Learn In Depth

Be Professional In Embedded System

Report on:

First Term Project 1: Pressure Controller

Prepared by:

Abanoub Salah

Submitted To:

Engr. Keroles Shenouda

Submitted Date:

9/17/2022

Contents

System Description

Client expects to deliver the software of the following system:

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

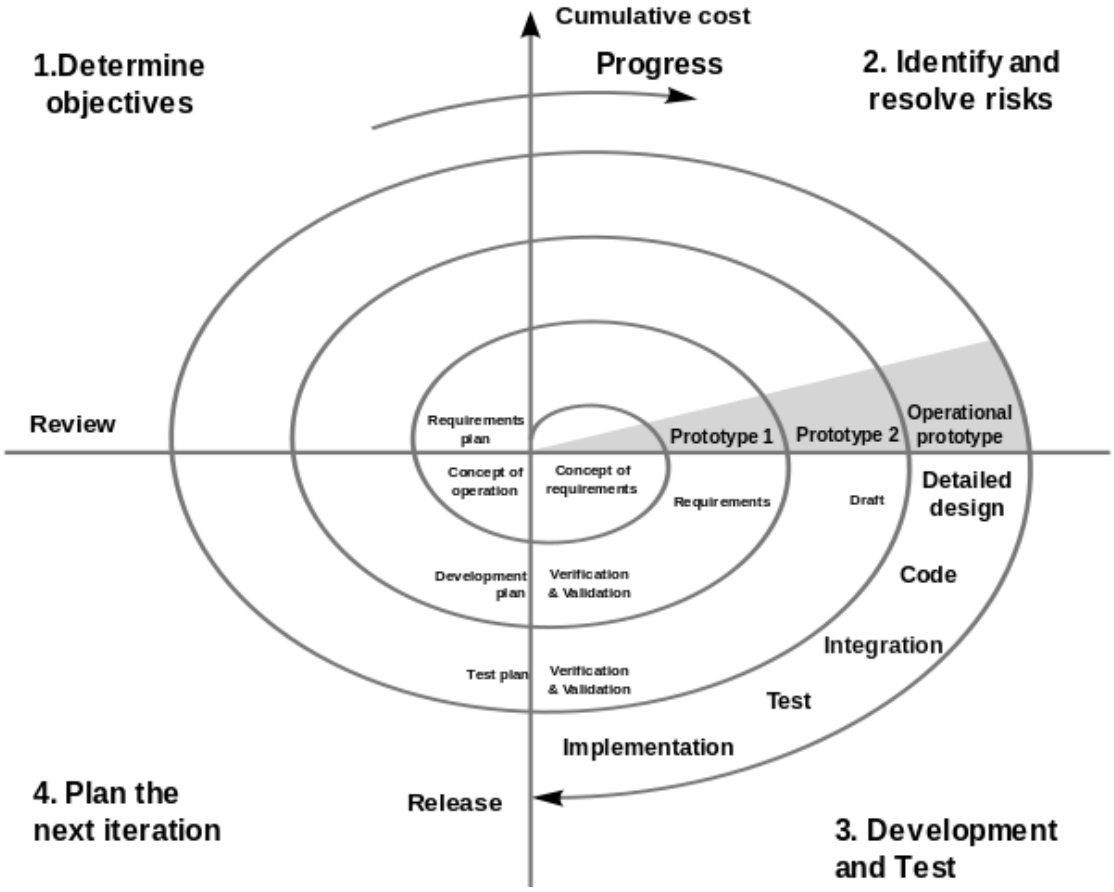
System Assumptions:

- System setup and shutdown procedures are not modeled.
- System maintenance are the user's responsibility
- System pressure sensor never fails
- System alarm never fail
- System never encounter power outage
- The "keep track of measured value" option is not modeled in the first version of the design

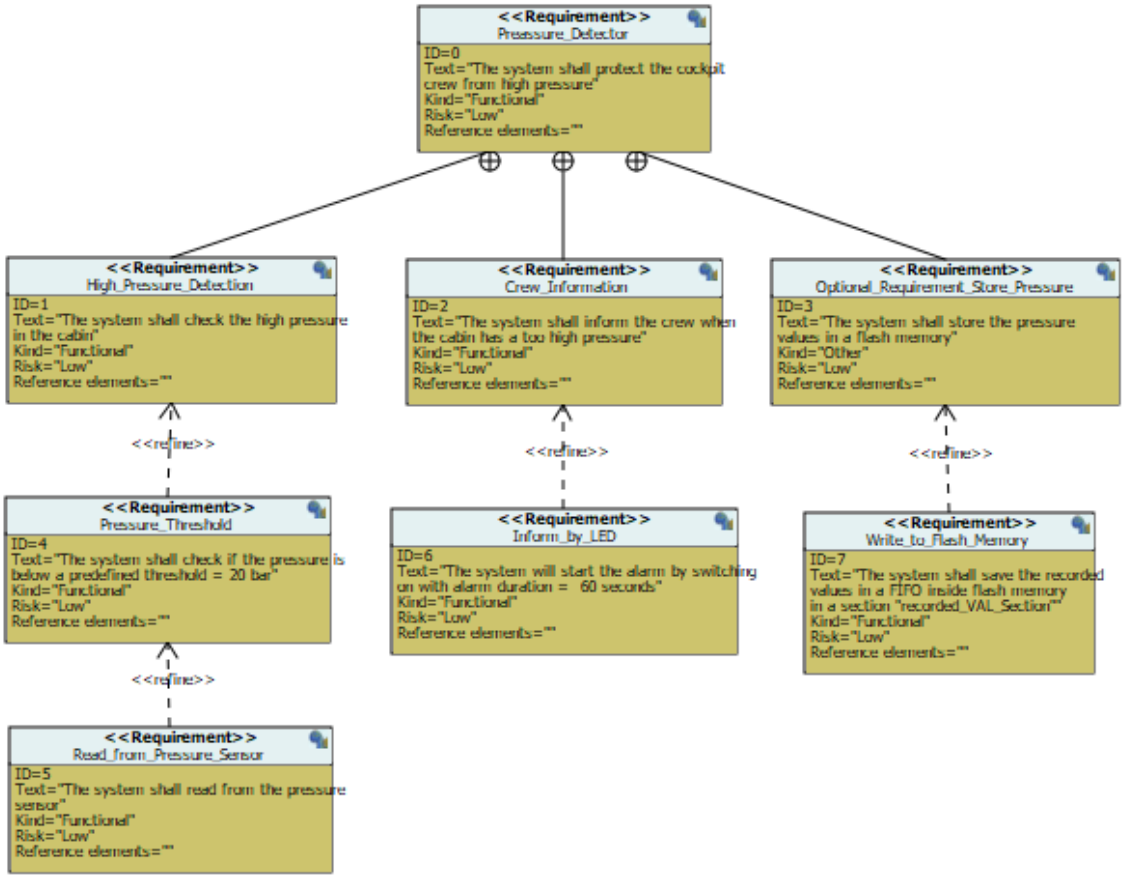
Methodology Used

Using spiral model justified as follow

- Avoidance of high risk
- Mission critical project
- Additional functionality can be added later
- Software produced early in the SDLC
- Cost is not an issue



