Jet Cabinet Pressure Detection System

Embedded System Design Project

User Guide for a Complete & Successful Prototype

By Eng. Ziad Elmarakbi

1st Version Release

Mastering Advanced Embedded Systems Online Diploma

Website:

https://www.learn-in-depth.com/online-diploma/zezoalbendary20%40gmail.com

Linkedin:
https://www.linkedin.com/in/ziad-elmarakbi-1610231b2/

Github:
https://github.com/ZiadElmarakbi?tab=repositories

HackerRank:
https://www.hackerrank.com/zezoalbendary20

Publishing Date : 10/19/2022

Table of Contents

1.0 Introduction

In Aerospace, embedded systems are used excessively. Some aircrafts have embedded systems at work from tip to tail, while others are using embedded systems in very specific ways. In each case, no matter how the aerospace applications embedded systems are being used, they have to be designed to withstand extreme conditions without any kind of loss of function. Some embedded systems perform critical functions that are responsible for passenger and operator safety, so only the highest calibre solutions are an option. In this project, a prototype of a cabinet high-pressure detection embedded system was architectured then simulated to mimic how would the system behave during an actual implementation.

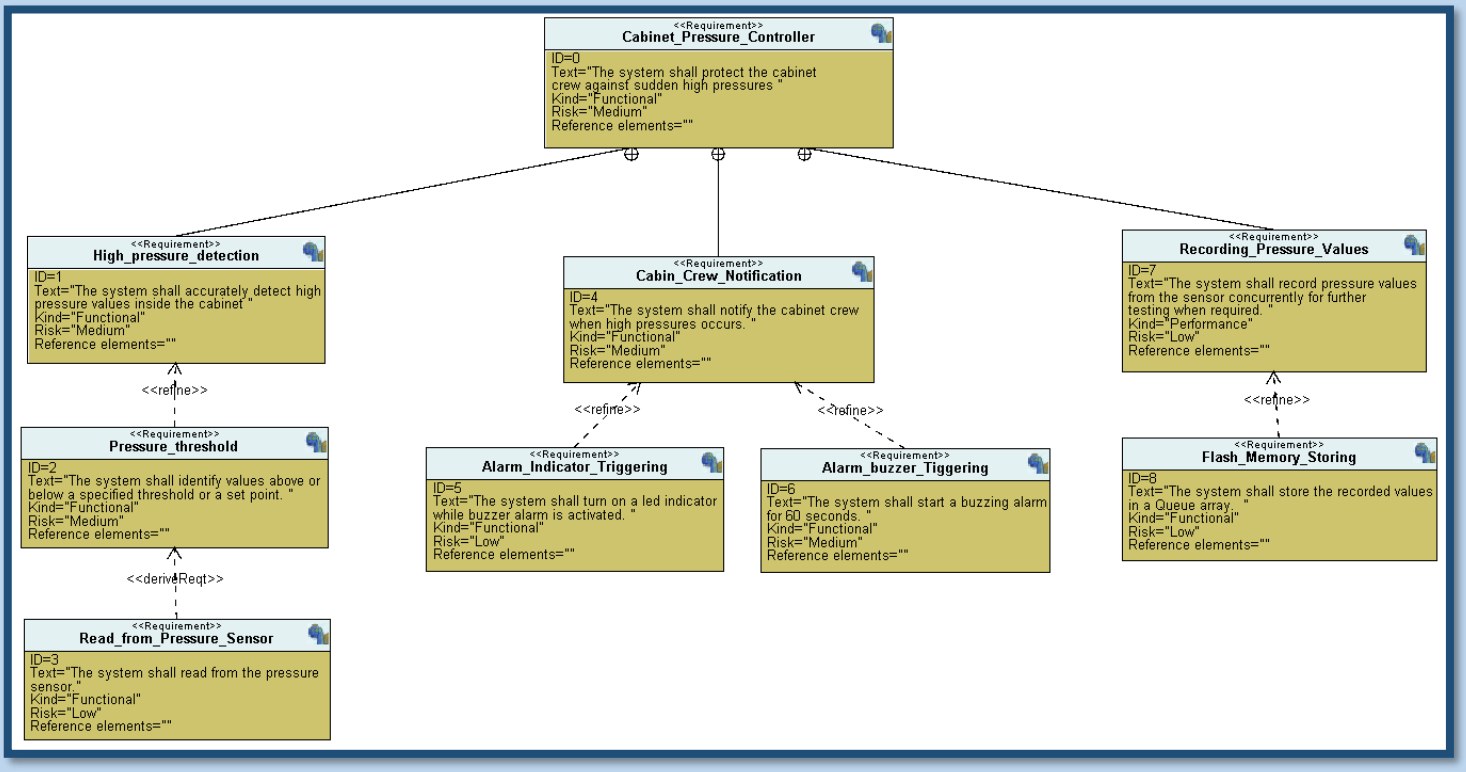
2.0 Case Study

The details and requirements for the jet cabinet high pressure detection project were discussed upon meeting with the client. The system is required to detect moderate to high pressure ranges at different setpoints then actuate an alarm. The alarm consists of a buzzer and three LEDs with green, yellow, and red colours respectively. If the pressure sensor detected any range of pressure values below 10 bars, the green LED will be activated for 3 seconds with no buzzer beeps. And if the pressure values fall in between 10 – 20 bars, the yellow LED will be activated and the buzzer will toggle (beeping on and off) for 12 seconds. Lastly, if the pressure values are above 20 bars, the red LED and the buzzer will be activated with a (constant long beep) for 18 seconds. During the operation of the system the values read from the pressure sensor has to be stored in flash memory. However, there was some assumptions and limitations about the system that were discussed with the client. Thus, a group of policies were represented to the client which was agreed on before signing the contract to disclaim complete responsibility when delivering the project. Those policies are as follows:

1. The client is liable to replace or repair any damaged components delivered for constructing the project prototype.
2. The client is liable to bring the correct models of the requested components as stated in the provided specification sheet.
3. The client is liable for the proper installations of the delivered system.
4. The client is liable for periodical maintenance for the supplied components. As our company will only provide maintenance for the functional system and its performance.
5. The client shall supply the requested components in the required dates.

Failure to fulfil any of the stated policies might affect the finalized system performance and will definitely affect the client's requested deadlines of submittals.

3.0 Requirements Diagram



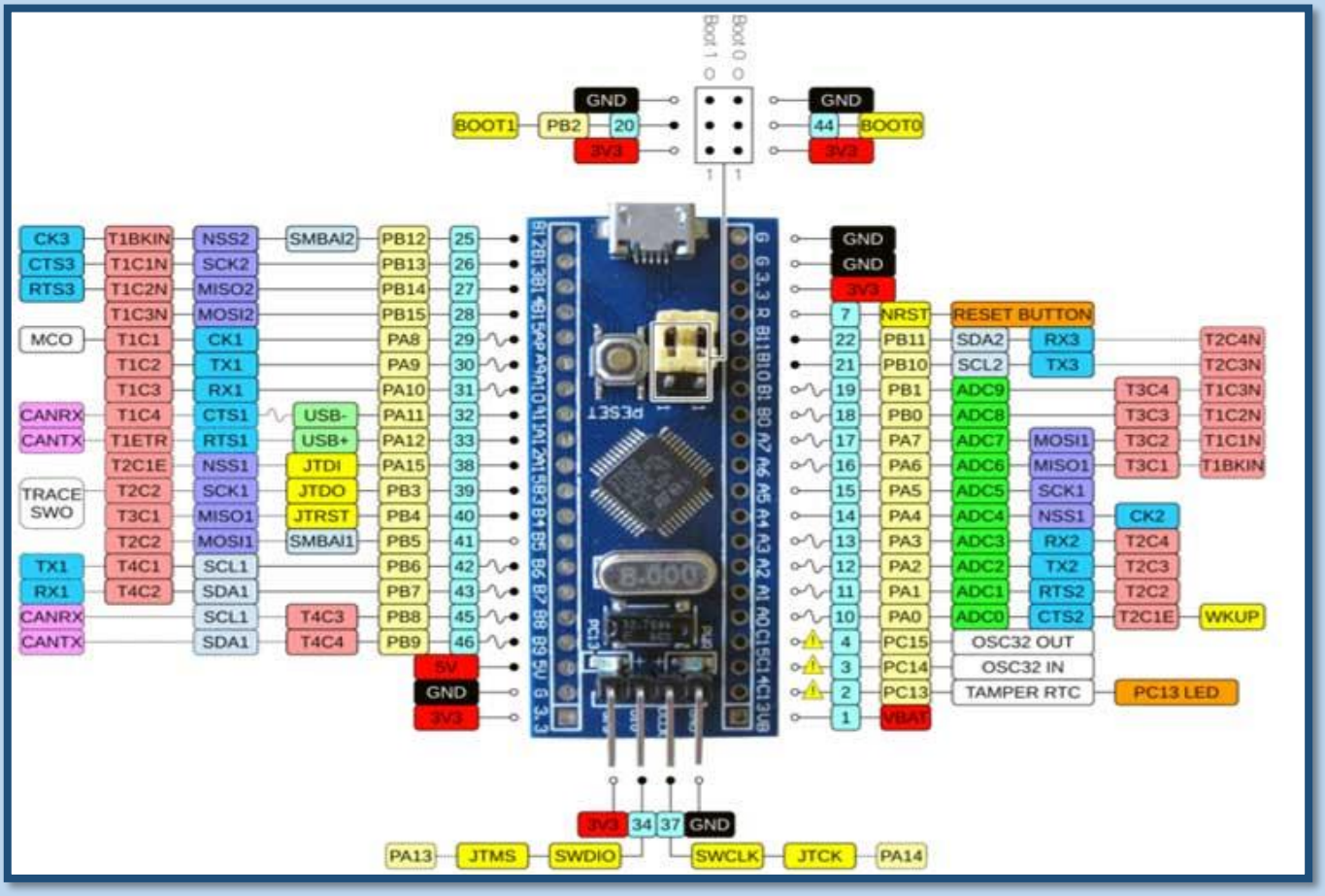
4.0 Space Exploration/Partitioning

The **STM32F103C6** System on chip **(SoC)** was used in this project. It's from the mainstream performance line family which incorporates the high-performance ARM® Cortex™-M3 32-bit RISC core operating at a **72 MHz** frequency, high-speed embedded memory (Flash memory up to **32 Kbytes** and SRAM up to **6 Kbytes**), and an extensive range of enhanced I/Os and peripherals connected to two APB buses. The device offers two **12-bit** ADCs, three general purpose 16-bit timers plus one PWM timer, as well as standard and advanced communication interfaces: up to two I^2Cs and SPIs, three USARTs, an USB and a CAN.

The STM32F103xx low-density performance line family operates from a **2.0 to 3.6 V** power supply. It is available in both the **–40 to +85 °C** temperature range and the **–40 to +105 °C**

extended temperature range. A comprehensive set of power-saving mode allows the design of low-power applications.

These features make all the STM32F103xx low-density performance line microcontroller family suitable for a wide range of applications such as motor drives, application control, medical and handheld equipment, PC and gaming peripherals, GPS platforms, industrial applications, PLCs, inverters, printers, scanners, alarm systems, video intercoms, and HVACs. Thus, it's a very powerful choice for this project's application. In fact, if cost was a considered factor in this project. This SoC would be overrated. The Pin layout of the STM32F103C6 is shown in the figure below.



