

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
- Methodology Used
- Requirements Diagram
- System Analysis
- System Design
- Embedded C Code
- Simulation result

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

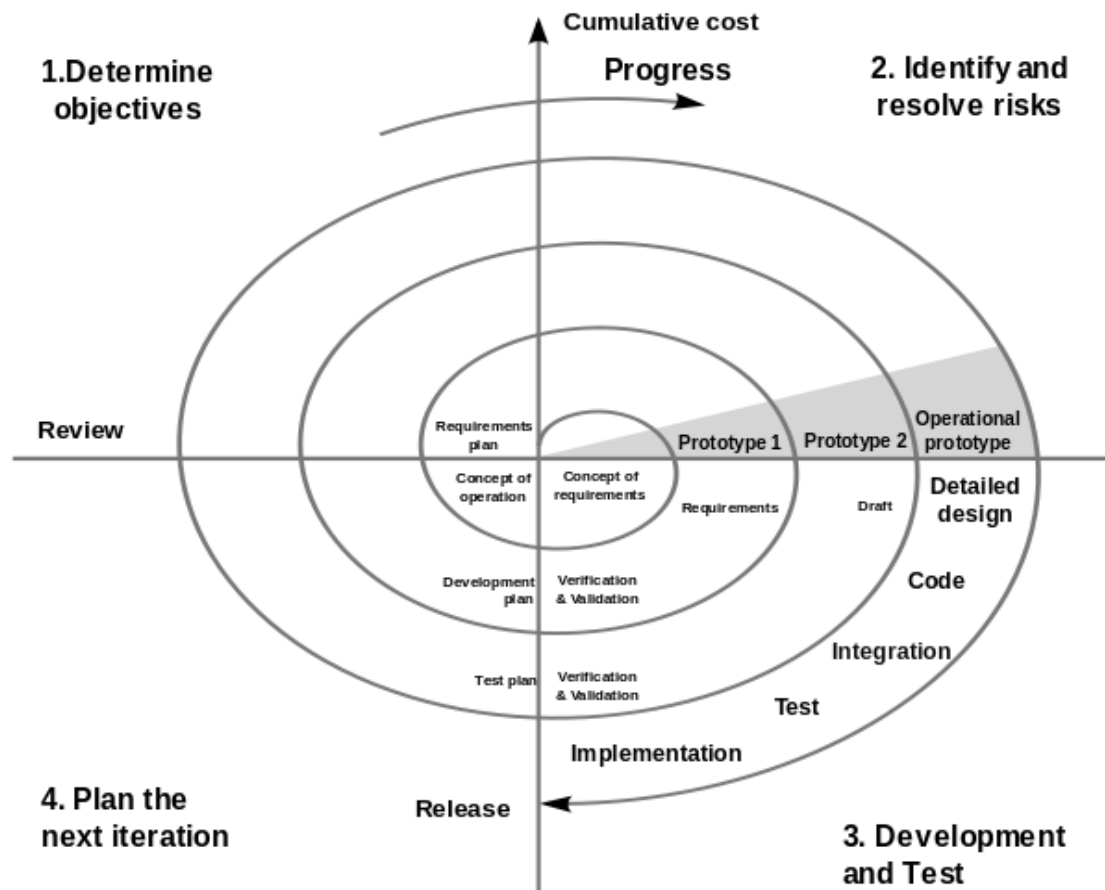
System Assumptions:

