000000	20
PS_State_ID	2000041A	PS_Busy (0)
PS_pressure	20000010	32
⊞Stack_top	20000018	dword[256]
AC_State_ID	20000418	AC_Idle (0)
nCount	BP+12 = @200003DC	5830478

Pressure = 32, LED On



Name	Address	Value
PC_State_ID	20000419	PC_Waiting (0)
AC_State_ID	20000418	AC_Idle (0)
PS_State_ID	2000041A	PS_Busy (0)
PC_State_ID	20000419	PC_Waiting (0)
PC_pressure	20000008	0
PC_pressure_threshold	20000000	20
PS_State_ID	2000041A	PS_Busy (0)
PS_pressure	20000010	0
⊞Stack_top	20000018	dword[256]
AC_State_ID	20000418	AC_Idle (0)
nCount	BP+12 = @200003E4	30505

Pressure = 0, LED Off

After some time (nCount)

Symbols table

```
MINGW64:/d/Studying/Embedded_KS/Unit 5/P1_Pressure_Detection/Code/...

MR_MaZeN@user-PC MINGW64 /d/Studying/Embedded_KS/Unit 5/P1_Pressure_Detection/Co
de/bin
$ arm-none-eabi-nm.exe Project1_Pressure_Controller.elf
20000418 B _E_bss
20000004 D _E_data
080003b8 T _E_text
20000004 B _S_bss
20000000 D _S_data
08000010 T AC_Init
20000418 B AC_State_ID
08000304 T Default_Handler
08000078 T Delay
08000000 T g_p_fn_Vectors
0800009c T getPressureVal
08000104 T GPIO_INITIALIZATION
08000304 W H_Fault_Handler
080001c8 T main
08000304 W NMI_Handler
20000004 B P_AC_state
2000000c B P_PC_state
20000014 B P_PS_state
20000008 B PC_pressure
20000000 D PC_pressure_threshold
20000419 B PC_State_ID
080002a4 T PS_Init
20000010 B PS_pressure
080001fc T PS_Set_Pressure
2000041a B PS_State_ID
08000310 T Reset_Handler
080000b4 T Set_Alarm_actuator
08000184 T setup
0800004c T ST_AC_Busy
08000020 T ST_AC_Idle
08000274 T ST_PC_Alarm
08000258 T ST_PC_Waiting
080002b0 T ST_PS_Busy
20000018 b Stack_top

MR_MaZeN@user-PC MINGW64 /d/Studying/Embedded_KS/Unit 5/P1_Pressure_Detection/Co
de/bin
$ |
```

Sections

main.o

```
$ arm-none-eabi-objdump.exe -h main.o

main.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000078  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  000000ac  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000ac  2**0
                  ALLOC
  3 .debug_info   000000fe  00000000  00000000  000000ac  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 0000009a  00000000  00000000  000001aa  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    00000058  00000000  00000000  00000244  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  0000029c  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   000000a3  00000000  00000000  000002bc  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str    000000cd  00000000  00000000  0000035f  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000012  00000000  00000000  0000042c  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  0000043e  2**0
                  CONTENTS, READONLY
 11 .debug_frame  00000048  00000000  00000000  00000474  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

Alarm_Controller.o

```
$ arm-none-eabi-objdump.exe -h Alarm_Controller.o

Alarm_Controller.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000068  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  0000009c  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000004  00000000  00000000  0000009c  2**2
                  ALLOC
  3 .debug_info   000000c6  00000000  00000000  0000009c  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 0000009a  00000000  00000000  00000162  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    00000084  00000000  00000000  000001fc  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  00000280  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   0000006e  00000000  00000000  000002a0  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str    000000b5  00000000  00000000  0000030e  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000012  00000000  00000000  000003c3  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  000003d5  2**0
                  CONTENTS, READONLY
 11 .debug_frame  0000005c  00000000  00000000  00000408  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

driver.o

```
$ arm-none-eabi-objdump.exe -h driver.o

driver.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000010c  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000000  00000000  00000000  00000140  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  00000140  2**0
                  ALLOC
  3 .debug_info   000000ab  00000000  00000000  00000140  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 00000085  00000000  00000000  000001eb  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    000000c8  00000000  00000000  00000270  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  00000338  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   00000058  00000000  00000000  00000358  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str    00000096  00000000  00000000  000003b0  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000012  00000000  00000000  00000446  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  00000458  2**0
                  CONTENTS, READONLY
 11 .debug_frame  00000078  00000000  00000000  0000048c  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

Pressure_Controller.o