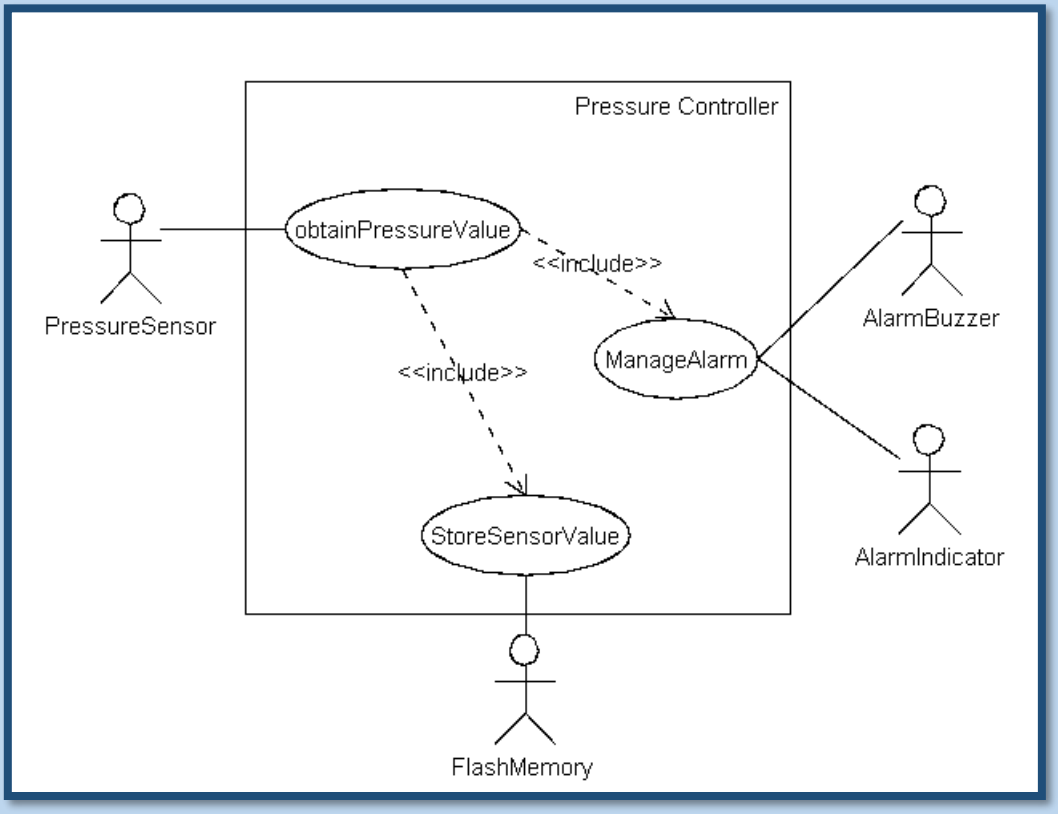
5.0 UML 2.0

UML (Unified Modelling Language) is a design language that is often used to develop and build computer applications. It consists of a family of graphical notations that assists in describing and designing software systems. It is mainly employed in the systems developed

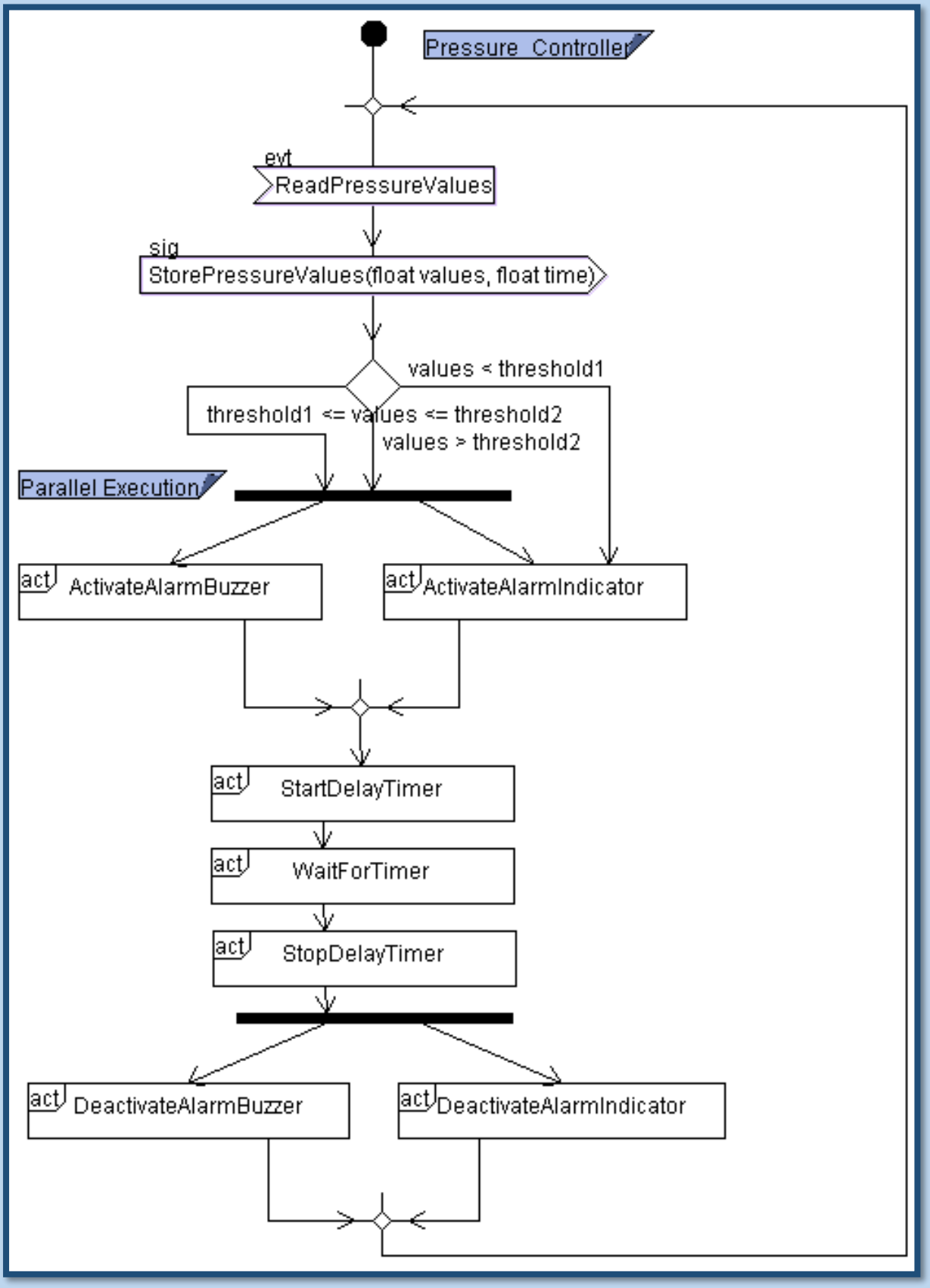
using an object-oriented style. UML is independent of implementation language. It can be used at various stages like analysis, design and programming. UML has undergone several phases of evolution. UML 1.0 is based on the industry standard for object-oriented modelling. However, UML 2.0 has been an industry standard focusing on the model-driven application integration. UML 2.0 has various advantages over UML 1.x (all version of UML 1.0) as many new powerful concepts have been added in UML 2.0. UML 2.0 is capable to provide better semantics or definitions. It has also worked to improve the internal structuring. There are numerous kinds of UML diagrams. However, the diagrams used in this project are the use case, activity, sequence, and state machine diagrams and they were created using TTool program.