- 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

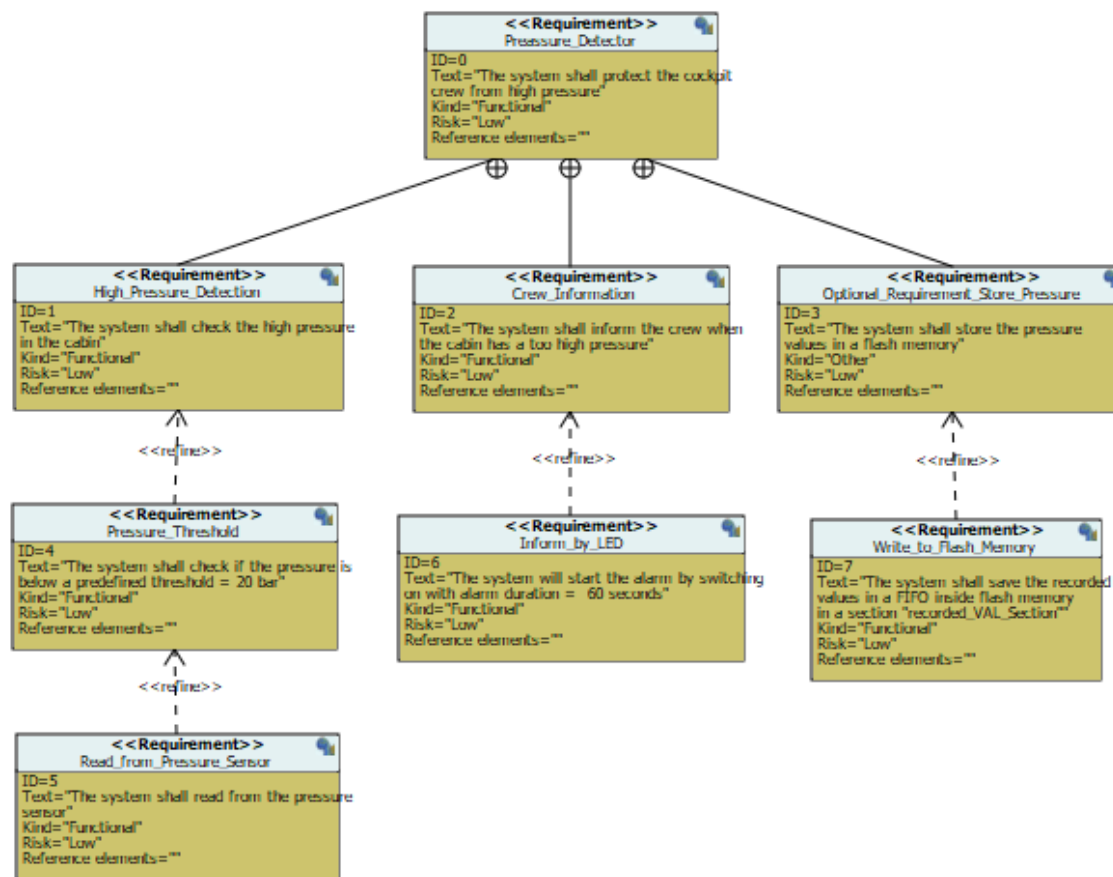
Methodology Used

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

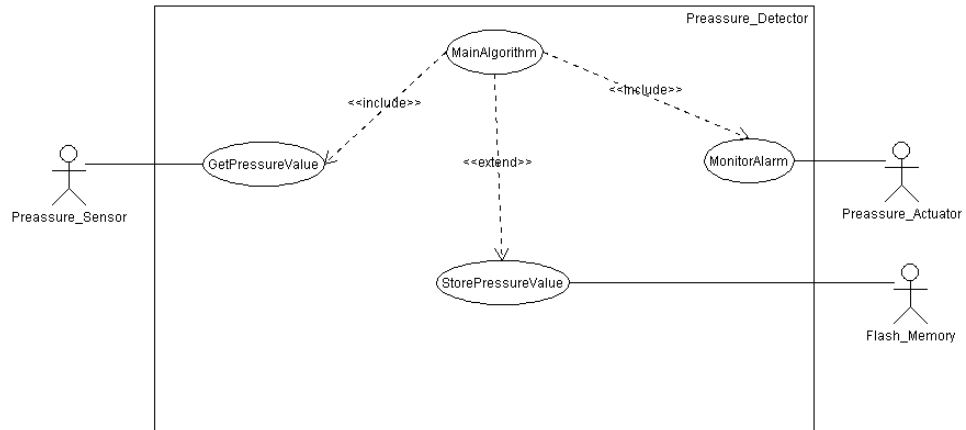


Space Exploration

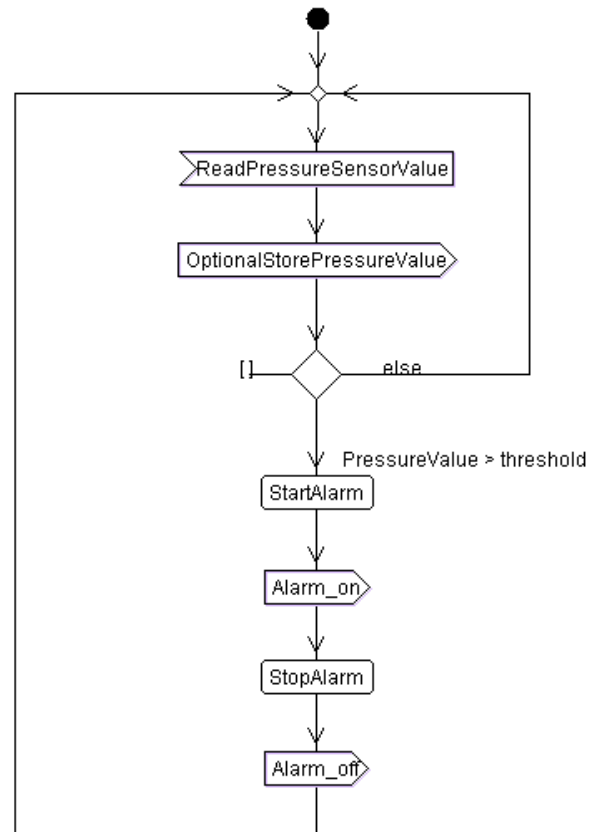
We will use STM32 ARM Cortex-M based μ Controller. It is a high end 32bit μ Controller capable of performing well for this project requirement.