Requirements Diagram

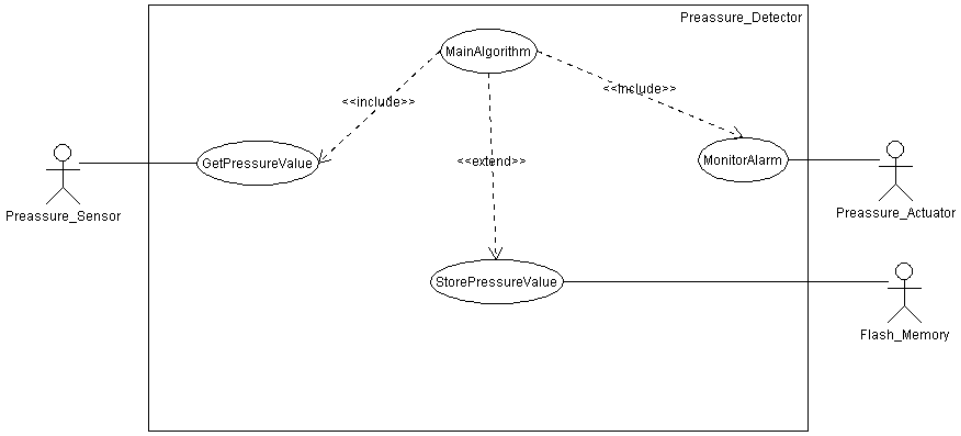


Space Exploration

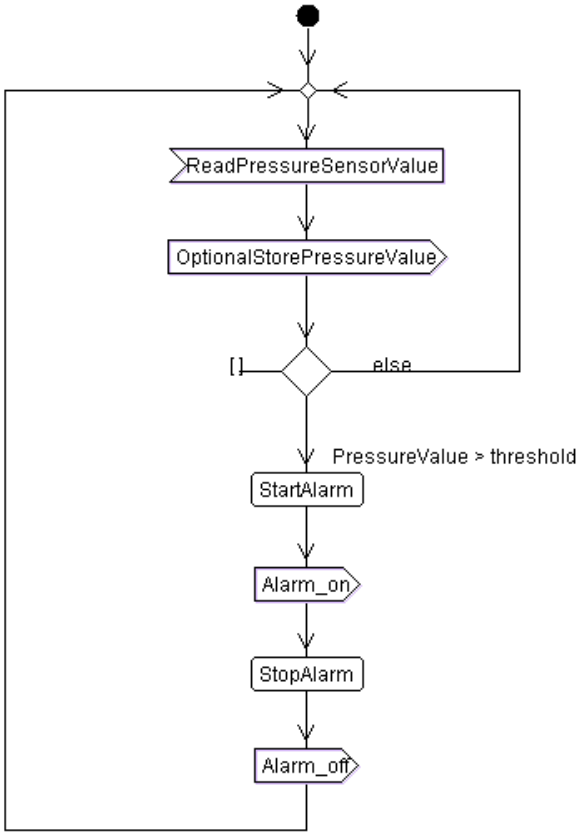
We will use STM32 ARM Cortex-M based µController. It is a high end 32bit µController capable of performing well for this project requirement.