5.1 System Analysis

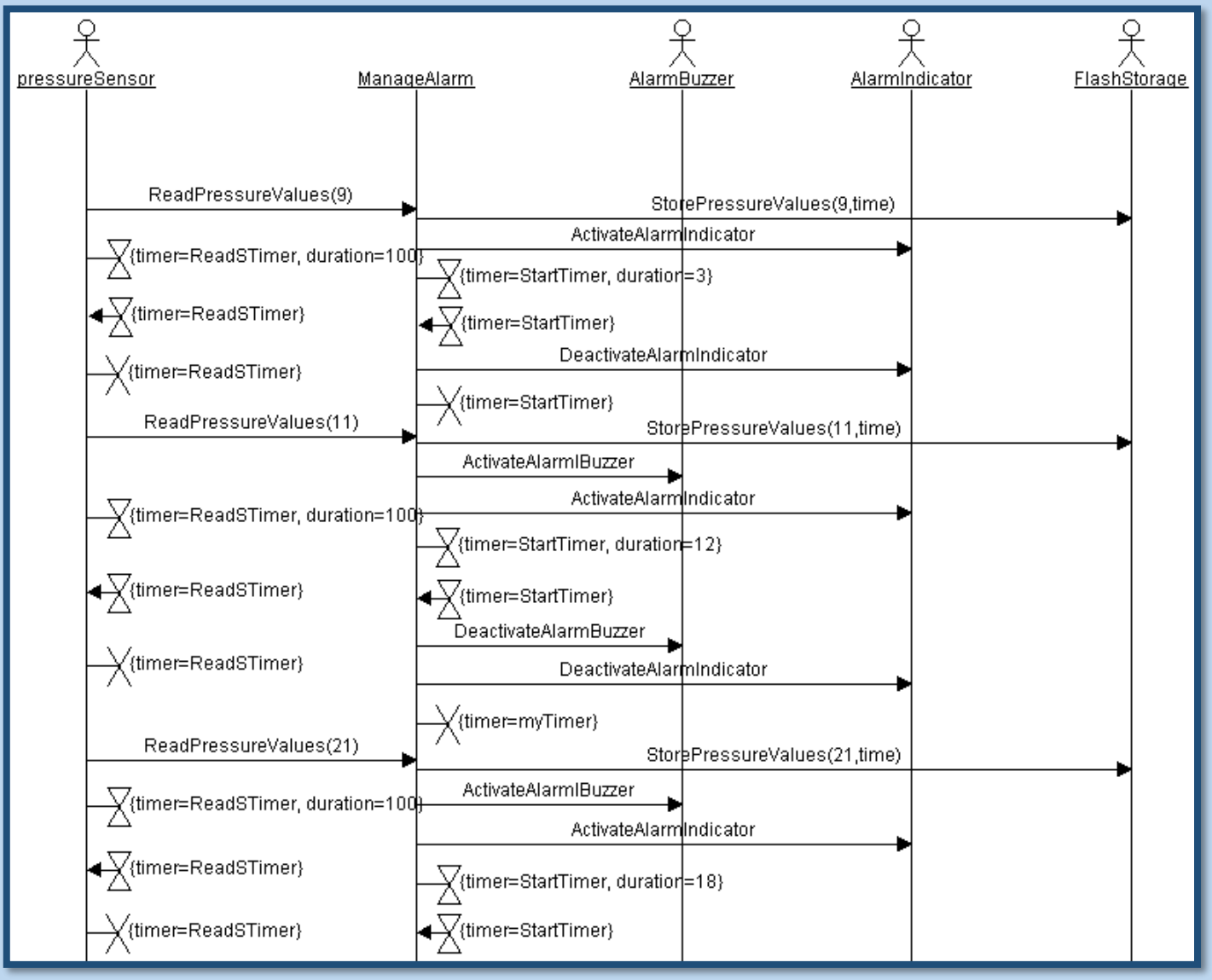
5.1.0 Use Case Diagram



5.1.1 Activity Diagram

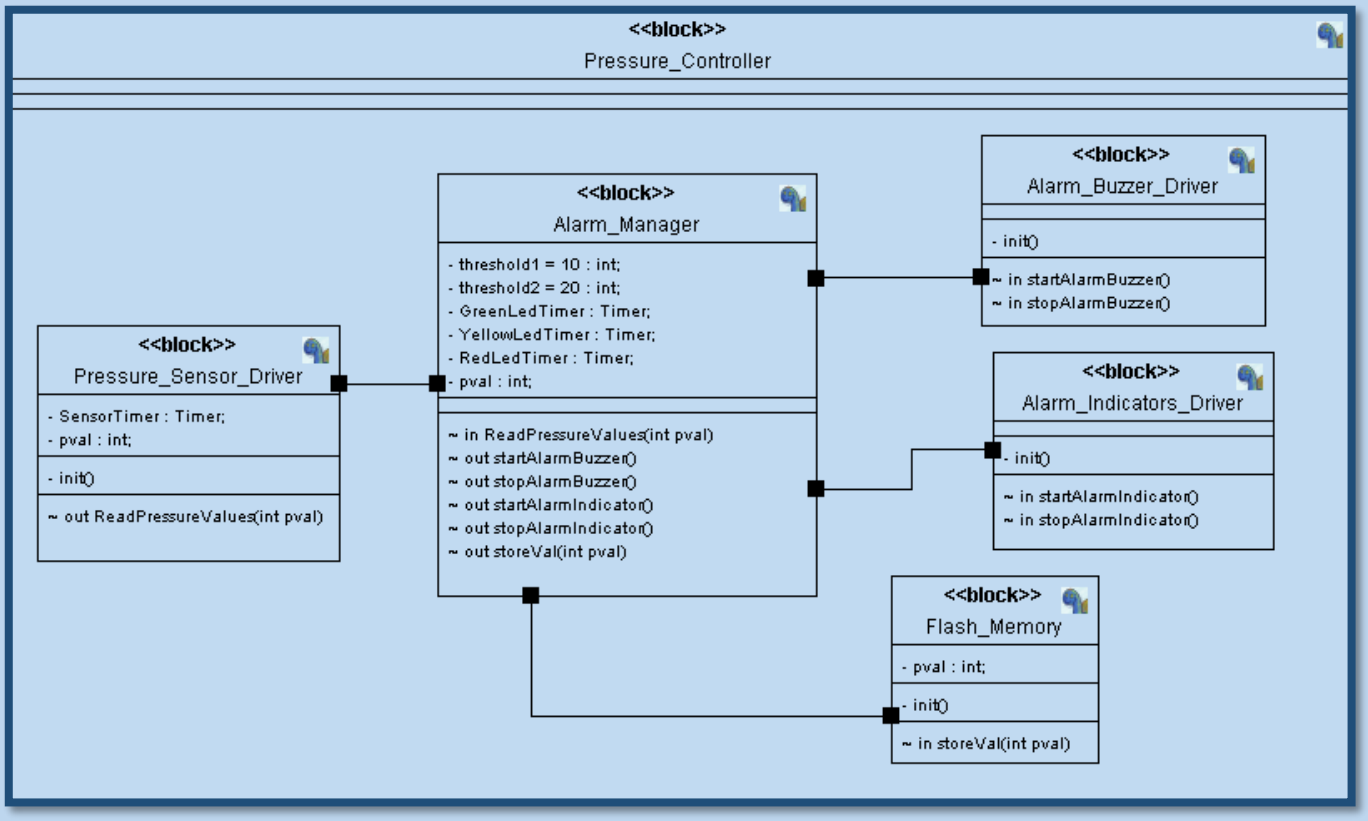


5.1.2 Sequence Diagram

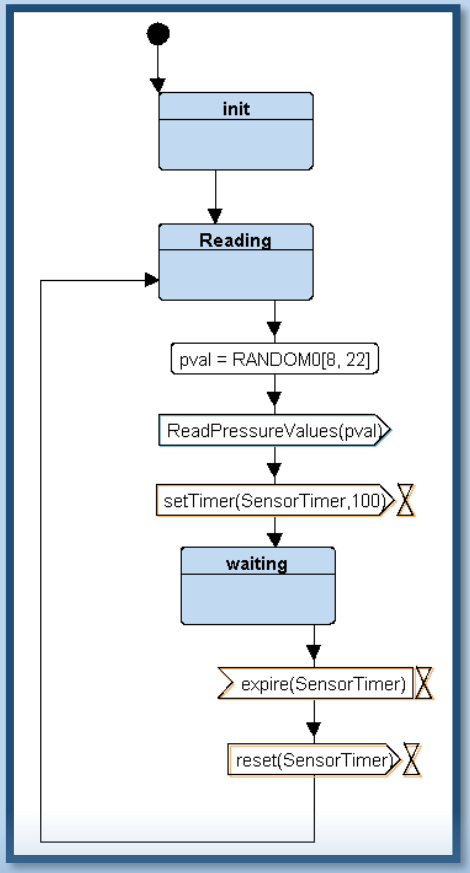


5.2 System Design

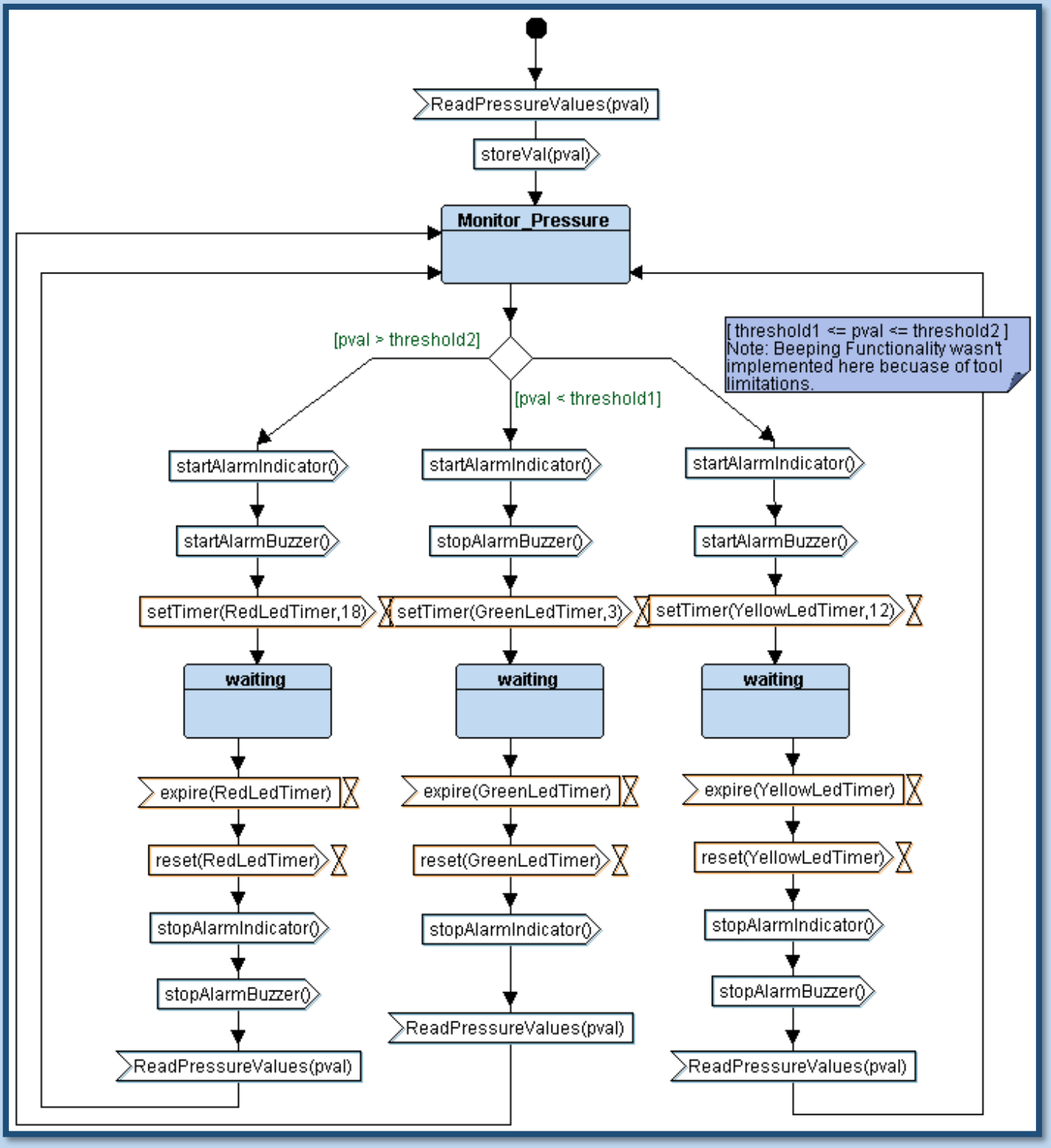
5.2.0 Block Diagram



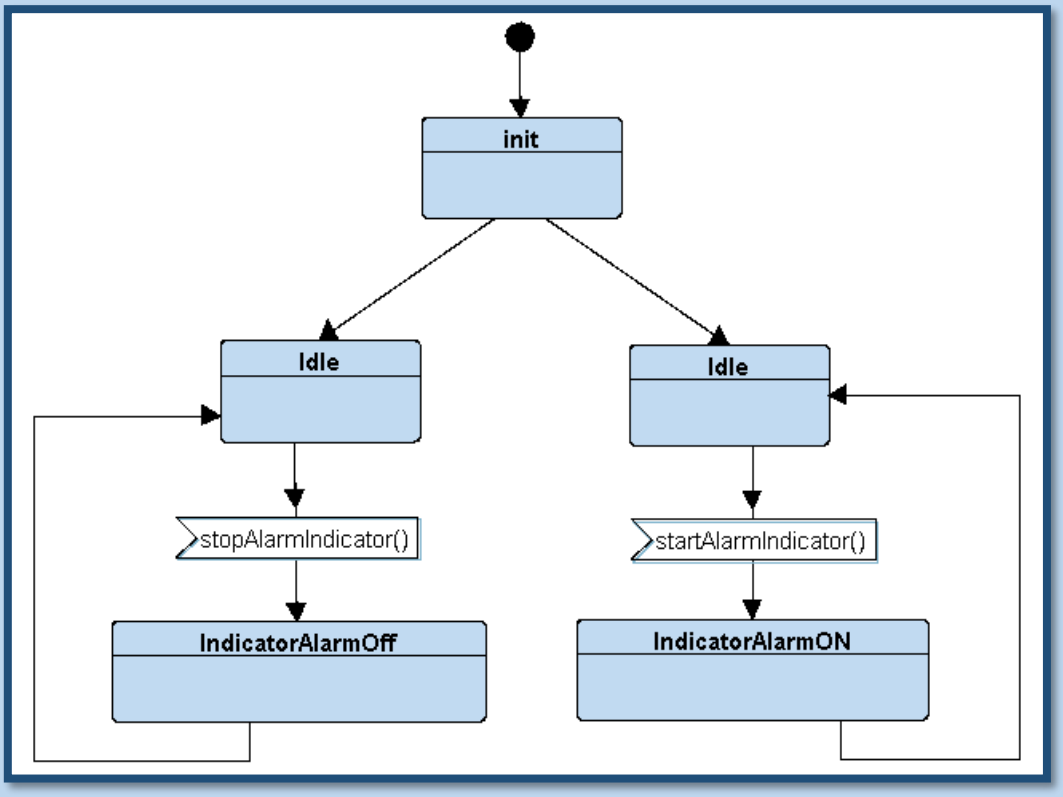
5.2.1 **State Machine 1:** Pressure Sensor Driver