System Analysis

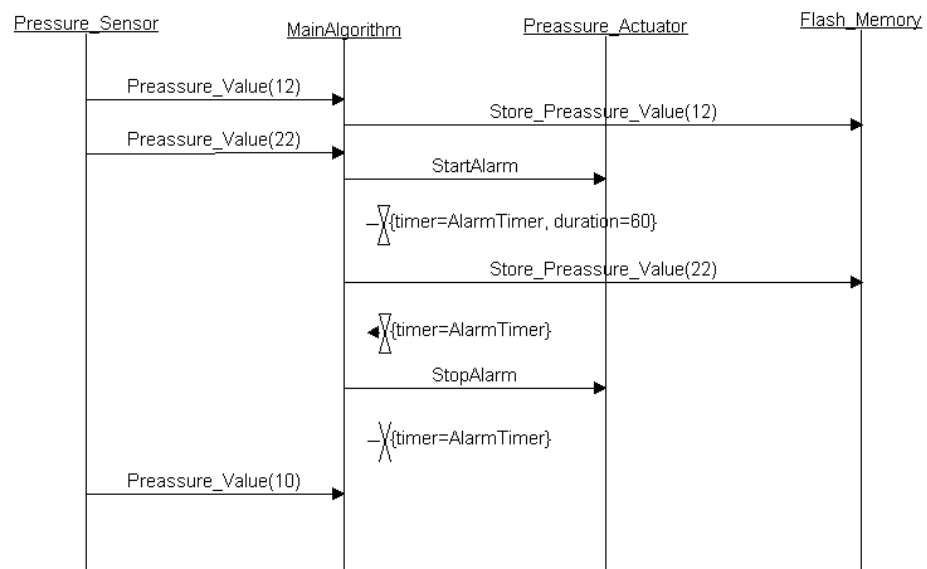
- Use case Diagram



- **Activity Diagram**

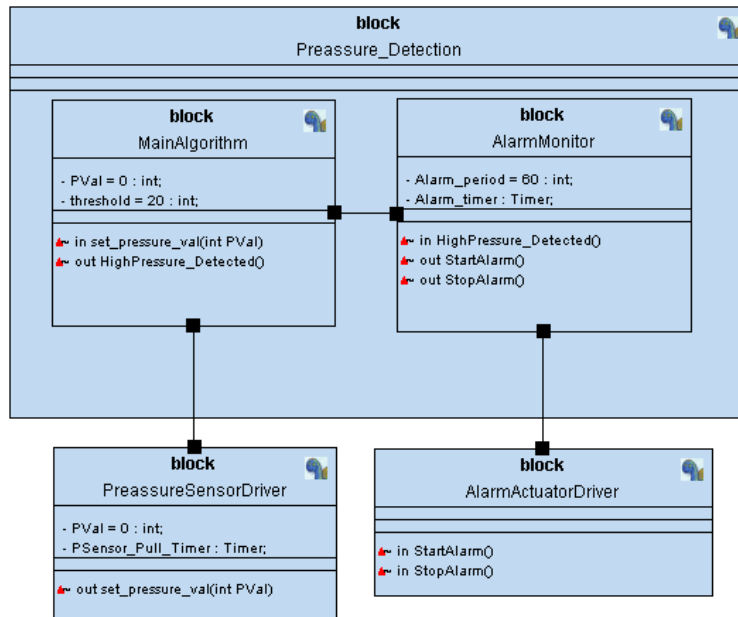


- **Sequence Diagram**



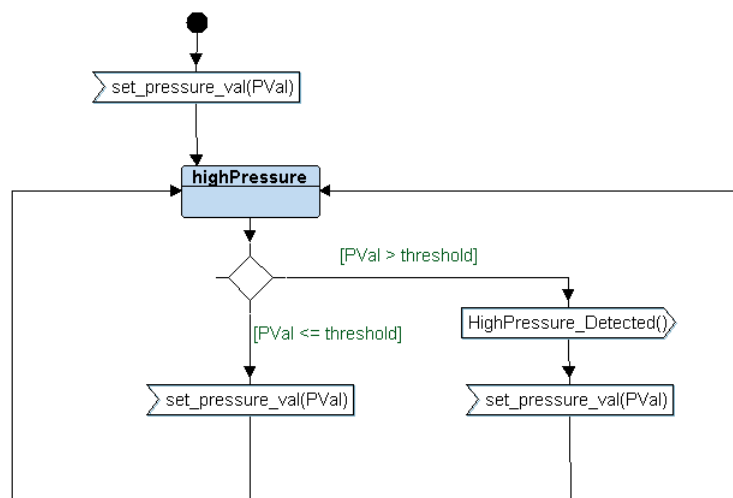
System Design

- Block Diagram

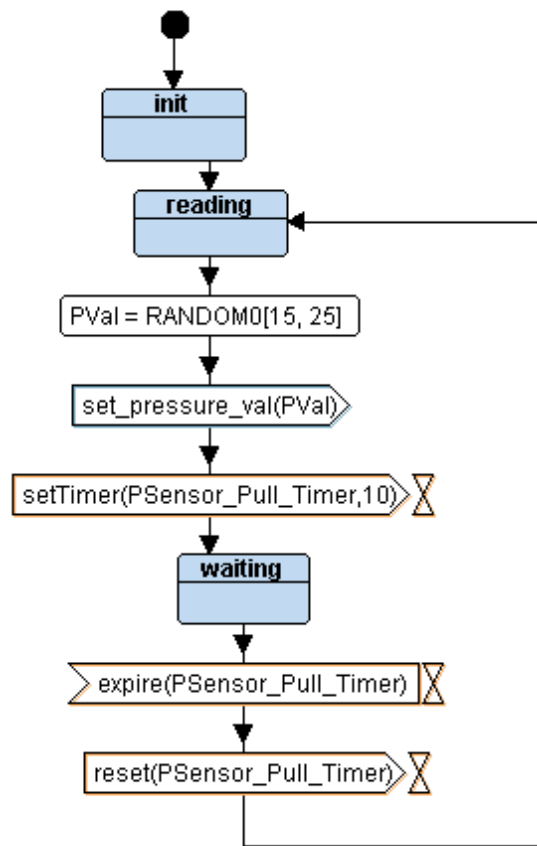


- State Diagrams

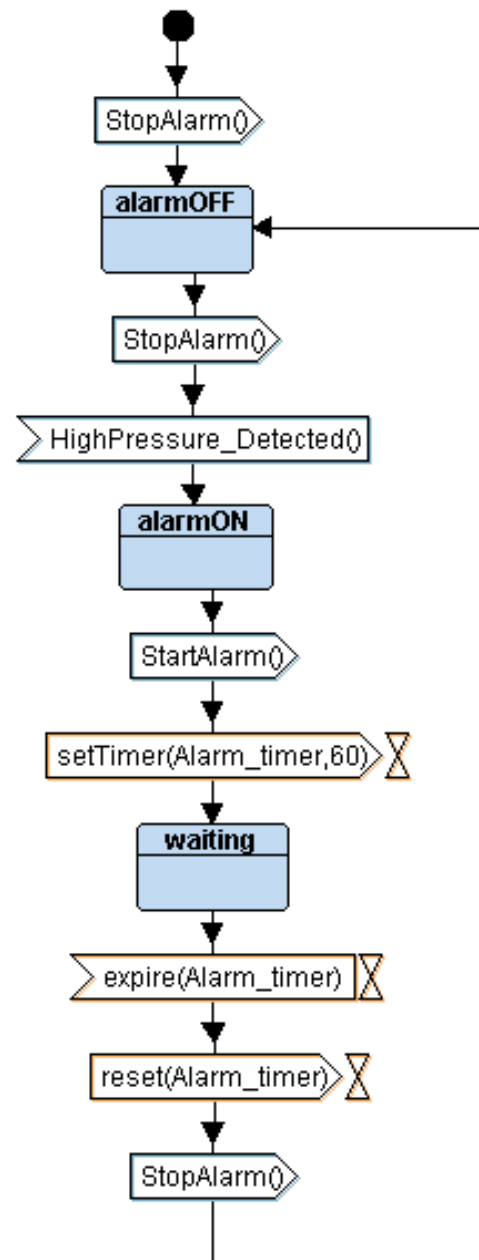
- MainAlgorithm



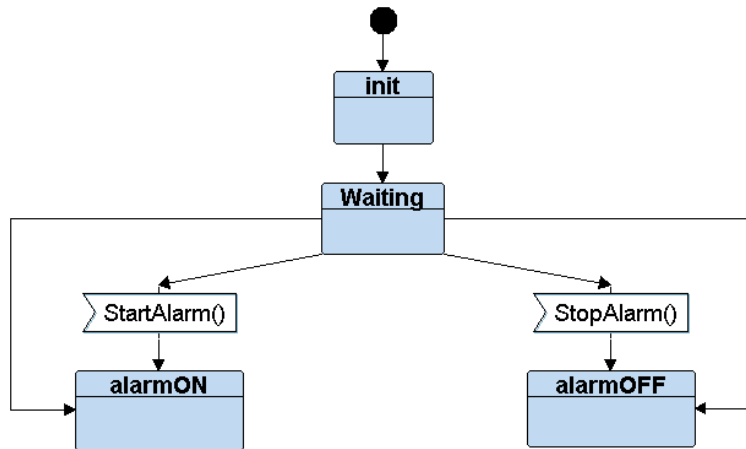
○ PressureSensorDriver



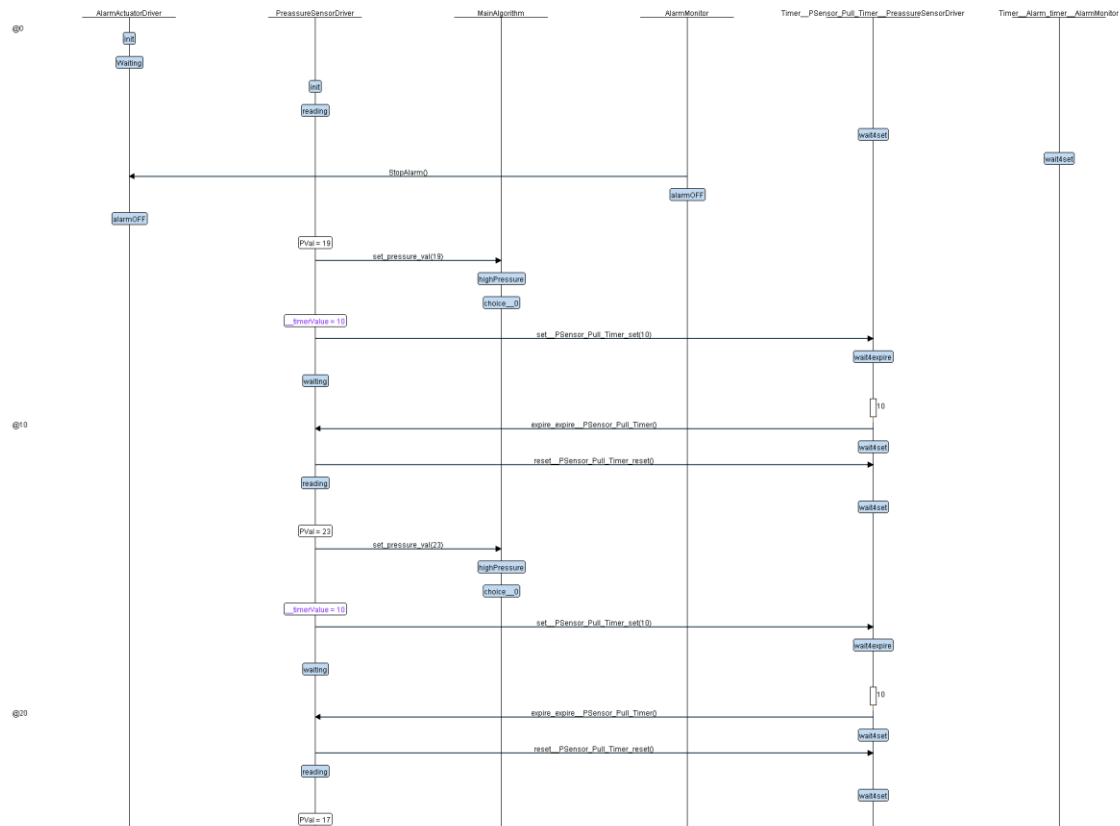
○ AlarmMonitor



○ AlarmActuatorDriver



● System Simulation Result



Embedded C Code

- *main*

```
9  #include "platform_types.h"
10 #include "driver.h"
11 #include "state.h"
12 #include "main_algorithm.h"
13 #include "alarm_monitor.h"
14 #include "pressure_sensor.h"
15 #include "alarm_actuator.h"
16
17 void setup(void)
18 {
19     /* Initiate all the drivers */
20     GPIO_Init();
21
22     /* Setting initial states */
23     Set_Alarm_actuator(ALARM_ACTUATOR_LOGIC_FALSE);
24     MainAlgorithm_state = STATE(MainAlgorithm_highPresure);
25     AlarmMonitor_state = STATE(AlarmMonitor_off);
26     PressureSensor_state = STATE(PressureSensor_init);
27     AlarmActuator_state = STATE(AlarmActuator_init);