System Analysis

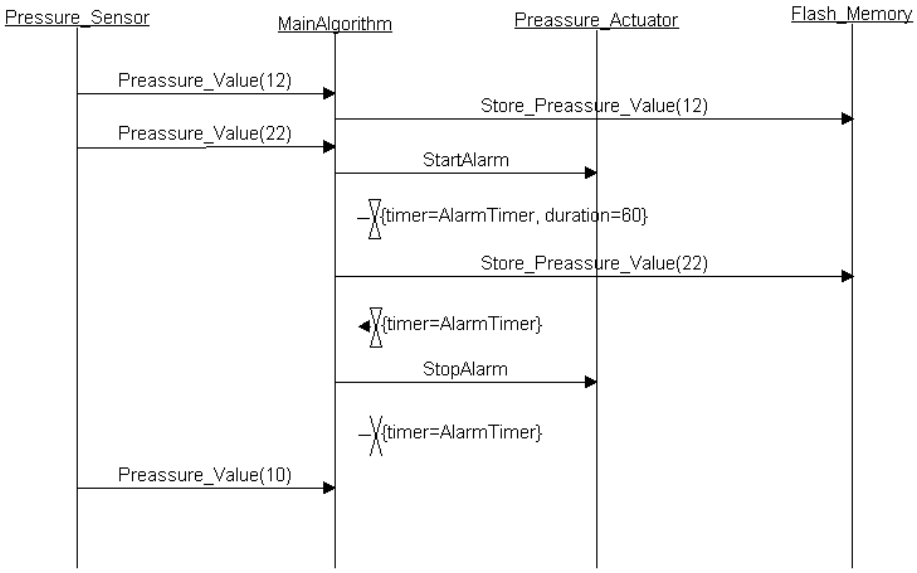
- **Use case Diagram**



- **Activity Diagram**

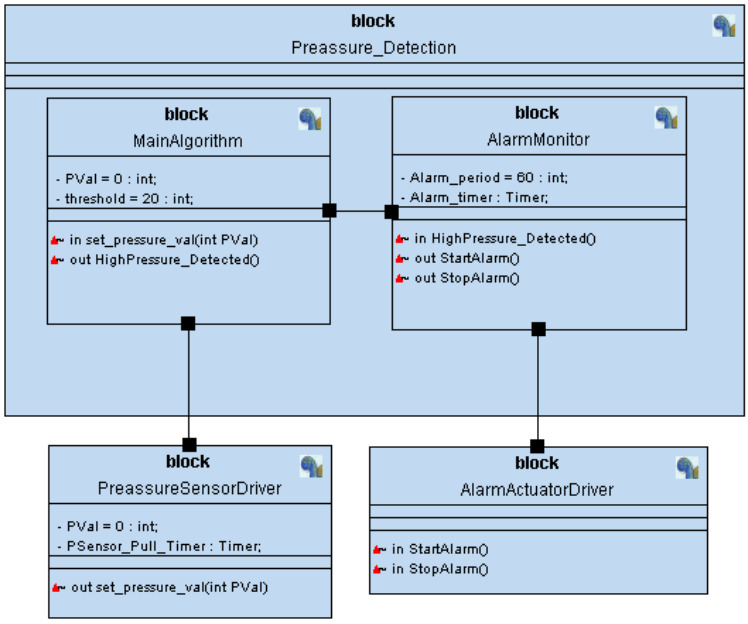


- **Sequence Diagram**

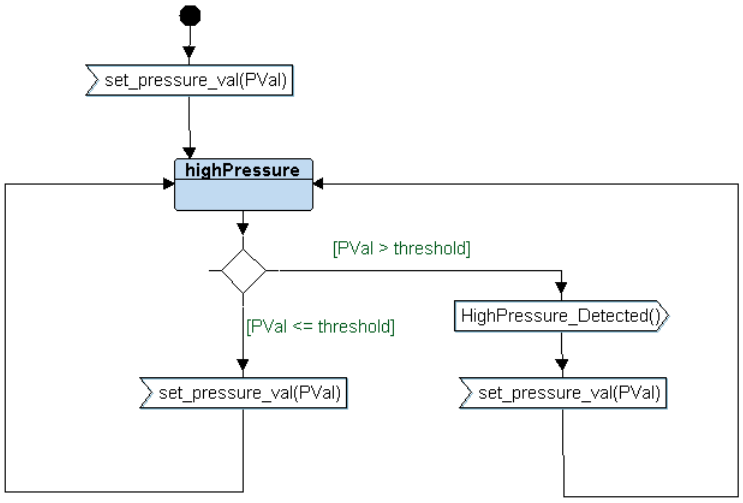


System Design

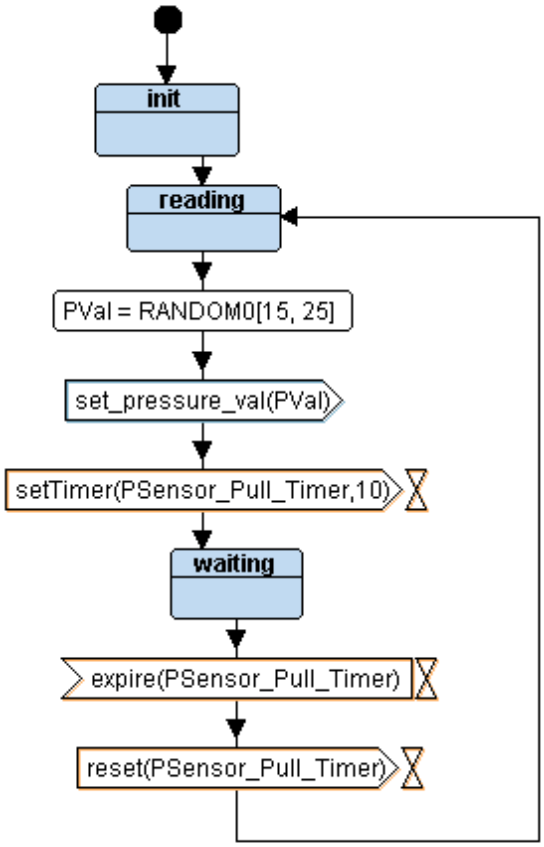
- **Block Diagram**



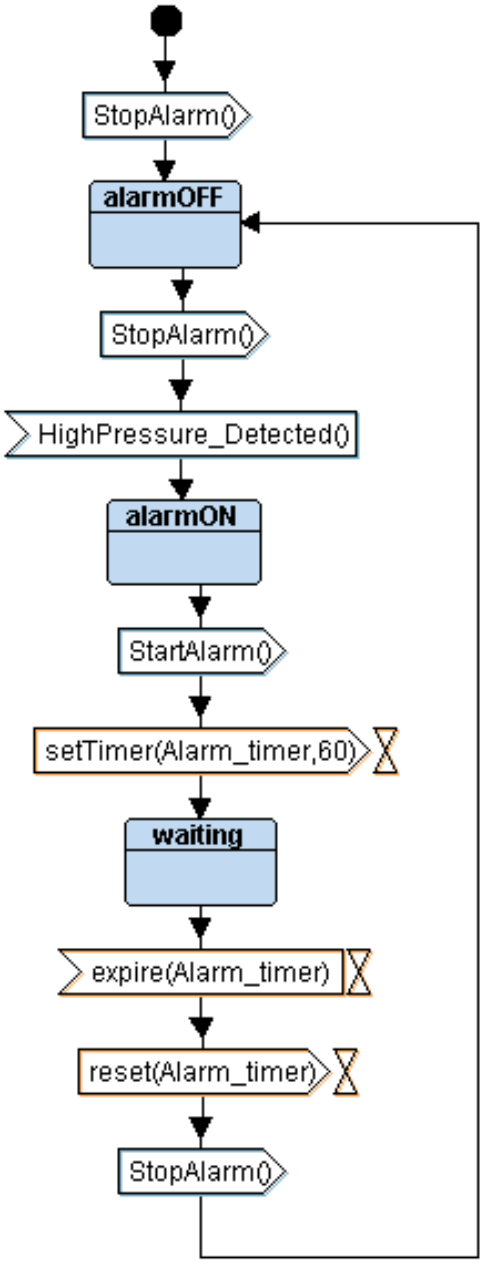
- **State Diagrams**
  - **MainAlgorithm**