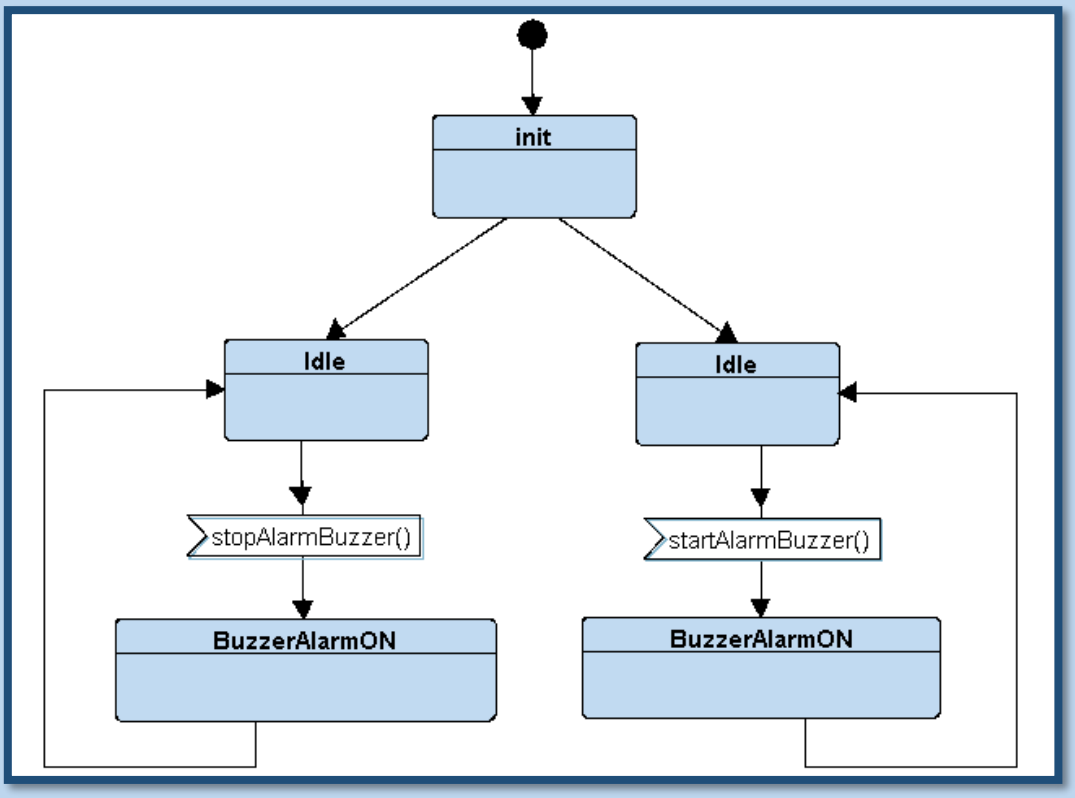
5.2.2 State Machine 2: Alarm Manager



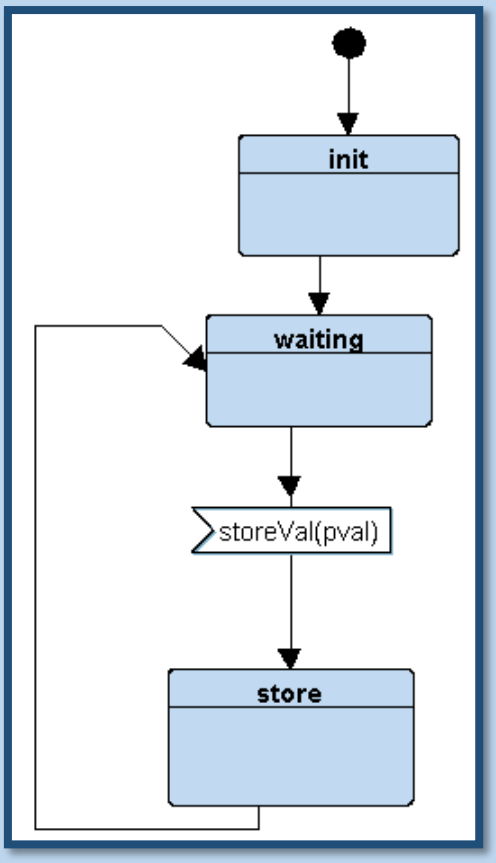
5.2.3 State Machine 3: Alarm Indicator Driver