28 }
29
30 int main()
31 {
32     setup();
33
34     while(TRUE)
35     {
36         /* call state function for each block */
37         MainAlgorithm_state();
38         AlarmMonitor_state();
39         PressureSensor_state();
40         AlarmActuator_state();
41         //TOGGLE_BIT(GPIOA_ODR, 12);
42         //Delay(0xFFFF);
43     }
44 }
45
46
47 /* *****
48  * END OF FILE: main.c
49  * *****
```

• driver

```

8
9 #include "driver.h"
10
11 void Delay(vuint32 nCount)
12 {
13     for(; nCount != 0U; nCount--);
14 }
15
16 uint32 getPressureVal(void)
17 {
18     return (GPIOA_IDR & 0xFFU);
19 }
20
21 void Set_Alarm_actuator(boolean state)
22 {
23     if(state == (boolean) TRUE)
24     {
25         SET_BIT(GPIOA_ODR, 13U);
26     }
27     else if(state == (boolean) FALSE)
28     {
29         RESET_BIT(GPIOA_ODR, 13U);
30     }
31 }
32
33 void GPIO_Init()
34 {
35     SET_BIT(APB2ENR, 2U);
36     /* Input mode with pull-down Port A, Pin 0-7 */
37     GPIOA_CRL = 0x88888888U;
38
39     /* General purpose output push-pull with max speed 2 MHz */
40     GPIOA_CRH &= 0x00000000U;
41     GPIOA_CRH |= 0x22222222U;
42 }
43
44 /* *****
45  * END OF FILE: driver.c
46  * *****
47