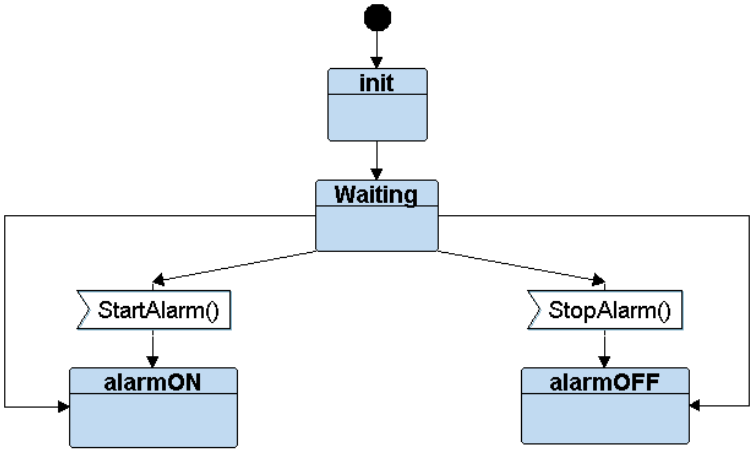
- **PressureSensorDriver**



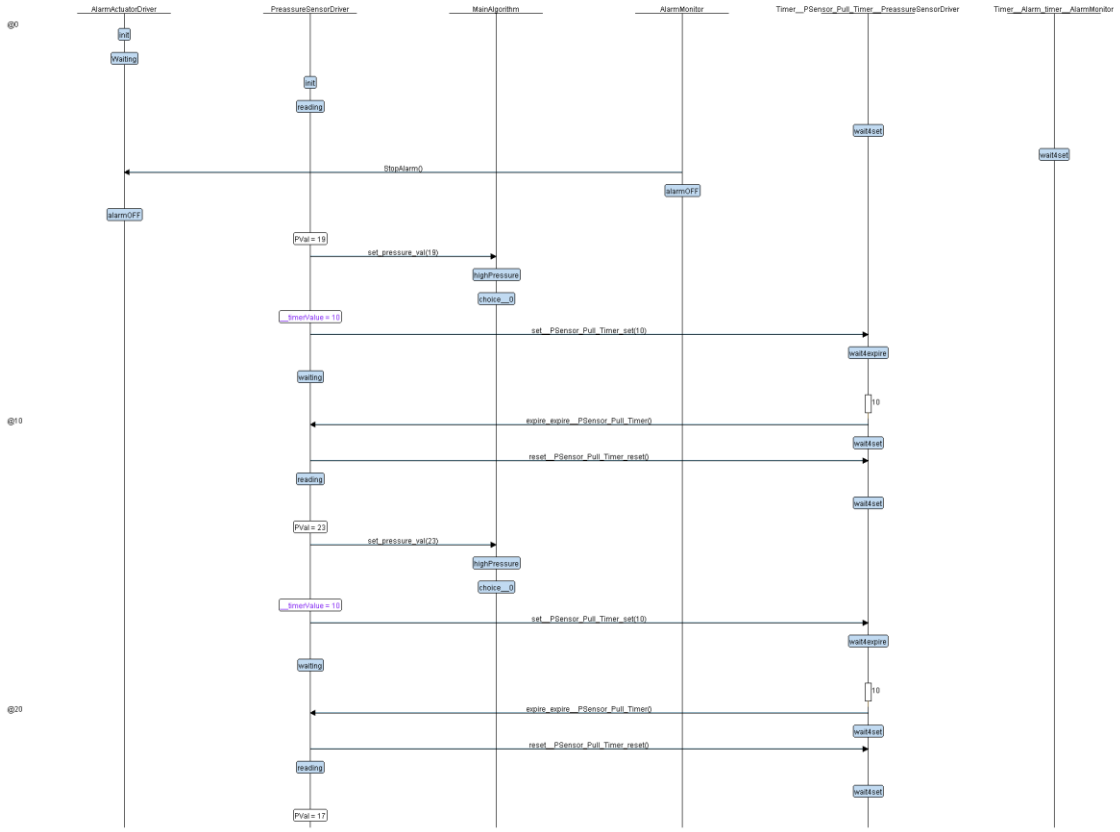
- **AlarmMonitor**



- **AlarmActuatorDriver**

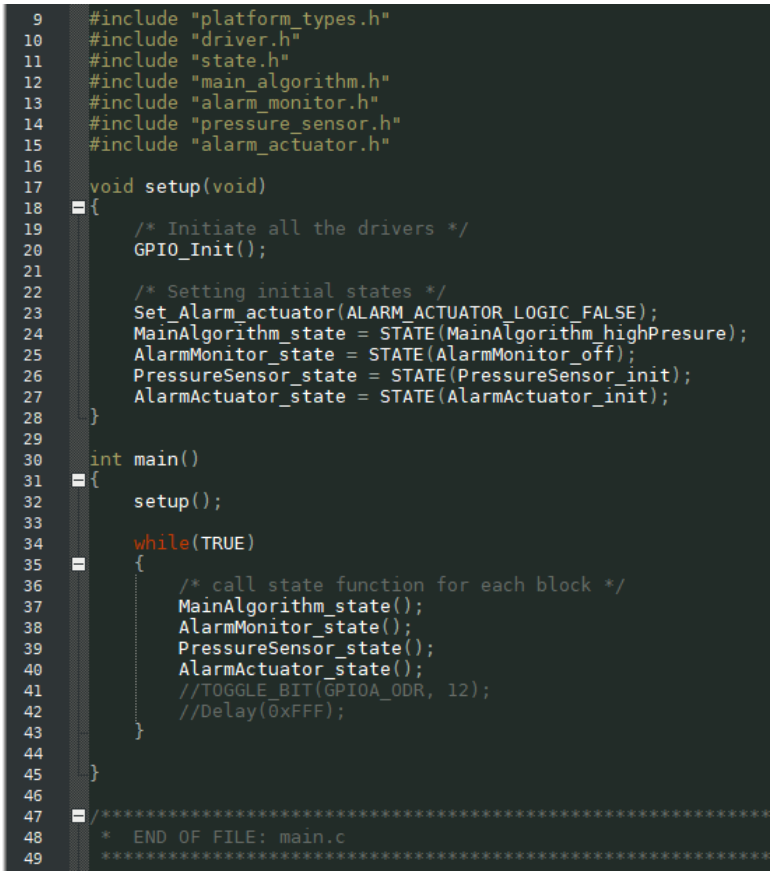


- **System Simulation Result**



Embedded C Code

- *main*



```c
#include "platform_types.h"
#include "driver.h"
#include "state.h"
#include "main_algorithm.h"
#include "alarm_monitor.h"
#include "pressure_sensor.h"
#include "alarm_actuator.h"

void setup(void)
{
    /* Initiate all the drivers */
    GPIO_Init();

    /* Setting initial states */
    Set_Alarm_actuator(ALARM_ACTUATOR_LOGIC_FALSE);
    MainAlgorithm_state = STATE(MainAlgorithm_highPresure);
    AlarmMonitor_state = STATE(AlarmMonitor_off);
    PressureSensor_state = STATE(PressureSensor_init);
    AlarmActuator_state = STATE(AlarmActuator_init);
}

int main()
{
    setup();

    while(TRUE)
    {
        /* call state function for each block */
        MainAlgorithm_state();
        AlarmMonitor_state();
        PressureSensor_state();
        AlarmActuator_state();
        //TOGGLE_BIT(GPIOA_ODR, 12);
        //Delay(0xFFF);
    }

}

/*****************************************************************
 *  END OF FILE: main.c
 *****************************************************************
```

- *driver*

```c
#include "driver.h"

void Delay(vuint32 nCount)
{
    for(; nCount != 0U; nCount--);
}

uint32 getPressureVal(void)
{
    return (GPIOA_IDR & 0xFFU);
}

void Set_Alarm_actuator(boolean state)
{
    if(state == (boolean) TRUE)
    {
        SET_BIT(GPIOA_ODR, 13U);
    }
    else if(state == (boolean) FALSE)
    {
        RESET_BIT(GPIOA_ODR, 13U);
    }
}

void GPIO_Init()
{
    SET_BIT(APB2ENR, 2U);
    /* Input mode with pull-down Port A, Pin 0-7 */
    GPIOA_CRL = 0x88888888U;

    /* General purpose output push-pull with max speed 2 MHz */
    GPIOA_CRH &= 0x00000000U;
    GPIOA_CRH |= 0x22222222U;
}

/**********************************************************************
 *  END OF FILE: driver.c
 **********************************************************************/
```

```c
#ifndef DRIVER_H_
#define DRIVER_H_

#include "platform_types.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 0x40010800U
#define BASE_RCC   0x40021000U

#define APB2ENR   *(volatile uint32_t *)(BASE_RCC + 0x18U)

#define GPIOA_CRL *(volatile uint32_t *)(GPIO_PORTA + 0x00U)
#define GPIOA_CRH *(volatile uint32_t *)(GPIO_PORTA + 0x04U)
#define GPIOA_IDR *(volatile uint32_t *)(GPIO_PORTA + 0x08U)
#define GPIOA_ODR *(volatile uint32_t *)(GPIO_PORTA + 0x0CU)

/**********************************************************************
 * @brief Add a delay load
 *
 * Add a delay load using for loop
 *
 * @param nCount                    The loop counter
 *
 * @returns None
 **********************************************************************/
void Delay(vuint32 nCount);