5.2.4 State Machine 4: Alarm Buzzer Driver

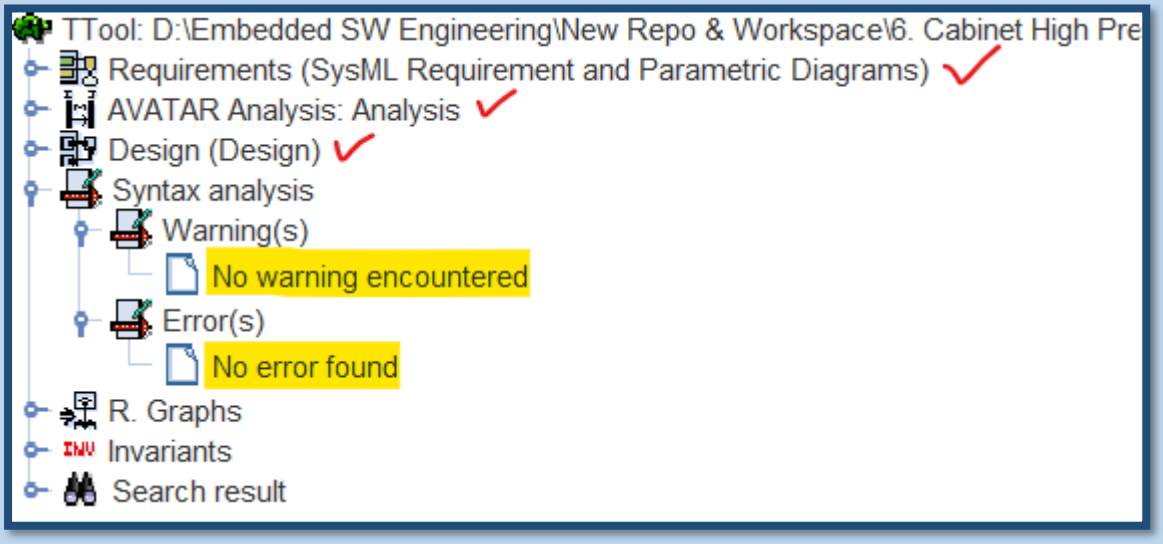


5.2.5 State Machine 5: Flash Memory Driver

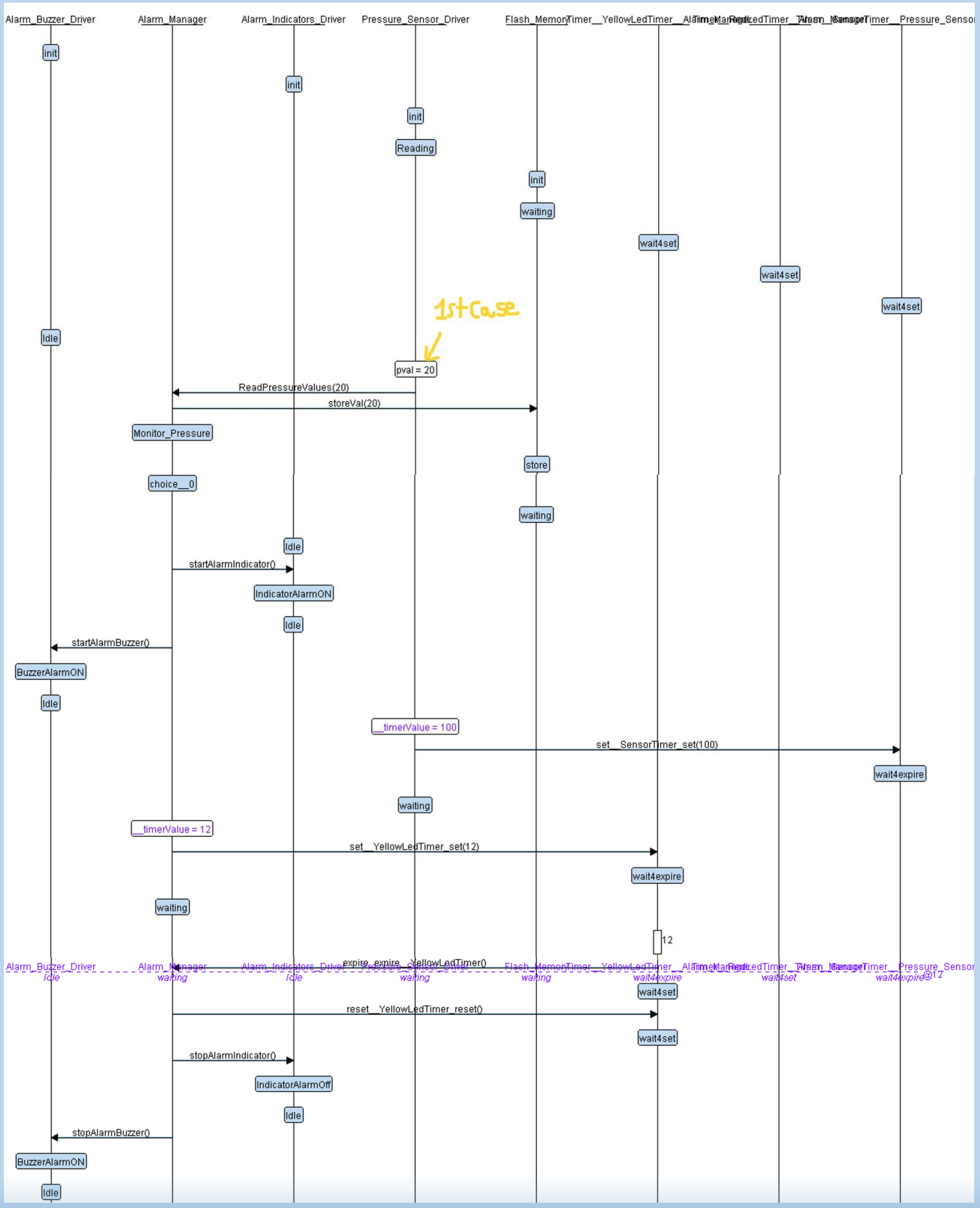


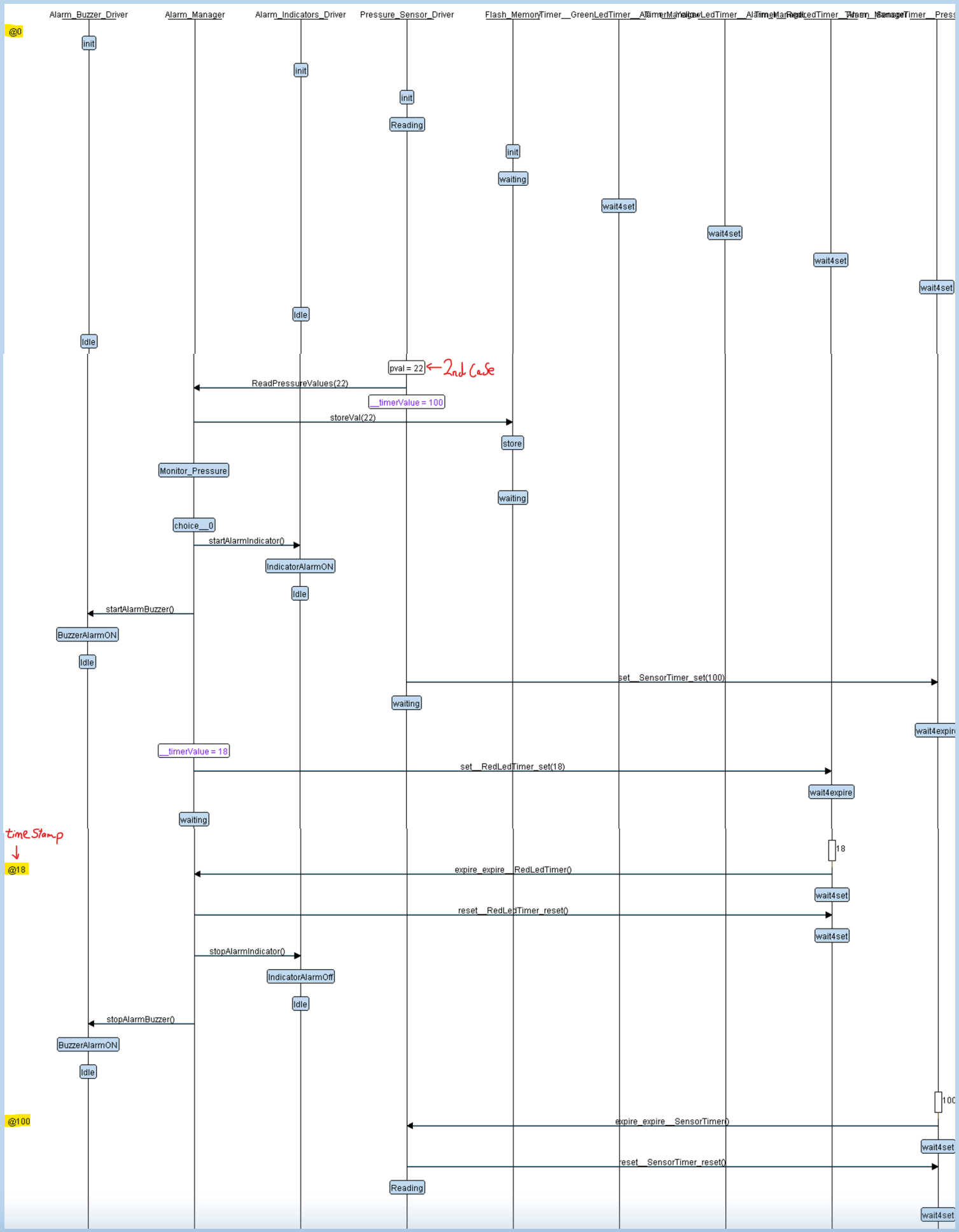
6.0 State Sequence Verification

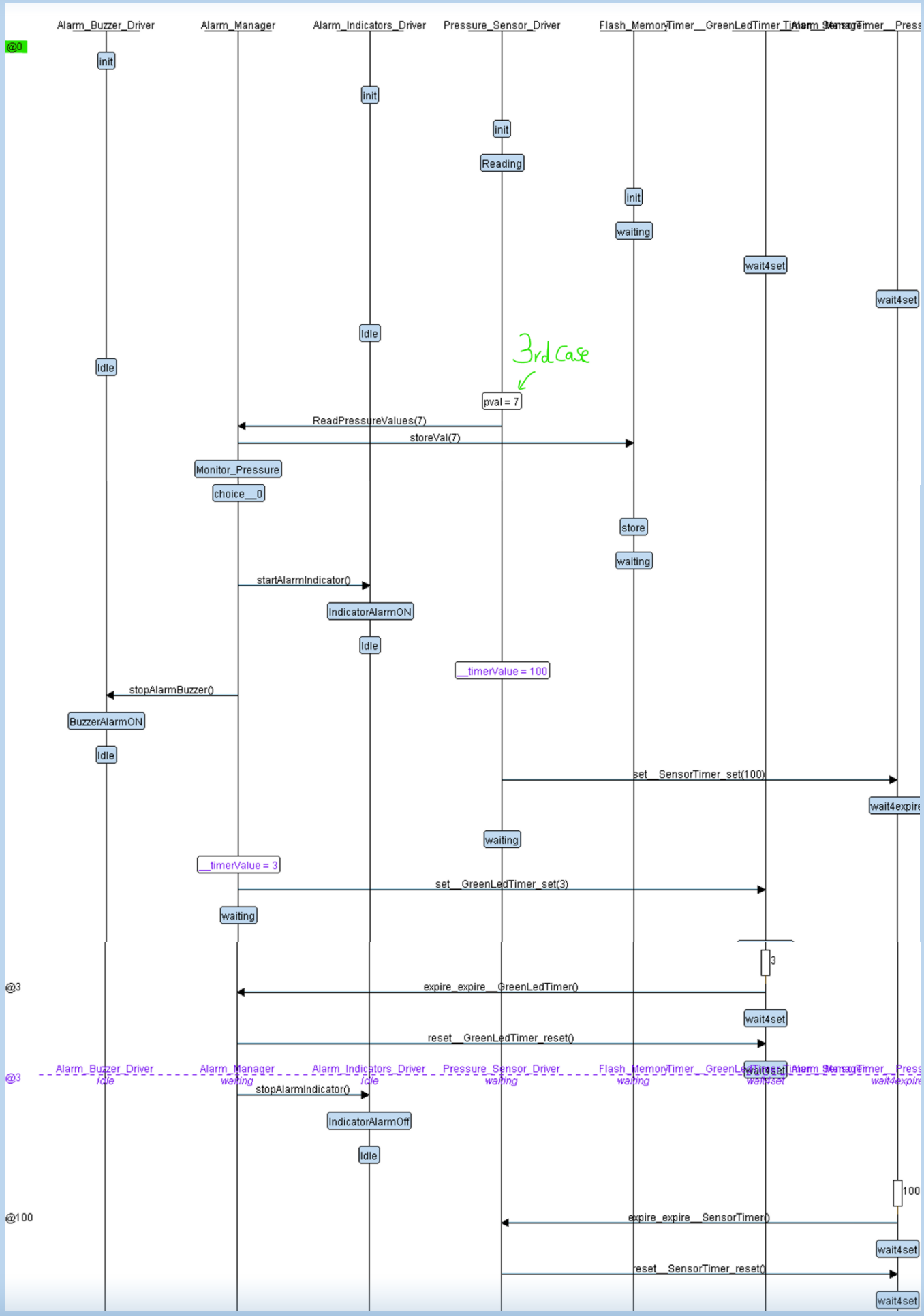
6.1 UML Tree Completion



6.2 Sequence Validation




Alarm_Buzzer_Driver
Alarm_Manager
Alarm_Indicators_Driver
Pressure_Sensor_Driver
Flash_MemoryTimer__GreenLedTimer__
@0
init
Reading
waiting
wait4set
Idle
pval = 22
2nd Case
ReadPressureValues(22)
__timerValue = 100
storeVal(22)
store
Monitor_Pressure
choice__0
startAlarmIndicator()
IndicatorAlarmON
startAlarmBuzzer()
BuzzerAlarmON
set__SensorTimer_set(100)
__timerValue = 18
set__RedLedTimer_set(18)
wait4expire
timeStamp
@18
18
expire_expire__RedLedTimer()
reset__RedLedTimer_reset()
stopAlarmIndicator()
IndicatorAlarmOff
stopAlarmBuzzer()
@100
expire_expire__SensorTimer()
reset__SensorTimer_reset()


Alarm_Buzzer_Driver
Alarm_Manager
Alarm_Indicators_Driver
Pressure_Sensor_Driver
Flash_MemoryTimer__GreenLedTimer
@0
init
Reading
waiting
wait4set
Idle
3rd Case
pval = 7
ReadPressureValues(7)
storeVal(7)
Monitor_Pressure
choice__0
store
startAlarmIndicator()
IndicatorAlarmON
__timerValue = 100
stopAlarmBuzzer()
BuzzerAlarmON
set__SensorTimer_set(100)
__timerValue = 3
set__GreenLedTimer_set(3)
@3
expire_expire__GreenLedTimer()
reset__GreenLedTimer_reset()
stopAlarmIndicator()
IndicatorAlarmOff
@100
expire_expire__SensorTimer()
reset__SensorTimer_reset()

7.0 System Implementation

The Designed System was employed by using both native and cross tool chains. Coherently, the compilation, assembling, and linking process were done by utilizing the Windows host machine open-source native tool chain “GNU” in which the software sequence of operation was validated. However, the implemented code was intended to be burned out on a target system on chip ‘SoC’. The STM target SoC has an ARM based processor. Thus, in order for the processor to recognize the burned hex or binary executable code, the ARM based cross tool chain “GNU” was utilized.

7.1 Windows Host Machine Native Tool Chain Utilization Using Eclipse IDE

Please note that the created functions in all the following C files were collapsed except the Main algorithm file to only view a layout of the script for each module and not the entire code. For fully accessing all the header and source files in detail please visit: https://github.com/ZiadElmarakbi/Embedded-Systems/tree/master/6.%20Jet%20Cabinet%20Pressure%20Detection%20System%20Project

7.1.0 Main Algorithm.c

```
// ---------------------- MainAlgorithm.c ----------------------//
// Created on: October 13, 2022
// Author: ZIAD ELMARAKBI

#include "Main_Init.h"
#include "States.h"

void setup(){

    //--------------- MCAL Initialization ---------------//
    GPIO_INITIALIZATION();

    //--------------- HAL Initialization ---------------//
    Pressure_Sensor_init();
    Alarm_Indicator_init();
    Alarm_Buzzer_init();
    Flash_Memory_init(flashptr, flashArr, flashSpace);

    //--------------- Block Initialization ---------------//
    Alarm_Manager_init();
}

int main(){

    setup();

    while(1)
    {
        Pressure_Sensor_ptr2Fun();
        Delay(1000);
        Alarm_Manager_ptr2Fun();
        Delay(1000);
    }
    return 0;
}

uint32_t generate_random(uint32_t x, uint32_t y, uint32_t count){
```

7.1.1 Pressure Sensor Driver.c

```
#include "Pressure_Sensor.h"

// Pointer to function to alternate between the states
void (*Pressure_Sensor_ptr2Fun)();

// To Store the pressure sensor values
uint32_t pval;

// Initializing the Pressure Sensor
void Pressure_Sensor_init(){

// Defining the Reading Pressure State
Define_State(ReadingPressureVal){

// Defining the Waiting Pressure State
Define_State(WaitingPressureVal){
```

7.1.2 Flash Memory Driver.c

```
#include "Flash_Memory_Driver.h"

fifo_flash flash; // Creating an Object From a FIFO Structure
fifo_flash* flashptr = &flash; // Creating a Pointer to a FIFO Structure
uint32_t flashArr[flashSpace]; // Creating a Flash Memory Storage

// Enumerating Flash Memory States
enum {

// Pointer to function to alternate between states
void (*Flash_Memory_ptr2Fun)(uint32_t);

// Initializing Flash Memory
void Flash_Memory_init(fifo_flash* flashptr, uint32_t* flashArr, uint32_t Length){

// Defining the Storing State
void Storing_State(uint32_t pval){

// Defining the Waiting State
void Waiting_State(uint32_t pval){

// Defining a function to Enqueue Values into the Flash Memory Queue Array
Flash_Memory_Status Flash_Memory_Enqueue(fifo_flash* flashptr, uint32_t* pval){
```

7.1.3 Alarm Manager.c

```
#include "Alarm_manager.h"

// Pointer to function to alternate between states
void (*Alarm_Manager_ptr2Fun)();

// Initializing Alarm Manager
void Alarm_Manager_init(){

// Defining Monitoring State
Define_State(MonitorPressure){

// Defining Waiting State
Define_State(Waiting){
```

7.1.4 Alarm Buzzer Driver.c

```
#include "Alarm_Buzzer_Driver.h"

void (*Alarm_Buzzer_ptr2Fun)();

void Alarm_Buzzer_init(){

// Defining a function to call the Alarming State
void startAlarmBuzzer(){

// Defining a function to call the Idle State
void stopAlarmBuzzer(){

Define_State(BuzzAlarm){

Define_State(BuzzIdle){
```

7.1.5 Alarm Indicator Driver.c

```
#include "Alarm_Indicator_Driver.h"

void (*Alarm_Indicator_ptr2Fun)(Indicator_status);

void Alarm_Indicator_init(){

void AlarmIndicatorOFF(Indicator_status LEDstatus){

void AlarmIndicatorON(Indicator_status LEDstatus){

void AlarmIndication_State(Indicator_status LEDstatus){

void AlarmIdle_State(Indicator_status LEDstatus){
```