```

```

9 #ifndef DRIVER_H_
10 #define DRIVER_H_
11
12 #include "platform_types.h"
13
14 #define SET_BIT(ADDRESS,BIT) ADDRESS |= (1<<BIT)
15 #define RESET_BIT(ADDRESS,BIT) ADDRESS &= ~(1<<BIT)
16 #define TOGGLE_BIT(ADDRESS,BIT) ADDRESS ^= (1<<BIT)
17 #define READ_BIT(ADDRESS,BIT) ((ADDRESS) & (1<<(BIT)))
18
19 #define GPIO_PORTA 0x40010800U
20 #define BASE_RCC 0x40021000U
21
22 #define APB2ENR *(volatile uint32_t *) (BASE_RCC + 0x18U)
23
24 #define GPIOA_CRL *(volatile uint32_t *) (GPIO_PORTA + 0x00U)
25 #define GPIOA_CRH *(volatile uint32_t *) (GPIO_PORTA + 0x04U)
26 #define GPIOA_IDR *(volatile uint32_t *) (GPIO_PORTA + 0x08U)
27 #define GPIOA_ODR *(volatile uint32_t *) (GPIO_PORTA + 0x0CU)
28
29
30 * @brief Add a delay load
31 *
32 * Add a delay load using for loop
33 *
34 * @param nCount The loop counter
35 *
36 * @returns None
37
38 void Delay(vuint32 nCount);
39
40
41 * @brief Gets the current pressure value
42 *
43 * Gets current pressure value by reading the GPIO lower byte
44 * of the port
45 *
46 * @param None
47 *
48 * @returns uint32 Current port value
49
50 uint32 getPressureVal(void);
51
52
53
54 * @brief Sets the actuator alarm
55 *
56 * Sets the actuator alarm by setting the corresponding bit as high
57 *
58 * @param boolean The pin state TRUE FALSE
59 *
60 *
61 * @returns None
62
63 void Set_Alarm_actuator(boolean state);
64
65
66 * @brief Initialize the GPIO peripheral
67 *
68 * Initialize the GPIO peripheral
69 *
70 * @param None
71 *
72 * @returns None
73
74 void GPIO_Init(void);
75
76 #endif /* DRIVER_H_ */
77
78
79 * END OF FILE: driver.h
80
81

```

• state

```

10 #ifndef STATE_H_
11 #define STATE_H_
12
13 #include "platform_types.h"
14
15 /* Main algorithm user defined pressure threshold */
16 #define MAIN_ALGORITHM_PRESSURE_THRESHOLD 20U
17
18 /* Alarm actuator output logic definitions */
19 #define FLIPPED_LOGIC 1U
20
21 #if (FLIPPED_LOGIC == 1)
22 #define ALARM_ACTUATOR_LOGIC_TRUE FALSE
23 #define ALARM_ACTUATOR_LOGIC_FALSE TRUE
24 #else
25 #define ALARM_ACTUATOR_LOGIC_TRUE TRUE
26 #define ALARM_ACTUATOR_LOGIC_FALSE FALSE
27 #endif
28
29 /* Sensor wait time */
30 #define PRESSURE_SENSOR_WAIT_TIME 0xFFU
31
32 /* Alarm monitor wait time, note that wait time is 60 seconds by requirements */
33 #define ALARM_MONITOR_WAIT_TIME 0xFFU
34
35 /* Macros to define states name and functions prototype */
36 #define STATE_define(stateFn) void ST_##stateFn(void)
37 #define STATE(stateFn) ST_##stateFn
38
39 /* States connection functions */
40
41 extern uint32 PressureSensor_GetPressureValue(void);
42 extern void AlarmActuator_StartAlarm(void);
43 extern void AlarmActuator_StopAlarm(void);
44 extern boolean MainAlgorithm_HighPressureDetected(void);
45
46 #endif /* STATE_H_ */
47
48
49
50 * END OF FILE: state.h
51
52

```

• main_algorithm

```

9  #include "main_algorithm.h"
10
11  /* Defining main algorithm status */
12  MainAlgorithm_statuses MainAlgorithm_status;
13
14  /* Pointer to main algorithm state function */
15  void (*MainAlgorithm_state)(void);
16
17  /* Main algorithm parameters */
18  static uint32 MainAlgorithm_pressureValue;
19  const static uint32 MainAlgorithm_pressureThreshold = MAIN_ALGORITHM_PRESSURE_THRESHOLD;
20
21  STATE_define(MainAlgorithm_highPressure)
22  {
23      /* State action */
24      MainAlgorithm_status = MainAlgorithm_highPressure;
25
26      /* Get pressure reading from pressure sensor */
27      MainAlgorithm_pressureValue = PressureSensor_GetPressureValue();
28
29      /* Check for event and update the state */
30      MainAlgorithm_state = STATE(MainAlgorithm_highPressure);
31  }
32
33  boolean MainAlgorithm_HighPressureDetected(void)
34  {
35      boolean isPressureHigh = (boolean) FALSE;
36      /* Check current pressure value with the threshold */
37      if(MainAlgorithm_pressureValue > MainAlgorithm_pressureThreshold)
38      {
39          isPressureHigh = (boolean) TRUE;
40      }
41
42      return isPressureHigh;
43  }
44
45  /* END OF FILE: main_algorithm.c
46  *****
47  */
48

```

```

8
9  #ifndef MAIN_ALGORITHM_H_
10 #define MAIN_ALGORITHM_H_
11
12  #include "platform_types.h"
13  #include "state.h"
14
15  /* Defining main algorithm statuses enumeration */
16  typedef enum
17  {
18      MainAlgorithm_highPressure,
19  } MainAlgorithm_statuses;
20
21  /* Main algorithm statuses prototypes */
22  STATE_define(MainAlgorithm_highPressure);
23
24  /* Main algorithm current state */
25  extern void (*MainAlgorithm_state)(void);
26
27  /* *****
28   * @brief Notify that a high pressure is detected
29   *
30   * Notify that a high pressure is detected by comparing
31   * current sensor reading with a pre-defined threshold
32   *
33   * @param None
34   *
35   * @returns boolean High pressure detected decision TRUE
36   *                   False
37   * *****
38   boolean MainAlgorithm_HighPressureDetected(void);
39
40  #endif /* MAIN_ALGORITHM_H_ */
41
42
43  /* END OF FILE: main_algorithm.h
44  *****
45  */
46