/**********************************************************************
 * @brief Gets the current pressure value
 *
 * Gets current pressure value by reading the GPIO lower byte
 * of the port
 *
 * @param None
 *
 * @returns uint32                  Current port value
 **********************************************************************/
uint32 getPressureVal(void);

/**********************************************************************
 * @brief Sets the actuator alarm
 *
 * Sets the actuator alarm by setting the corresponding bit as high
 *
 * @param boolean                   The pin state   TRUE
 *                                                  FALSE
 *
 * @returns None
 **********************************************************************/
void Set_Alarm_actuator(boolean state);

/**********************************************************************
 * @brief Initialize the GPIO peripheral
 *
 * Initialize the GPIO peripheral
 *
 * @param None
 *
 * @returns None
 **********************************************************************/
void GPIO_Init(void);

#endif /* DRIVER_H_ */

/**********************************************************************************************************
 *  END OF FILE: driver.h
 **********************************************************************************************************/
```

- *state*

```c
#ifndef STATE_H_
#define STATE_H_

#include "platform_types.h"

/* Main algorithm user defined pressure threshold */
#define MAIN_ALGORITHM_PRESSURE_THRESHOLD               20U

/* Alarm actuator output logic definitions */
#define FLIPPED_LOGIC                                   1U

#if (FLIPPED_LOGIC == 1)
#define ALARM_ACTUATOR_LOGIC_TRUE                       FALSE
#define ALARM_ACTUATOR_LOGIC_FALSE                      TRUE
#else
#define ALARM_ACTUATOR_LOGIC_TRUE                       TRUE
#define ALARM_ACTUATOR_LOGIC_FALSE                      FALSE
#endif

/* Sensor wait time */
#define PRESSURE_SENSOR_WAIT_TIME                       0xFU

/* Alarm monitor wait time, note that wait time is 60 seconds by requirements */
#define ALARM_MONITOR_WAIT_TIME                         0xFFU

/* Macros to define states name and functions prototype */
#define STATE_define(stateFn) void ST_##stateFn(void)
#define STATE(stateFn) ST_##stateFn

/* States connection functions */

extern uint32 PressureSensor_GetPressureValue(void);
extern void AlarmActuator_StartAlarm(void);
extern void AlarmActuator_StopAlarm(void);
extern boolean MainAlgorithm_HighPressureDetected(void);

#endif /* STATE_H_ */


/**********************************************************************************************************
 *  END OF FILE: state.h
 **********************************************************************************************************/
```