7.2 ARM Target SoC Cross Tool Chain Utilization Using Command Prompt

7.2.0 Command Line Scripting (Without MakeFile Automation)

Cross-Toolchain Utilization.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\Embedded SW Engineering\New Repo & Workspace\6. Cabinet High Pressure Detection System Project\Modules & Drivers>arm-none-eabi-gcc.exe -c -mcpu=cortex
-m3 -mthumb -gdwarf-2 -I . Startup.c -o Startup.o
PS D:\Embedded SW Engineering\New Repo & Workspace\6. Cabinet High Pressure Detection System Project\Modules & Drivers>arm-none-eabi-gcc.exe -c -mcpu=cortex
-m3 -mthumb -gdwarf-2 -I . Main_Algorithm.c -o Main_Algorithm.o
PS D:\Embedded SW Engineering\New Repo & Workspace\6. Cabinet High Pressure Detection System Project\Modules & Drivers>arm-none-eabi-gcc.exe -c -mcpu=cortex
-m3 -mthumb -gdwarf-2 -I . Pressure_Sensor.c -o Pressure_Sensor.o
PS D:\Embedded SW Engineering\New Repo & Workspace\6. Cabinet High Pressure Detection System Project\Modules & Drivers>arm-none-eabi-gcc.exe -c -mcpu=cortex
-m3 -mthumb -gdwarf-2 -I . Alarm_manager.c -o Alarm_manager.o
PS D:\Embedded SW Engineering\New Repo & Workspace\6. Cabinet High Pressure Detection System Project\Modules & Drivers>arm-none-eabi-gcc.exe -c -mcpu=cortex
-m3 -mthumb -gdwarf-2 -I . Flash_Memory_Driver.c -o Flash_Memory_Driver.o
PS D:\Embedded SW Engineering\New Repo & Workspace\6. Cabinet High Pressure Detection System Project\Modules & Drivers>arm-none-eabi-gcc.exe -c -mcpu=cortex
-m3 -mthumb -gdwarf-2 -I . Alarm_Buzzer_Driver.c -o Alarm_Buzzer_Driver.o
PS D:\Embedded SW Engineering\New Repo & Workspace\6. Cabinet High Pressure Detection System Project\Modules & Drivers>arm-none-eabi-gcc.exe -c -mcpu=cortex
-m3 -mthumb -gdwarf-2 -I . Alarm_Indicator_Driver.c -o Alarm_Indicator_Driver.o
PS D:\Embedded SW Engineering\New Repo & Workspace\6. Cabinet High Pressure Detection System Project\Modules & Drivers>arm-none-eabi-gcc.exe -c -mcpu=cortex
-m3 -mthumb -gdwarf-2 -I . BareMetalDriver.c -o BareMetalDriver.o
PS D:\Embedded SW Engineering\New Repo & Workspace\6. Cabinet High Pressure Detection System Project\Modules & Drivers>arm-none-eabi-ld.exe -T LinkerScript.
ld -Map=Mapping_file.map Startup.o Main_Algorithm.o Pressure_Sensor.o Alarm_manager.o Flash_Memory_Driver.o Alarm_Buzzer_Driver.o Alarm_Indicator_Driver.o B
areMetalDriver.o -o Pressure_Detection_System.elf
C:\ARM_Toolchain\bin\arm-none-eabi-ld.exe: cannot find .map: No such file or directory
PS D:\Embedded SW Engineering\New Repo & Workspace\6. Cabinet High Pressure Detection System Project\Modules & Drivers>arm-none-eabi-ld.exe -T LinkerScript.
ld Startup.o Main_Algorithm.o Pressure_Sensor.o Alarm_manager.o Flash_Memory_Driver.o Alarm_Buzzer_Driver.o Alarm_Indicator_Driver.o BareMetalDriver.o -o Pr
essure_Detection_System.elf                    PS D:\Embedded SW Engineering\New Repo & Workspace\6. Cabinet High Pressure Detection System Project\Modules
 & Drivers>arm-none-eabi-objdump.exe -D Pressure_Detection_System.elf >> Pressure_Detection_System_Analysis.txt
PS D:\Embedded SW Engineering\New Repo & Workspace\6. Cabinet High Pressure Detection System Project\Modules & Drivers>arm-none-eabi-objdump.exe -h Pressure
_Detection_System.elf >> System_VMA_LMA_Sections.txt
PS D:\Embedded SW Engineering\New Repo & Workspace\6. Cabinet High Pressure Detection System Project\Modules & Drivers>arm-none-eabi-nm.exe Pressure_Detecti
on_System.elf >> System_Symbolic_Table.txt
PS D:\Embedded SW Engineering\New Repo & Workspace\6. Cabinet High Pressure Detection System Project\Modules & Drivers>arm-none-eabi-objcopy.exe -O binary P
ressure_Detection_System.elf System_Executable.bin
PS D:\Embedded SW Engineering\New Repo & Workspace\6. Cabinet High Pressure Detection System Project\Modules & Drivers>history
```

All the C files have been cross compiled, assembled, and linked. After linking all the object files. The generated. elf image was disassembled, and sectionalized to observe all the virtual and loading memory addresses (VMA and LMA) as well as to view all the created symbols after linking for further analysis. Additionally, a .map file and a binary image were created to show full analysis and to be burned on the STM targeted SoC respectively.

Command Line History

```
 Id CommandLine
 -- -----------
  1 arm-none-eabi-gcc.exe -c -mcpu=cortex-m3 -mthumb -gdwarf-2 -I . Startup.c -o Startup.o
  2 arm-none-eabi-gcc.exe -c -mcpu=cortex-m3 -mthumb -gdwarf-2 -I . Main_Algorithm.c -o Main_Algorithm.o
  3 arm-none-eabi-gcc.exe -c -mcpu=cortex-m3 -mthumb -gdwarf-2 -I . Pressure_Sensor.c -o Pressure_Sensor.o
  4 arm-none-eabi-gcc.exe -c -mcpu=cortex-m3 -mthumb -gdwarf-2 -I . Alarm_manager.c -o Alarm_manager.o
  5 arm-none-eabi-gcc.exe -c -mcpu=cortex-m3 -mthumb -gdwarf-2 -I . Flash_Memory_Driver.c -o Flash_Memory_Driver.o
  6 arm-none-eabi-gcc.exe -c -mcpu=cortex-m3 -mthumb -gdwarf-2 -I . Alarm_Buzzer_Driver.c -o Alarm_Buzzer_Driver.o
  7 arm-none-eabi-gcc.exe -c -mcpu=cortex-m3 -mthumb -gdwarf-2 -I . Alarm_Indicator_Driver.c -o Alarm_Indicator_Driver.o
  8 arm-none-eabi-gcc.exe -c -mcpu=cortex-m3 -mthumb -gdwarf-2 -I . BareMetalDriver.c -o BareMetalDriver.o
  9 arm-none-eabi-ld.exe -T LinkerScript.ld -Map=Mapping_file.map Startup.o Main_Algorithm.o Pressure_Sensor.o Alarm_manager.o Flash_Memory_Driver.o Al.
 10 arm-none-eabi-ld.exe -T LinkerScript.ld Startup.o Main_Algorithm.o Pressure_Sensor.o Alarm_manager.o Flash_Memory_Driver.o Alarm_Buzzer_Driver.o Al.
 11 arm-none-eabi-objdump.exe -D Pressure_Detection_System.elf >> Pressure_Detection_System_Analysis.txt
 12 arm-none-eabi-objdump.exe -h Pressure_Detection_System.elf >> System_VMA_LMA_Sections.txt
 13 arm-none-eabi-nm.exe Pressure_Detection_System.elf >> System_Symbolic_Table.txt
 14 arm-none-eabi-objcopy.exe -O binary Pressure_Detection_System.elf System_Executable.bin
```

7.2.1 MakeFile

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


all: $(PROJECTNAME).bin
	@echo "-----------------Build Done Successfully !!-----------------"


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


$(PROJECTNAME).elf: $(OBJ)
	$(CC)ld.exe -T LinkerScript.ld $(OBJ) -o $@ -Map=Maped_file.map



$(PROJECTNAME).bin: $(PROJECTNAME).elf
	$(CC)objcopy.exe -O binary $<  $@
	$(CC)objdump.exe -D $< >>Pressure_Detection_System_Analysis.txt
	$(CC)objdump.exe -h $< >>System_VMA_LMA_Sections.txt
	$(CC)nm.exe $< >>System_Symbolic_Table.txt


clean_all:
	rm *.o *.elf *.bin *.txt *.map
	@echo "-----------All Files Removed Successfully !!------------"


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

MakeFile Automation

```
Ziad@FX506HC MINGW64 /d/Embedded SW Engineering/New Repo & Workspace/6. Cabinet High Pressure Detection System Project/Modules & Dri
vers (master)
$ make
arm-none-eabi-gcc.exe -c -mcpu=cortex-m3 -mthumb -gdwarf-2 -I . Alarm_Buzzer_Driver.c -o Alarm_Buzzer_Driver.o
arm-none-eabi-gcc.exe -c -mcpu=cortex-m3 -mthumb -gdwarf-2 -I . Alarm_Indicator_Driver.c -o Alarm_Indicator_Driver.o
arm-none-eabi-gcc.exe -c -mcpu=cortex-m3 -mthumb -gdwarf-2 -I . Alarm_manager.c -o Alarm_manager.o
arm-none-eabi-gcc.exe -c -mcpu=cortex-m3 -mthumb -gdwarf-2 -I . BareMetalDriver.c -o BareMetalDriver.o
arm-none-eabi-gcc.exe -c -mcpu=cortex-m3 -mthumb -gdwarf-2 -I . Flash_Memory_Driver.c -o Flash_Memory_Driver.o
arm-none-eabi-gcc.exe -c -mcpu=cortex-m3 -mthumb -gdwarf-2 -I . Main_Algorithm.c -o Main_Algorithm.o
arm-none-eabi-gcc.exe -c -mcpu=cortex-m3 -mthumb -gdwarf-2 -I . Pressure_Sensor.c -o Pressure_Sensor.o
arm-none-eabi-gcc.exe -c -mcpu=cortex-m3 -mthumb -gdwarf-2 -I . Startup.c -o Startup.o
arm-none-eabi-ld.exe -T LinkerScript.ld Alarm_Buzzer_Driver.o Alarm_Indicator_Driver.o Alarm_manager.o BareMetalDriver.o Flash_Memor
y_Driver.o Main_Algorithm.o Pressure_Sensor.o Startup.o -o Pressure_Detection_System.elf -Map=Maped_file.map
arm-none-eabi-objcopy.exe -O binary Pressure_Detection_System.elf  Pressure_Detection_System.bin
arm-none-eabi-objdump.exe -D Pressure_Detection_System.elf >>Pressure_Detection_System_Analysis.txt
arm-none-eabi-objdump.exe -h Pressure_Detection_System.elf >>System_VMA_LMA_Sections.txt
arm-none-eabi-nm.exe Pressure_Detection_System.elf >>System_Symbolic_Table.txt
-----------------Build Done Successfully !!-----------------

Ziad@FX506HC MINGW64 /d/Embedded SW Engineering/New Repo & Workspace/6. Cabinet High Pressure Detection System Project/Modules & Dri
vers (master)
$ make clean_all
rm *.o *.elf *.bin *.txt *.map
-----------All Files Removed Successfully !!------------
```

7.2.2 Linker Script.ld File

```
/* Linker_Script Cortex M3
Eng. Ziad Elmarakbi
*/

MEMORY
{
flash(rx) : ORIGIN = 0x08000000, LENGTH = 128k
sram(rwx) : ORIGIN = 0x20000000, LENGTH = 20k
}

SECTIONS
{

.text : {

  *(.vectors*)
  *(.text*)
  *(.rodata)
  _E_Text_ = . ;
}> flash

.data : {

  _S_data_ = . ;
  *(.data)
  _E_data_ = . ;

}> sram AT> flash

.bss : {

  _S_bss_ = . ;
  *(.bss)
  _E_bss_ = . ;

  . = . + 0x1000;
  stack_top = . ;

}> sram
}
```

7.2.3 Startup.c File

```c
// Eng. Ziad Mohamed Elbendary _

#include <stdint.h> // AUTOSAR defined typedef convention.

void reset_handler(void);
void NMI_handler(void) __attribute__((weak, alias ("default_handler")));;
void H_fault_handler(void)__attribute__((weak, alias ("default_handler")));;
void NM_fault_handler(void)__attribute__((weak, alias ("default_handler")));;
void Bus_fault(void)__attribute__((weak, alias ("default_handler")));;
void Usage_fault_handler(void)__attribute__((weak, alias ("default_handler")));;

extern int main(void);
extern uint32_t stack_top;
extern uint32_t _E_Text_ ;
extern uint32_t _S_data_ ;
extern uint32_t _E_data_ ;
extern uint32_t _S_bss_ ;
extern uint32_t _E_bss_ ;

// Aliased Function (Unifiable Function)
void default_handler (){
    reset_handler();
}

uint32_t vectors[]__attribute__((section(".vectors"))) = {

    (uint32_t) &stack_top,
    (uint32_t) &reset_handler,
    (uint32_t) &NMI_handler,
    (uint32_t) &NM_fault_handler,
    (uint32_t) &H_fault_handler,
    (uint32_t) &Bus_fault,
    (uint32_t) &Usage_fault_handler