```

• Pressure_sensor

```

9  #include "pressure_sensor.h"
10 #include "driver.h"
11
12  /* Defining pressure sensor status */
13  PressureSensor_statuses PressureSensor_status;
14
15  /* Pointer to pressure sensor state function */
16  void (*PressureSensor_state)(void);
17
18  static uint32 PressureSensor_PressureValue;
19
20  STATE_define(PressureSensor_init)
21  {
22      /* Pressure sensor initialization procedures to put it in initiation state */
23      /* State action */
24      PressureSensor_status = PressureSensor_init;
25
26      /* Check for event and update the state */
27      PressureSensor_state = STATE(PressureSensor_reading);
28  }
29
30  STATE_define(PressureSensor_reading)
31  {
32      /* State action */
33      PressureSensor_status = PressureSensor_reading;
34
35      /* Get the current sensor reading */
36      PressureSensor_PressureValue = getPressureVal();
37
38      /* Check for event and update the state */
39      PressureSensor_state = STATE(PressureSensor_waiting);
40  }
41
42  STATE_define(PressureSensor_waiting)
43  {
44      /* State action */
45      PressureSensor_status = PressureSensor_waiting;
46
47      /* Waiting for pressure sensor data */
48      Delay(PRESSURE_SENSOR_WAIT_TIME);
49
50      /* Check for event and update the state */
51      PressureSensor_state = STATE(PressureSensor_reading);
52  }
53
54  uint32 PressureSensor_GetPressureValue(void)
55  {
56      /* return saved pressure sensor reading */
57      return PressureSensor_PressureValue;
58  }
59
60  /* *****
61   * END OF FILE: pressure_sensor.c
62   *****
63  */
64

```

```

9  #ifndef PRESSURE_SENSOR_H_
10 #define PRESSURE_SENSOR_H_
11
12  #include "platform_types.h"
13  #include "state.h"
14
15  /* Defining pressure sensor statuses enumeration */
16  typedef enum
17  {
18      PressureSensor_init,
19      PressureSensor_reading,
20      PressureSensor_waiting,
21  } PressureSensor_statuses;
22
23  /* Pressure sensor statuses prototypes */
24  STATE_define(PressureSensor_init);
25  STATE_define(PressureSensor_reading);
26  STATE_define(PressureSensor_waiting);
27
28  /* Pressure sensor current state */
29  extern void (*PressureSensor_state)(void);
30
31  /* *****
32   * @brief Gets the pressure reading
33   *
34   * Gets the pressure reading from the pressure sensor
35   *
36   * @param None
37   *
38   * @returns uint32 Pressure reading
39   * *****
40   uint32 PressureSensor_GetPressureValue(void);
41
42  #endif /* PRESSURE_SENSOR_H_ */
43
44
45  /* *****
46   * END OF FILE: pressure_sensor.h
47   *****
48  */
49

```

- *alarm_monitor*

```

9  #include "alarm_monitor.h"
10
11  /* Defining pressure sensor status */
12  AlarmMonitor_statuses AlarmMonitor_status;
13
14  /* Pointer to alarm monitor state function */
15  void (*AlarmMonitor_state)(void);
16
17  STATE_define(AlarmMonitor_off)
18  = {
19      /* State action */
20      AlarmMonitor_status = AlarmMonitor_off;
21
22      AlarmActuator_StopAlarm();
23
24      /* Check for event and update the state */
25      if(MainAlgorithm_HighPressureDetected() == TRUE)
26      {
27          AlarmMonitor_state = STATE(AlarmMonitor_on);
28      }
29  }
30
31  STATE_define(AlarmMonitor_on)
32  = {
33      /* State action */
34      AlarmMonitor_status = AlarmMonitor_on;
35
36      AlarmActuator_StartAlarm();
37
38      /* Check for event and update the state */
39      AlarmMonitor_state = STATE(AlarmMonitor_waiting);
40  }
41
42  STATE_define(AlarmMonitor_waiting)
43  = {
44      /* State action */
45      AlarmMonitor_status = AlarmMonitor_waiting;
46
47      Delay(ALARM_MONITOR_WAIT_TIME);
48
49      /* Check for event and update the state */
50      AlarmMonitor_state = STATE(AlarmMonitor_off);
51  }
52
53  /* *****
54  * END OF FILE: alarm_monitor.c
55  *****
56

```

```

4  *
5  * @author Abanoub Salah
6  * @date September 13, 2022
7  * *****
8
9  #ifndef ALARM_MONITOR_H_
10 #define ALARM_MONITOR_H_
11
12 #include "platform_types.h"
13 #include "state.h"
14 #include "driver.h"
15
16 /* Defining pressure sensor statuses enumeration */
17 typedef enum
18 = {
19     AlarmMonitor_off,
20     AlarmMonitor_on,
21     AlarmMonitor_waiting,
22 } AlarmMonitor_statuses;
23
24 /* Alarm monitor statuses prototypes */
25 STATE_define(AlarmMonitor_off);
26 STATE_define(AlarmMonitor_on);
27 STATE_define(AlarmMonitor_waiting);
28
29 /* Alarm monitor current state */
30 extern void (*AlarmMonitor_state)(void);
31
32 #endif /* ALARM_MONITOR_H_ */
33
34
35 /* *****
36 * END OF FILE: alarm_monitor.h
37 *****
38

```

• alarm_actuator

```

9  #include "alarm_actuator.h"
10 #include "driver.h"
11
12 /* Defining alarm actuator status */
13 AlarmActuator_statuses AlarmActuator_status;
14
15 /* Pointer to alarm actuator state function */
16 void (*AlarmActuator_state)(void);
17
18 STATE_define(AlarmActuator_off)
19 {
20     /* State action */
21     AlarmActuator_status = AlarmActuator_off;
22
23     Set_Alarm_actuator(ALARM_ACTUATOR_LOGIC_FALSE);
24
25     /* Check for event and update the state */
26     AlarmActuator_state = STATE(AlarmActuator_waiting);
27 }
28
29 STATE_define(AlarmActuator_on)
30 {
31     /* State action */
32     AlarmActuator_status = AlarmActuator_on;
33
34     Set_Alarm_actuator(ALARM_ACTUATOR_LOGIC_TRUE);
35
36     /* Check for event and update the state */
37     AlarmActuator_state = STATE(AlarmActuator_waiting);
38 }
39
40 STATE_define(AlarmActuator_init)
41 {
42     /* Alarm actuator initialization procedures to put it in known state */
43     /* State action */
44     AlarmActuator_status = AlarmActuator_init;
45
46     /* Check for event and update the state */
47     AlarmActuator_state = STATE(AlarmActuator_waiting);
48 }
49
50 STATE_define(AlarmActuator_waiting)
51 {
52     /* State action */
53     AlarmActuator_status = AlarmActuator_waiting;
54 }
55
56 void AlarmActuator_StartAlarm(void)
57 {
58     /* Update current State */
59     AlarmActuator_state = STATE(AlarmActuator_on);
60 }
61
62 void AlarmActuator_StopAlarm(void)
63 {
64     /* Update current State */
65     AlarmActuator_state = STATE(AlarmActuator_off);
66 }
67
68 /* *****
69  * END OF FILE: alarm_actuator.c
70  * *****
71