- *main_algorithm*

```c
#include "main_algorithm.h"

/* Defining main algorithm status */
MainAlgorithm_statuses MainAlgorithm_status;

/* Pointer to main algorithm state function */
void (*MainAlgorithm_state)(void);

/* Main algorithm parameters */
static uint32 MainAlgorithm_pressureValue;
const static uint32 MainAlgorithm_pressureThreshold = MAIN_ALGORITHM_PRESSURE_THRESHOLD;

STATE_define(MainAlgorithm_highPresure)
{
    /* State action */
    MainAlgorithm_status = MainAlgorithm_highPresure;

    /* Get pressure reading from pressure sensor */
    MainAlgorithm_pressureValue = PressureSensor_GetPressureValue();

    /* Check for event and update the state */
    MainAlgorithm_state = STATE(MainAlgorithm_highPresure);
}

boolean MainAlgorithm_HighPressureDetected(void)
{
    boolean isPressureHigh = (boolean) FALSE;
    /* Check current pressure value with the threshold */
    if(MainAlgorithm_pressureValue > MainAlgorithm_pressureThreshold)
    {
        isPressureHigh = (boolean) TRUE;
    }

    return isPressureHigh;
}

/**********************************************************************************
 *  END OF FILE: main_algorithm.c
 **********************************************************************************/
```

```c

#ifndef MAIN_ALGORITHM_H_
#define MAIN_ALGORITHM_H_

#include "platform_types.h"
#include "state.h"

/* Defining main algorithm statuses enumeration */
typedef enum
{
    MainAlgorithm_highPresure,
} MainAlgorithm_statuses;

/* Main algorithm statuses prototypes */
STATE_define(MainAlgorithm_highPresure);

/* Main algorithm current state */
extern void (*MainAlgorithm_state)(void);

/**************************************************************************
 * @brief Notify that a high pressure is detected
 *
 * Notify that a high pressure is detected by comparing
 * current sensor reading with a pre-defined threshold
 *
 * @param None
 *
 * @returns boolean                High pressure detected decision TRUE
                                                                   False
 **************************************************************************/
boolean MainAlgorithm_HighPressureDetected(void);

#endif /* MAIN_ALGORITHM_H_ */


/**********************************************************************************
 *  END OF FILE: main_algorithm.h
 **********************************************************************************/
```

- Pressure_sensor

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

/* Defining pressure sensor status */
PressureSensor_statuses PressureSensor_status;

/* Pointer to pressure sensor state function */
void (*PressureSensor_state)(void);

static uint32 PressureSensor_PressureValue;

STATE_define(PressureSensor_init)
{
    /* Pressure sensor initialization procedures to put it in initiation state */
    /* State action */
    PressureSensor_status = PressureSensor_init;

    /* Check for event and update the state */
    PressureSensor_state = STATE(PressureSensor_reading);
}

STATE_define(PressureSensor_reading)
{
    /* State action */
    PressureSensor_status = PressureSensor_reading;

    /* Get the current sensor reading */
    PressureSensor_PressureValue = getPressureVal();

    /* Check for event and update the state */
    PressureSensor_state = STATE(PressureSensor_waiting);
}

STATE_define(PressureSensor_waiting)
{
    /* State action */
    PressureSensor_status = PressureSensor_waiting;

    /* Waiting for pressure sensor data */
    Delay(PRESSURE_SENSOR_WAIT_TIME);

    /* Check for event and update the state */
    PressureSensor_state = STATE(PressureSensor_reading);
}

uint32 PressureSensor_GetPressureValue(void)
{
    /* return saved pressure sensor reading */
    return PressureSensor_PressureValue;
}

/**********************************************************************************
 *  END OF FILE: pressure_sensor.c
 **********************************************************************************/
```

```c
#ifndef PRESSURE_SENSOR_H_
#define PRESSURE_SENSOR_H_

#include "platform_types.h"
#include "state.h"

/* Defining pressure sensor statuses enumeration */
typedef enum
{
    PressureSensor_init,
    PressureSensor_reading,
    PressureSensor_waiting,
} PressureSensor_statuses;

/* Pressure sensor statuses prototypes */
STATE_define(PressureSensor_init);
STATE_define(PressureSensor_reading);
STATE_define(PressureSensor_waiting);

/* Pressure sensor current state */
extern void (*PressureSensor_state)(void);

/**************************************************************************
 * @brief Gets the pressure reading
 *
 * Gets the pressure reading from the pressure sensor
 *
 * @param None
 *
 * @returns uint32                 Pressure reading
 **************************************************************************/
uint32 PressureSensor_GetPressureValue(void);

#endif /* PRESSURE_SENSOR_H_ */


/**********************************************************************************
 *  END OF FILE: pressure_sensor.h
 **********************************************************************************/
```

- *alarm_monitor*

```c
#include "alarm_monitor.h"

/* Defining pressure sensor status */
AlarmMonitor_statuses AlarmMonitor_status;

/* Pointer to alarm monitor state function */
void (*AlarmMonitor_state)(void);

STATE_define(AlarmMonitor_off)
{
    /* State action */
    AlarmMonitor_status = AlarmMonitor_off;

    AlarmActuator_StopAlarm();

    /* Check for event and update the state */
    if(MainAlgorithm_HighPressureDetected() == TRUE)
    {
        AlarmMonitor_state = STATE(AlarmMonitor_on);
    }
}

STATE_define(AlarmMonitor_on)
{
    /* State action */
    AlarmMonitor_status = AlarmMonitor_on;

    AlarmActuator_StartAlarm();

    /* Check for event and update the state */
    AlarmMonitor_state = STATE(AlarmMonitor_waiting);
}

STATE_define(AlarmMonitor_waiting)
{
    /* State action */
    AlarmMonitor_status = AlarmMonitor_waiting;

    Delay(ALARM_MONITOR_WAIT_TIME);

    /* Check for event and update the state */
    AlarmMonitor_state = STATE(AlarmMonitor_off);
}

/**************************************************
 *  END OF FILE: alarm_monitor.c
 **************************************************/
```

```c
 *
 * @author Abanoub Salah
 * @date September 13, 2022
 * ************************************************

#ifndef ALARM_MONITOR_H_
#define ALARM_MONITOR_H_

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

/* Defining pressure sensor statuses enumeration */
typedef enum
{
    AlarmMonitor_off,
    AlarmMonitor_on,
    AlarmMonitor_waiting,
} AlarmMonitor_statuses;

/* Alarm monitor statuses prototypes */
STATE_define(AlarmMonitor_off);
STATE_define(AlarmMonitor_on);
STATE_define(AlarmMonitor_waiting);

/* Alarm monitor current state */
extern void (*AlarmMonitor_state)(void);