    };

// Send byte by byte from ROM to RAM for .data section

void reset_handler(void){

uint32_t Data_Size = (uint8_t*)&_E_data_ - (uint8_t*)&_S_data_;
uint8_t* P_dest = (uint8_t*)&_S_data_;
uint8_t* P_src = (uint8_t*)&_E_Text_;
uint32_t i;

for(i = 0; i < Data_Size; i++){

*((uint8_t*)P_dest++) = *((uint8_t*)P_src++);

}

// Initialize the .bss section with zeros
uint32_t bss_size = (uint8_t*)&_E_bss_ - (uint8_t*) &_S_bss_;
P_dest = (uint8_t*)&_S_bss_;

for(i = 0; i<bss_size; i++){

    *((uint8_t*)P_dest++) = (uint8_t)0;

}
    main();
}
```

7.3 Files Analysis

7.3.0. elf File Symbols

```
20000004 B _E_bss_
20000004 D _E_data_
0800083c T _E_Text_
20000004 B _S_bss_
20000000 D _S_data_
0800001c T Alarm_Buzzer_init
20001004 B Alarm_Buzzer_ptr2Fun
20001008 B Alarm_Buzzer_status
080000d0 T Alarm_Indicator_init
20001010 B Alarm_Indicator_ptr2Fun
2000100c B Alarm_Indicator_Status
0800023c T Alarm_Manager_init
20001014 B Alarm_Manager_ptr2Fun
2000101c B Alarm_Manager_State
080001ec T AlarmIdle_State
0800019c T AlarmIndication_State
08000134 T AlarmIndicatorOFF
08000168 T AlarmIndicatorON
08000780 W Bus_fault
08000780 T default_handler
08000334 T Delay
20001028 B flash
08000614 T Flash_Memory_Enqueue
08000518 T Flash_Memory_init
20001020 B Flash_Memory_ptr2Fun
20001024 B Flash_Memory_state
2000103c B flashArr
20000000 D flashptr
08000358 T getPressureVal
08000494 T GPIO_INITIALIZATION
08000780 W H_fault_handler
20001018 B i
080006bc T main
08000780 W NM_fault_handler
08000780 W NMI_handler
080006f0 T Pressure_Sensor_init
20001058 B Pressure_Sensor_ptr2Fun
20001050 B Pressure_sensor_state
20001054 B pval
0800078c T reset_handler
08000370 T Set_Alarm_Buzzer
080003c0 T Set_Alarm_Indicator
08000684 T setup
08000098 T ST_BuzzAlarm
080000b4 T ST_BuzzIdle
08000258 T ST_MonitorPressure
0800070c T ST_ReadingPressureVal
08000300 T ST_Waiting
0800074c T ST_WaitingPressureVal
20001004 B stack_top
08000050 T startAlarmBuzzer
08000074 T stopAlarmBuzzer
08000560 T Storing_State
08000780 W Usage_fault_handler
08000000 T vectors
080005a4 T Waiting_State
```

7.3.1. elf File Sections

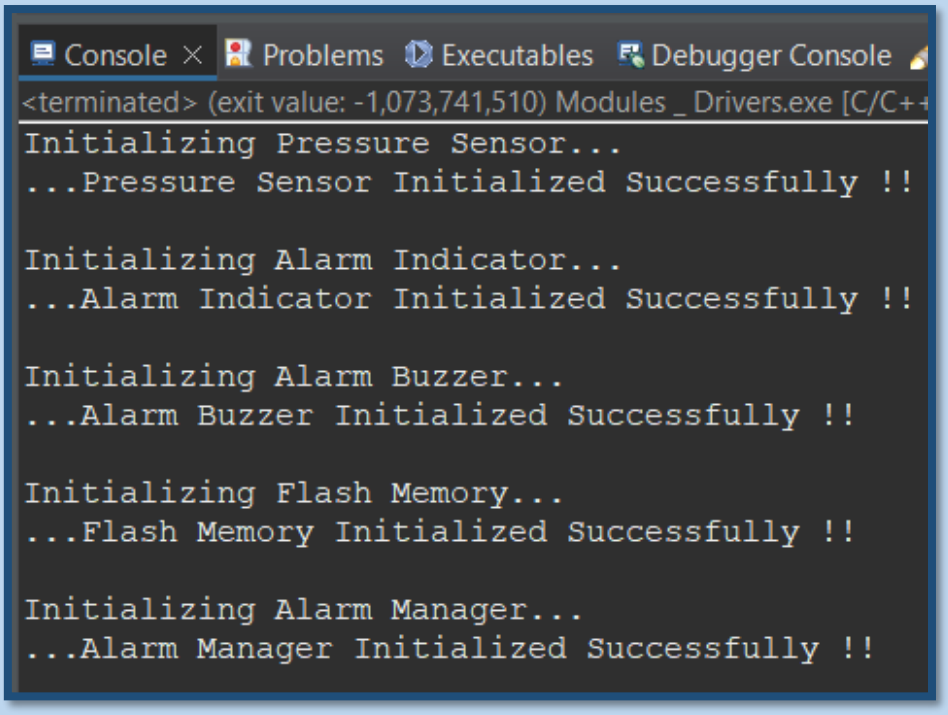
```
Pressure_Detection_System.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000083c  08000000  08000000  00008000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000004  20000000  0800083c  00010000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00001058  20000004  08000840  00010004  2**2
                  ALLOC
  3 .debug_info   00000c1a  00000000  00000000  00010004  2**0
                  CONTENTS, READONLY, DEBUGGING
  4 .debug_abbrev 000005e7  00000000  00000000  00010c1e  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    0000058c  00000000  00000000  00011205  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000100  00000000  00000000  00011791  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_line   000005e2  00000000  00000000  00011891  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .debug_str    00000597  00000000  00000000  00011e73  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000011  00000000  00000000  0001240a  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  0001241b  2**0
                  CONTENTS, READONLY
 11 .debug_frame  000003b0  00000000  00000000  00012450  2**2
                  CONTENTS, READONLY, DEBUGGING
```

8.0 Testing & Simulation

8.1 Software Testing

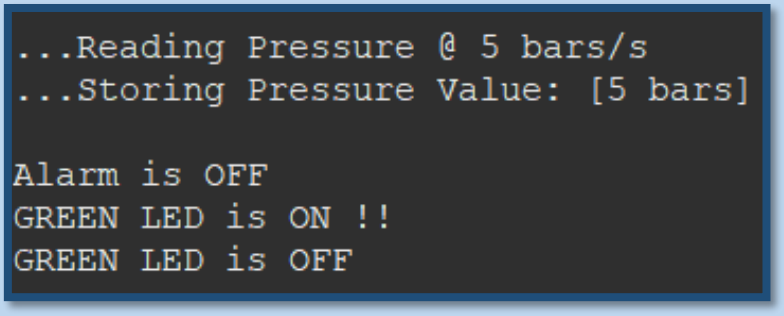
To test the sequence of operation, random values were generated to check how would the system interact when the pressure sensor starts reading. The generated values were set between the ranges of (8 - 21) which will fulfil all the defined conditions. The following figures shows a verification of the software system.

Initializing MCAL, HAL, and System Block Modules



8.1.0 Debugging LED Indicators & Buzzer Alarms Sequence of Operation

1st Case (pval < threshold 1)



2nd Case (threshold 1 <= pval <= threshold 2)

```
...Reading Pressure @ 12 bars/s
...Storing Pressure Value: [12 bars]

YELLOW LED is ON !!

!!! Alarm is Buzzing !!!
Alarm is OFF
!!! Alarm is Buzzing !!!
Alarm is OFF
!!! Alarm is Buzzing !!!
Alarm is OFF
!!! Alarm is Buzzing !!!
Alarm is OFF
!!! Alarm is Buzzing !!!
Alarm is OFF

YELLOW LED is OFF
```

3rd Case (pval > threshold 2)

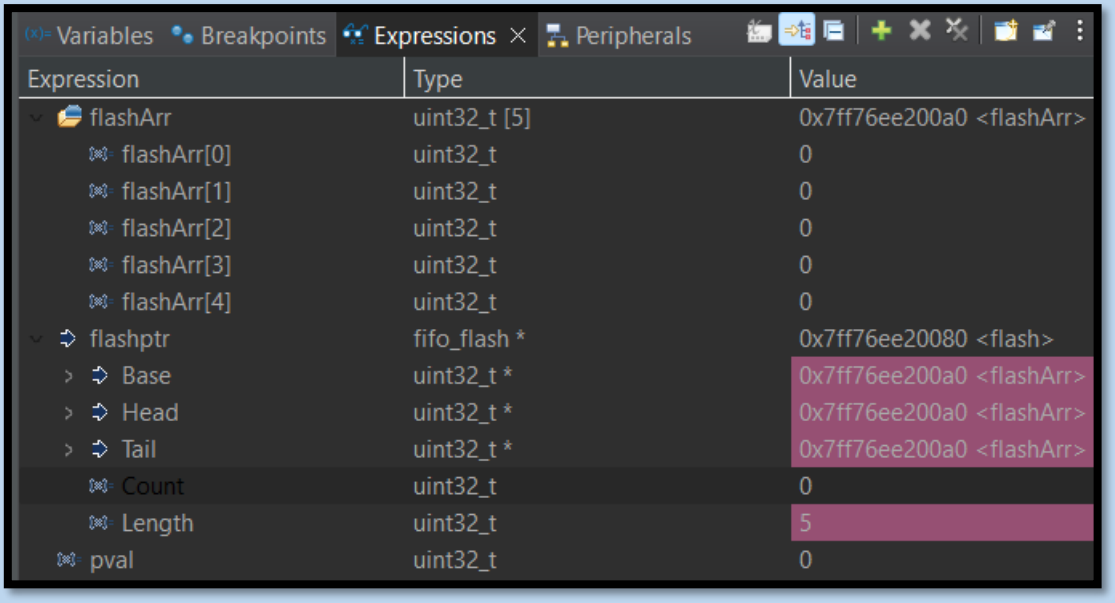
```
...Reading Pressure @ 30 bars/s
...Storing Pressure Value: [30 bars]

!!! Alarm is Buzzing !!!
RED LED is ON !!
Alarm is OFF
RED LED is OFF
```

Flash Memory is Full (After 5 iterations)

```
...Reading Pressure @ 2 bars/s
Warning!! Flash Memory is Full.