```

```

8
9  #ifndef ALARM_ACTUATOR_H_
10 #define ALARM_ACTUATOR_H_
11
12 #include "platform_types.h"
13 #include "state.h"
14
15 /* Defining alarm actuator statuses enumeration */
16 typedef enum
17 {
18     AlarmActuator_off,
19     AlarmActuator_on,
20     AlarmActuator_init,
21     AlarmActuator_waiting,
22 } AlarmActuator_statuses;
23
24 /* Alarm actuator statuses prototypes */
25 STATE_define(AlarmActuator_off);
26 STATE_define(AlarmActuator_on);
27 STATE_define(AlarmActuator_init);
28 STATE_define(AlarmActuator_waiting);
29
30 /* Alarm actuator current state */
31 extern void (*AlarmActuator_state)(void);
32
33 /* *****
34  * @brief Starts the alarm
35  *
36  * Starts the alarm by setting a pin high
37  *
38  * @param None
39  *
40  * @returns None
41  * *****
42  void AlarmActuator_StartAlarm(void);
43
44 /* *****
45  * @brief Stops the alarm
46  *
47  * Stops the alarm by setting a pin high
48  *
49  * @param None
50  *
51  * @returns None
52  * *****
53  void AlarmActuator_StopAlarm(void);
54
55 #endif /* ALARM_ACTUATOR_H_ */
56
57
58 /* *****
59  * END OF FILE: alarm_actuator.h
60  * *****
61

```


- *startup*

```
9  #include "platform_types.h"
10
11  extern uint32 __data_start__;
12  extern uint32 __data_end__;
13  extern uint32 __bss_start__;
14  extern uint32 __bss_end__;
15  extern uint32 __text_end__;
16  extern uint32 __stack_top__;
17
18  extern sint32 main(void);
19
20  void Reset_Handler(void);
21
22  void Default_Handler()
23  {
24      Reset_Handler();
25  }
26
27  void NMI_Handler(void) __attribute__((weak, alias ("Default_Handler")));
28  void H_fault_Handler(void) __attribute__((weak, alias ("Default_Handler")));
29
30  void (* vectorTable[])(void) __attribute__((section(".vectors"))) =
31  {
32      (void (*)(void))(&__stack_top__),
33      (void (*)(void))(&Reset_Handler),
34      (void (*)(void))(&NMI_Handler),
35      (void (*)(void))(&H_fault_Handler),
36  };
37
38  void Reset_Handler (void)
39  {
40      /* copy data from ROM to RAM */
41      uint32 DATA_size = (uint32 *) &__data_end__ - (uint32 *) &__data_start__;
42      uint8 *P_src = (uint8 *) &__text_end__;
43      uint8 *P_dst = (uint8 *) &__data_start__;
44
45      for( uint32 i = 0; i < DATA_size; ++i)
46      {
47          *((uint8 *) P_dst++) = *((uint8 *) P_src++);
48      }
49
50      /* initiate the .bss section with zeros */
51      uint32 bss_size = (uint32 *) &__bss_end__ - (uint32 *) &__bss_start__;
52
53      P_dst = (uint8 *) &__bss_start__;
54
55      for(uint32 i = 0; i < bss_size; ++i)
56      {
57          *((uint8 *) P_dst++) = (uint8) 0;
58      }
59
60      /* jump to main */
61      main();
62  }
63
64  /*
65   * END OF FILE: startup.c
66   */
67
```

- *linker_script*

```

1  /* list of memory sections */
2  MEMORY
3  {
4      flash (RX) : ORIGIN = 0x08000000, LENGTH = 32K
5      sram (RWX) : ORIGIN = 0x20000000, LENGTH = 10K
6  }
7
8  /* Entry Point */
9  ENTRY(Reset_Handler)
10
11 /* list of output sections */
12 SECTIONS
13 {
14     .text : {
15         *(.vectors*)
16         *(.text*)
17         KEEP(*(.init))
18         KEEP(*(.fini))
19         __text_end__ = . ;
20     } > flash
21
22     .rodata : {
23         . = ALIGN(4);
24         *(.rodata)
25         *(.rodata*)
26         . = ALIGN(4);
27     } > flash
28
29     .data :
30     {
31         __data_start__ = . ;
32         *(vtable)
33         *(.data*)
34
35         . = ALIGN(4);
36         /* preinit data */
37         KEEP(*(.preinit_array))
38
39         . = ALIGN(4);
40         /* init data */
41         KEEP(*(.init_array.*))
42         KEEP(*(.init_array))
43
44         . = ALIGN(4);
45         /* finit data */
46         KEEP(*(.fini_array.*))
47         KEEP(*(.fini_array))
48
49         . = ALIGN(4) ;
50         __data_end__ = . ;
51     } > sram AT> flash
52
53     .bss :
54     {
55         __bss_start__ = . ;
56         *(.bss*)
57         *(COMMON)
58         . = ALIGN(4) ;
59         __bss_end__ = . ;
60         . = ALIGN(4) ;
61         . = . + 0x1000 ;
62         __stack_top__ = . ;
63     } > sram
64 }