#endif /* ALARM_MONITOR_H_ */


/**************************************************
 *  END OF FILE: alarm_monitor.h
 **************************************************/
```

- *alarm_actuator*

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

/* Defining alarm actuator status */
AlarmActuator_statuses AlarmActuator_status;

/* Pointer to alarm actuator state function */
void (*AlarmActuator_state)(void);

STATE_define(AlarmActuator_off)
{
    /* State action */
    AlarmActuator_status = AlarmActuator_off;

    Set_Alarm_actuator(ALARM_ACTUATOR_LOGIC_FALSE);

    /* Check for event and update the state */
    AlarmActuator_state = STATE(AlarmActuator_waiting);
}

STATE_define(AlarmActuator_on)
{
    /* State action */
    AlarmActuator_status = AlarmActuator_on;

    Set_Alarm_actuator(ALARM_ACTUATOR_LOGIC_TRUE);

    /* Check for event and update the state */
    AlarmActuator_state = STATE(AlarmActuator_waiting);
}

STATE_define(AlarmActuator_init)
{
    /* Alarm actuator initialization procedures to put it in known state */
    /* State action */
    AlarmActuator_status = AlarmActuator_init;

    /* Check for event and update the state */
    AlarmActuator_state = STATE(AlarmActuator_waiting);
}

STATE_define(AlarmActuator_waiting)
{
    /* State action */
    AlarmActuator_status = AlarmActuator_waiting;
}

void AlarmActuator_StartAlarm(void)
{
    /* Update current State */
    AlarmActuator_state = STATE(AlarmActuator_on);
}

void AlarmActuator_StopAlarm(void)
{
    /* Update current State */
    AlarmActuator_state = STATE(AlarmActuator_off);
}

/*****************************************************************************
 *  END OF FILE: alarm_actuator.c
 *****************************************************************************/

```

```c
#ifndef ALARM_ACTUATOR_H_
#define ALARM_ACTUATOR_H_

#include "platform_types.h"
#include "state.h"

/* Defining alarm actuator statuses enumeration */
typedef enum
{
    AlarmActuator_off,
    AlarmActuator_on,
    AlarmActuator_init,
    AlarmActuator_waiting,
} AlarmActuator_statuses;

/* Alarm actuator statuses prototypes */
STATE_define(AlarmActuator_off);
STATE_define(AlarmActuator_on);
STATE_define(AlarmActuator_init);
STATE_define(AlarmActuator_waiting);

/* Alarm actuator current state */
extern void (*AlarmActuator_state)(void);

/*****************************************************
 * @brief Starts the alarm
 *
 * Starts the alarm by setting a pin high
 *
 * @param None
 *
 * @returns None
 *****************************************************/
void AlarmActuator_StartAlarm(void);

/*****************************************************
 * @brief Stops the alarm
 *
 * Stops the alarm by setting a pin high
 *
 * @param None
 *
 * @returns None
 *****************************************************/
void AlarmActuator_StopAlarm(void);

#endif /* ALARM_ACTUATOR_H_ */


/*****************************************************
 *  END OF FILE: alarm_actuator.h
 *****************************************************/

```

- *startup*

```c
#include "platform_types.h"

extern uint32 __data_start__;
extern uint32 __data_end__;
extern uint32 __bss_start__;
extern uint32 __bss_end__;
extern uint32 __text_end__;
extern uint32 __stack_top__;

extern sint32 main(void);

void Reset_Handler(void);

void Default_Handler()
{
    Reset_Handler();
}

void NMI_Handler(void) __attribute__ ((weak, alias ("Default_Handler")));
void H_fault_Handler(void) __attribute__ ((weak, alias ("Default_Handler")));

void (* vectorTable[])() __attribute__((section(".vectors"))) =
{
    (void (*)())((uint32) &(__stack_top__)),
    (void (*)())((uint32) &(Reset_Handler)),
    (void (*)())((uint32) &(NMI_Handler)),
    (void (*)())((uint32) &(H_fault_Handler)),
};

void Reset_Handler (void)
{
    /* copy data from ROM to RAM */
    uint32 DATA_size = (uint32 *) &(__data_end__) - (uint32 *) &(__data_start__);
    uint8 *P_src = (uint8 *) &(__text_end__);
    uint8 *P_dst = (uint8 *) &(__data_start__);

    for( uint32 i = 0; i < DATA_size; ++i)
    {
        *((uint8 *) P_dst++) = *((uint8*) P_src++);
    }

    /* initiate the .bss section with zeros */
    uint32 bss_size = (uint32 *) &(__bss_end__) - (uint32 *) &(__bss_start__);

    P_dst = (uint8 *) &(__bss_start__);

    for(uint32 i = 0; i < bss_size; ++i)
    {
        *((uint8 *) P_dst++) = (uint8) 0;
    }

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

/**************************************************************************************
 *  END OF FILE: startup.c
 **************************************************************************************/
```

- *linker_script*

```
/* list of memory sections */
MEMORY
{
    flash (RX) : ORIGIN = 0x08000000, LENGTH = 32K
    sram (RWX) : ORIGIN = 0x20000000, LENGTH = 10K
}

/* Entry Point */
ENTRY(Reset_Handler)

/* list of output sections */
SECTIONS
{
    .text : {
        *(.vectors*)
        *(.text*)
        KEEP(*(.init))
        KEEP(*(.fini))
        __text_end__ = . ;
    } > flash

    .rodata : {
    . = ALIGN(4);
    *(.rodata)
    *(.rodata*)
    . = ALIGN(4);
    } > flash

    .data :
    {
        __data_start__ = . ;
        *(vtable)
        *(.data*)

        . = ALIGN(4);
        /* preinit data */
        KEEP(*(.preinit_array))

        . = ALIGN(4);
        /* init data */
        KEEP(*(SORT(.init_array.*)))
        KEEP(*(.init_array))

        . = ALIGN(4);
        /* finit data */
        KEEP(*(SORT(.fini_array.*)))
        KEEP(*(.fini_array))

        . = ALIGN(4) ;
        __data_end__ = . ;
    }> sram AT> flash

    .bss :
    {
        __bss_start__ = . ;
        *(.bss*)
        *(COMMON)
        . = ALIGN(4) ;
        __bss_end__ = . ;
        . = ALIGN(4) ;
        . = . + 0x1000 ;
        __stack_top__ = . ;
    } > sram
}
```