```
$ arm-none-eabi-objdump.exe -h Pressure_Controller.o

Pressure_Controller.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000000a8  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000004  00000000  00000000  000000dc  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000008  00000000  00000000  000000e0  2**2
                  ALLOC
  3 .debug_info   00000123  00000000  00000000  000000e0  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 000000c0  00000000  00000000  00000203  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    00000090  00000000  00000000  000002c3  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  00000353  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   00000083  00000000  00000000  00000373  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str    000000f0  00000000  00000000  000003f6  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000012  00000000  00000000  000004e6  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  000004f8  2**0
                  CONTENTS, READONLY
 11 .debug_frame  00000064  00000000  00000000  0000052c  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

Pressure_Sensor.o

```
$ arm-none-eabi-objdump.exe -h Pressure_Sensor.o

Pressure_Sensor.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000060  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  00000094  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000008  00000000  00000000  00000094  2**2
                  ALLOC
  3 .debug_info   000000e3  00000000  00000000  00000094  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 000000ad  00000000  00000000  00000177  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    00000058  00000000  00000000  00000224  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  0000027c  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   0000006a  00000000  00000000  0000029c  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str    000000bf  00000000  00000000  00000306  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000012  00000000  00000000  000003c5  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  000003d7  2**0
                  CONTENTS, READONLY
 11 .debug_frame  00000044  00000000  00000000  0000040c  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

startup.o

```
$ arm-none-eabi-objdump.exe -h startup.o

startup.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000000b4  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  000000e8  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000400  00000000  00000000  000000e8  2**2
                  ALLOC
  3 .vectors      00000010  00000000  00000000  000000e8  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA
  4 .debug_info   00000183  00000000  00000000  000000f8  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  5 .debug_abbrev 000000c8  00000000  00000000  0000027b  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_loc    00000064  00000000  00000000  00000343  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_aranges 00000020  00000000  00000000  000003a7  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_line   0000006c  00000000  00000000  000003c7  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  9 .debug_str    0000015d  00000000  00000000  00000433  2**0
                  CONTENTS, READONLY, DEBUGGING
 10 .comment      00000012  00000000  00000000  00000590  2**0
                  CONTENTS, READONLY
 11 .ARM.attributes 00000033  00000000  00000000  000005a2  2**0
                  CONTENTS, READONLY
 12 .debug_frame  0000004c  00000000  00000000  000005d8  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

Project.elf

```
$ arm-none-eabi-objdump.exe -h Project1_Pressure_Controller.elf

Project1_Pressure_Controller.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000003b8  08000000  08000000  00008000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000004  20000000  080003b8  00010000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000417  20000004  080003bc  00010004  2**2
                  ALLOC
  3 .debug_info   000005f8  00000000  00000000  00010004  2**0
                  CONTENTS, READONLY, DEBUGGING
  4 .debug_abbrev 000003ee  00000000  00000000  000105fc  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    000002f0  00000000  00000000  000109ea  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 000000c0  00000000  00000000  00010cda  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_line   000002c2  00000000  00000000  00010d9a  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .debug_str    000002a4  00000000  00000000  0001105c  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000011  00000000  00000000  00011300  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  00011311  2**0
                  CONTENTS, READONLY
 11 .debug_frame  00000210  00000000  00000000  00011344  2**2
                  CONTENTS, READONLY, DEBUGGING
```