```

• Makefile

```
24 include sources.mk
25
26 BUILD = release
27 BUILDDIR = build
28 TARGET = Pressure_Controller
29 OBJS := $(SOURCES:.c=.o)
30 DEPS := $(SOURCES:.c=.d)
31
32 # Architectures Specific Flags
33 LINKER_FILE = linker_script.ld
34 CPU = cortex-m3
35 SPEC =
36
37 # Compiler Flags and Defines
38 CC = arm-none-eabi-gcc
39 LD = arm-none-eabi-ld
40 SIZE = arm-none-eabi-size
41 OBJDUMP = arm-none-eabi-objdump
42 OBJCOPY = arm-none-eabi-objcopy
43 LDFLAGS = -T $(LINKER_FILE) -Map=$(BUILDDIR)/$(TARGET).map
44 CFLAGS = -Wall -mcpu=$(CPU) $(INCLUDES) -mthumb
45 CPPFLAGS =
46
47 ifeq ($(BUILD), debug)
48 CFLAGS := -g3 -gdwarf-2 -O0 $(CFLAGS)
49 endif
50
51 # Cppcheck variables
52 DUMPS = $(SOURCES:.c=.c.dump)
53 CPPCHECK = cppcheck
54 MISRA := "$(shell which cppcheck)/../addons/misra.py"
55
56 .PHONY: build
57 build: make_dirs $(OBJS)
58     @echo Linking object files in to an executable...
59     @$(LD) $(LDFLAGS) $(addprefix $(BUILDDIR)/,$(notdir $(OBJS))) -o $(BUILDDIR)/$(TARGET).elf
60 ifeq ($(BUILD), debug)
61     @echo Generating assembly file from generated executable...
62     @$(OBJDUMP) -D -S $(BUILDDIR)/$(TARGET).elf > $(BUILDDIR)/$(TARGET).asm
63 endif
64     @echo Generating hex file from generated executable...
65     @$(OBJCOPY) -O ihex $(BUILDDIR)/$(TARGET).elf $(BUILDDIR)/$(TARGET).hex
66     @echo Executable and object final files sizes
67     @$(SIZE) -Btd $(BUILDDIR)/$(TARGET).elf $(addprefix $(BUILDDIR)/,$(notdir $(OBJS)))
68
69 .PHONY: compile-all
70 compile-all: $(OBJS)
71     @echo Finished generating object files
72
73 %.i: %.c
74     @$(CC) -E $(CFLAGS) -o $(BUILDDIR)/$(notdir $@) $<
75
76 %.asm: %.c %.o
77     @$(CC) -S $(CFLAGS) $<
78     @$(OBJDUMP) -D -S $(BUILDDIR)/$(notdir $(word 2,$*)) > $(BUILDDIR)/$(notdir $@)
79
80 %.o: %.c
81     @$(CC) -c $< $(CFLAGS) -o $(BUILDDIR)/$(notdir $@)
82
83 make_dirs:
84     @mkdir -p $(BUILDDIR)
85
86 .PHONY: clean
87 clean:
88     @echo Removing executable, hex and map files
89     @rm -f $(BUILDDIR)/*.elf $(BUILDDIR)/*.hex $(BUILDDIR)/*.map
90
91 .PHONY: clean_all
92 clean_all:
93     @echo Removing all files generated by make
94     @rm -f $(BUILDDIR)/.*
95
96 # Generate dump file from c file
97 %.c.dump: %.c
98     @$(CPPCHECK) --dump $(INCLUDES) $<
99     @mv $@ $(BUILDDIR)/$(notdir $@)
100
101 .PHONY: misra_compliant
102 misra_compliant: $(DUMPS)
103     @echo Displaying MISRA rules violation discovered by Cppcheck
104     @python $(MISRA) $(realpath $(addprefix $(BUILDDIR)/,$(notdir $(DUMPS))))
105
```

```
1 #-----
2 #
3 # file includes project sources and headers folder
4 #
5 #-----
6
7 SOURCES =    src/main.c           \
8              src/driver.c         \
9              src/pressure_sensor.c \
10             src/alarm_monitor.c   \
11             src/main_algorithm.c  \
12             src/alarm_actuator.c  \
13             src/startup.c
14
15 # Add your include paths to this variable
16 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)*

```
1
2 Memory Configuration
3
4 Name          Origin          Length          Attributes
5 flash         0x0000000008000000 0x0000000000000000 xrw
6 sram          0x0000000020000000 0x0000000000002800 xrw
7 *default*    0x0000000000000000 0xffffffffffffff
8
9 Linker script and memory map
10
11
12 .text         0x0000000008000000 0x40c
13 *(.vectors*)
14 .vectors      0x0000000008000000 0x10 build/startup.o
15              0x0000000008000000 vectorTable
16
17 *(.text*)
18 .text         0x0000000008000010 0x80 build/main.o
19              0x0000000008000010 setup
20              0x000000000800005c main
21 .text         0x0000000008000090 0xc0 build/driver.o
22              0x0000000008000090 Delay
23              0x00000000080000b2 getPressureVal
24              0x00000000080000c8 Set_Alarm_actuator
25              0x000000000800010c GPIO_Init
26 .text         0x0000000008000150 0x90 build/pressure_sensor.o
27              0x0000000008000150 ST_PressureSensor_init
28              0x0000000008000174 ST_PressureSensor_reading
29              0x00000000080001a4 ST_PressureSensor_waiting
30              0x00000000080001cc PressureSensor_GetPressureValue
31 .text         0x00000000080001e0 0x7c build/alarm_monitor.o
32              0x00000000080001e0 ST_AlarmMonitor_off
33              0x0000000008000210 ST_AlarmMonitor_on
34              0x0000000008000234 ST_AlarmMonitor_waiting
35 .text         0x000000000800025c 0x58 build/main_algorithm.o
36              0x000000000800025c ST_MainAlgorithm_highPresure
37              0x000000000800028c MainAlgorithm_HighPressureDetected
38 .text         0x00000000080002b4 0xc4 build/alarm_actuator.o
39              0x00000000080002b4 ST_AlarmActuator_off
40              0x00000000080002dc ST_AlarmActuator_on
41              0x0000000008000304 ST_AlarmActuator_init
42              0x0000000008000328 ST_AlarmActuator_waiting
43              0x0000000008000340 AlarmActuator_StartAlarm
44              0x000000000800035c AlarmActuator_StopAlarm
45 .text         0x0000000008000378 0x94 build/startup.o
46              0x0000000008000378 H_fault_Handler
47              0x0000000008000378 Default_Handler
48              0x0000000008000378 NMI_Handler
49              0x0000000008000384 Reset_Handler
```

- *Proteus simulation*