- *Makefile*

```
include sources.mk

BUILD = release
BUILDDIR = build
TARGET = Pressure_Controller
OBJS := $(SOURCES:.c=.o)
DEPS := $(SOURCES:.c=.d)

# Architectures Specific Flags
LINKER_FILE = linker_script.ld
CPU = cortex-m3
SPEC =

# Compiler Flags and Defines
CC = arm-none-eabi-gcc
LD = arm-none-eabi-ld
SIZE = arm-none-eabi-size
OBJDUMP = arm-none-eabi-objdump
OBJCOPY = arm-none-eabi-objcopy
LDFLAGS = -T $(LINKER_FILE) -Map=$(BUILDDIR)/$(TARGET).map
CFLAGS = -Wall -mcpu=$(CPU) $(INCLUDES) -mthumb
CPPFLAGS =

ifeq ($(BUILD), debug)
CFLAGS := -g3 -gdwarf-2 -O0 $(CFLAGS)
endif

# Cppcheck variables
DUMPS = $(SOURCES:.c=.c.dump)
CPPCHECK = cppcheck
MISRA := "$(shell which cppcheck)/../addons/misra.py"

.PHONY: build
build : make_dirs $(OBJS)
    @echo Linking object files in to an executable...
    @$(LD) $(LDFLAGS) $(addprefix $(BUILDDIR)/,$(notdir $(OBJS))) -o $(BUILDDIR)/$(TARGET).elf
ifeq ($(BUILD), debug)
    @echo Generating assembly file from generated executable...
    @$(OBJDUMP) -D -S $(BUILDDIR)/$(TARGET).elf > $(BUILDDIR)/$(TARGET).asm
endif
    @echo Generating hex file from generated executable...
    @$(OBJCOPY) -O ihex $(BUILDDIR)/$(TARGET).elf $(BUILDDIR)/$(TARGET).hex
    @echo Executable and object final files sizes
    @$(SIZE) -Btd $(BUILDDIR)/$(TARGET).elf $(addprefix $(BUILDDIR)/,$(notdir $(OBJS)))

.PHONY: compile-all
compile_all : $(OBJS)
    @echo Finished generating object files

%.i : %.c
    @$(CC) -E $(CFLAGS) -o $(BUILDDIR)/$(notdir $@) $<

%.asm : %.c %.o
    @$(CC) -S $(CFLAGS) $<
    @$(OBJDUMP) -D -S $(BUILDDIR)/$(notdir $(word 2,$^)) > $(BUILDDIR)/$(notdir $@)

%.o : %.c
    @$(CC) -c $< $(CFLAGS) -o $(BUILDDIR)/$(notdir $@)

make_dirs :
    @mkdir -p $(BUILDDIR)

.PHONY: clean
clean :
    @echo Removing executable, hex and map files
    @rm -f $(BUILDDIR)/*.elf $(BUILDDIR)/*.hex $(BUILDDIR)/*.map

.PHONY: clean_all
clean_all :
    @echo Removing all files generated by make
    @rm -f $(BUILDDIR)/*.*

# Generate dump file from c file
%.c.dump: %.c
    @$(CPPCHECK) --dump $(INCLUDES) $<
    @mv $@ $(BUILDDIR)/$(notdir $@)

.PHONY: misra_compliant
misra_compliant : $(DUMPS)
    @echo Displaying MISRA rules violation discovered by Cppcheck
    @-python $(MISRA) $(realpath $(addprefix $(BUILDDIR)/,$(notdir $(DUMPS))))

```

```
#-------------------------------------------------
#
# file includes project sources and headers folder
#
#-------------------------------------------------

SOURCES =   src/main.c              \
            src/driver.c            \
            src/pressure_sensor.c   \
            src/alarm_monitor.c     \
            src/main_algorithm.c    \
            src/alarm_actuator.c    \
            src/startup.c

# Add your include paths to this variable
INCLUDES = -I./inc
```

Simulation result

- *Compilation result*

```
$ make
Linking object files in to an executable...
Generating hex file from generated executable...
Executable and object final files sizes
   text    data     bss     dec     hex filename
   1040       0    4136    5176    1438 build/Pressure_Controller.elf
    128       0       0     128      80 build/main.o
    192       0       0     192      c0 build/driver.o
    144       0      12     156      9c build/pressure_sensor.o
    124       0       8     132      84 build/alarm_monitor.o
     92       0      12     104      68 build/main_algorithm.o
    196       0       8     204      cc build/alarm_actuator.o
    148      16       0     164      a4 build/startup.o
   2064      16    4176    6256    1870 (TOTALS)
```

- *map_file (*partial)*

```
Memory Configuration

Name             Origin             Length             Attributes
flash            0x0000000008000000 0x0000000000008000 xr
sram             0x0000000020000000 0x0000000000002800 xrw
*default*        0x0000000000000000 0xffffffffffffffff

Linker script and memory map


.text           0x0000000008000000      0x40c
 *(.vectors*)
 .vectors       0x0000000008000000       0x10 build/startup.o
                0x0000000008000000                vectorTable
 *(.text*)
 .text          0x0000000008000010       0x80 build/main.o
                0x0000000008000010                setup
                0x000000000800005c                main
 .text          0x0000000008000090       0xc0 build/driver.o
                0x0000000008000090                Delay
                0x00000000080000b2                getPressureVal
                0x00000000080000c8                Set_Alarm_actuator
                0x000000000800010c                GPIO_Init
 .text          0x0000000008000150       0x90 build/pressure_sensor.o
                0x0000000008000150                ST_PressureSensor_init
                0x0000000008000174                ST_PressureSensor_reading
                0x00000000080001a4                ST_PressureSensor_waiting
                0x00000000080001cc                PressureSensor_GetPressureValue
 .text          0x00000000080001e0       0x7c build/alarm_monitor.o
                0x00000000080001e0                ST_AlarmMonitor_off
                0x0000000008000210                ST_AlarmMonitor_on
                0x0000000008000234                ST_AlarmMonitor_waiting
 .text          0x000000000800025c       0x58 build/main_algorithm.o
                0x000000000800025c                ST_MainAlgorithm_highPresure
                0x000000000800028c                MainAlgorithm_HighPressureDetected
 .text          0x00000000080002b4       0xc4 build/alarm_actuator.o
                0x00000000080002b4                ST_AlarmActuator_off
                0x00000000080002dc                ST_AlarmActuator_on
                0x0000000008000304                ST_AlarmActuator_init
                0x0000000008000328                ST_AlarmActuator_waiting
                0x0000000008000340                AlarmActuator_StartAlarm
                0x000000000800035c                AlarmActuator_StopAlarm
 .text          0x0000000008000378       0x94 build/startup.o
                0x0000000008000378                H_fault_Handler
                0x0000000008000378                Default_Handler
                0x0000000008000378                NMI_Handler
                0x0000000008000384                Reset_Handler
```

- *Proteus simulation*