Alarm is OFF
GREEN LED is ON !!
GREEN LED is OFF
```

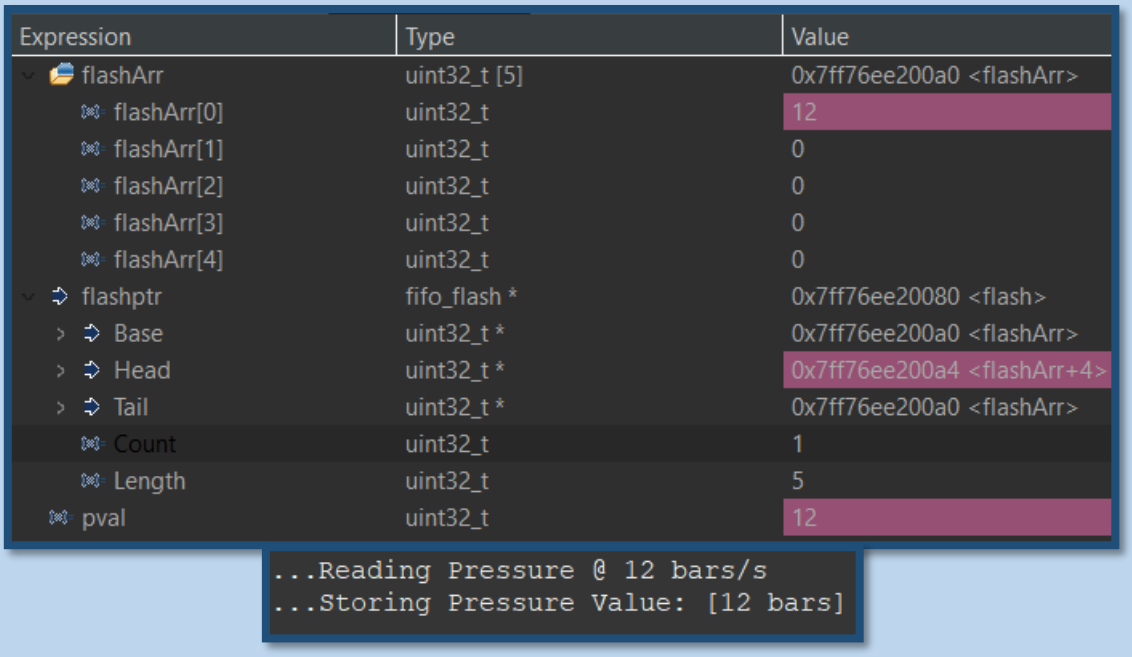
8.1.1 Debugging Flash Memory Storage Sequence of Operation

Initializing the FIFO Structure



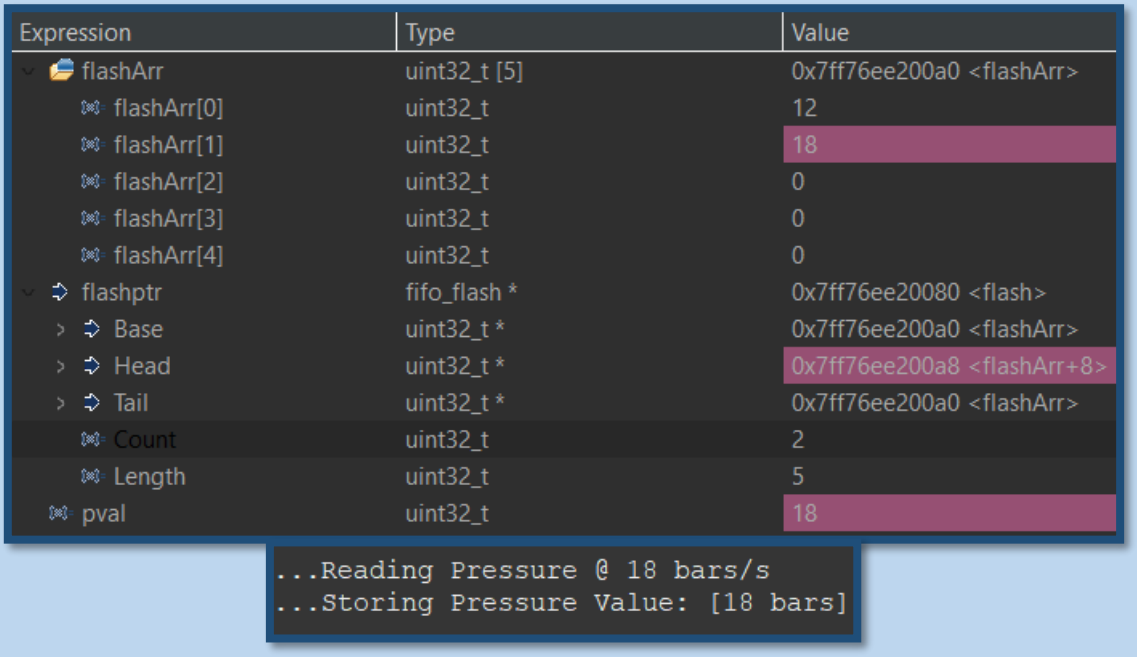
As observed, upon initialization, the base, head, and tail pointers are reserving the address of the first element at the flash memory array. The count variable is 0 as no element has been added yet in the flash memory.

Adding the 1st Element



After the pressure sensor read the 1st value (generated random value) which is stored in pval (pval = 12), pval then gets occupies the first location in the flash memory array as the first element. Consequently, the head pointer will increment by 4 bytes to point to the next location in the flash memory and the count variable will also increment to indicate the number of elements stored in the flash memory. This process will keep on repeating upon generating a new value until the flash memory is full as shown in the following figures.

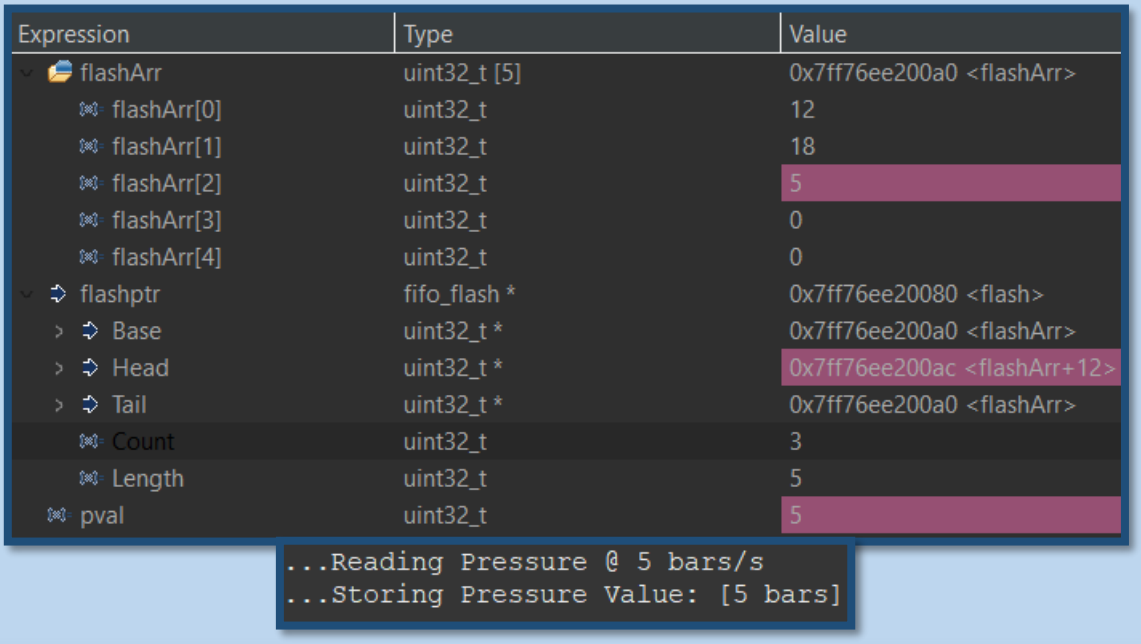
Adding the 2nd Element



Expression	Type	Value
flashArr	uint32_t [5]	0x7ff76ee200a0 <flashArr>
flashArr[0]	uint32_t	12
flashArr[1]	uint32_t	18
flashArr[2]	uint32_t	0
flashArr[3]	uint32_t	0
flashArr[4]	uint32_t	0
flashptr	fifo_flash *	0x7ff76ee20080 <flash>
Base	uint32_t *	0x7ff76ee200a0 <flashArr>
Head	uint32_t *	0x7ff76ee200a8 <flashArr+8>
Tail	uint32_t *	0x7ff76ee200a0 <flashArr>
Count	uint32_t	2
Length	uint32_t	5
pval	uint32_t	18

```
...Reading Pressure @ 18 bars/s
...Storing Pressure Value: [18 bars]
```

Adding the 3rd Element



Expression	Type	Value
flashArr	uint32_t [5]	0x7ff76ee200a0 <flashArr>
flashArr[0]	uint32_t	12
flashArr[1]	uint32_t	18
flashArr[2]	uint32_t	5
flashArr[3]	uint32_t	0
flashArr[4]	uint32_t	0
flashptr	fifo_flash *	0x7ff76ee20080 <flash>
Base	uint32_t *	0x7ff76ee200a0 <flashArr>
Head	uint32_t *	0x7ff76ee200ac <flashArr+12>
Tail	uint32_t *	0x7ff76ee200a0 <flashArr>
Count	uint32_t	3
Length	uint32_t	5
pval	uint32_t	5

```
...Reading Pressure @ 5 bars/s
...Storing Pressure Value: [5 bars]
```

Adding the 4th Element

Expression	Type	Value
flashArr	uint32_t [5]	0x7ff76ee200a0 <flashArr>
flashArr[0]	uint32_t	12
flashArr[1]	uint32_t	18
flashArr[2]	uint32_t	5
flashArr[3]	uint32_t	11
flashArr[4]	uint32_t	0
flashptr	fifo_flash *	0x7ff76ee20080 <flash>
Base	uint32_t *	0x7ff76ee200a0 <flashArr>
Head	uint32_t *	0x7ff76ee200b0 <flashArr+16>
Tail	uint32_t *	0x7ff76ee200a0 <flashArr>
Count	uint32_t	4
Length	uint32_t	5
pval	uint32_t	11

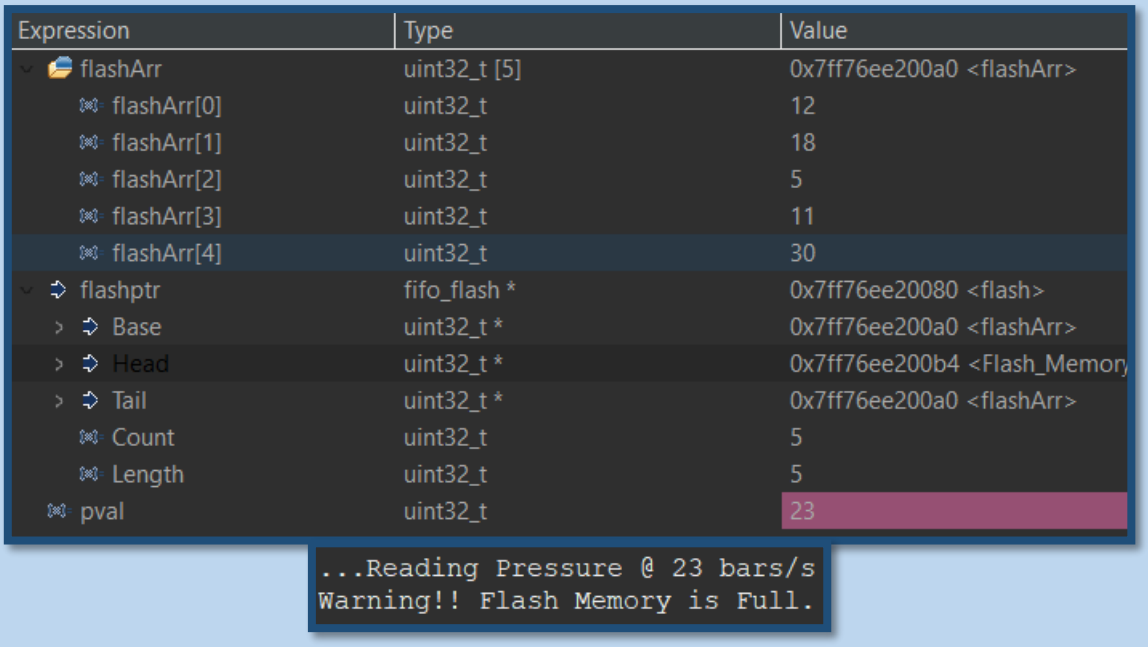
```
...Reading Pressure @ 11 bars/s
...Storing Pressure Value: [11 bars]
```

Adding the Last Element

Expression	Type	Value
flashArr	uint32_t [5]	0x7ff76ee200a0 <flashArr>
flashArr[0]	uint32_t	12
flashArr[1]	uint32_t	18
flashArr[2]	uint32_t	5
flashArr[3]	uint32_t	11
flashArr[4]	uint32_t	30
flashptr	fifo_flash *	0x7ff76ee20080 <flash>
Base	uint32_t *	0x7ff76ee200a0 <flashArr>
Head	uint32_t *	0x7ff76ee200b4 <Flash_Memory_
Tail	uint32_t *	0x7ff76ee200a0 <flashArr>
Count	uint32_t	5
Length	uint32_t	5
pval	uint32_t	30

```
...Reading Pressure @ 30 bars/s
...Storing Pressure Value: [30 bars]
```

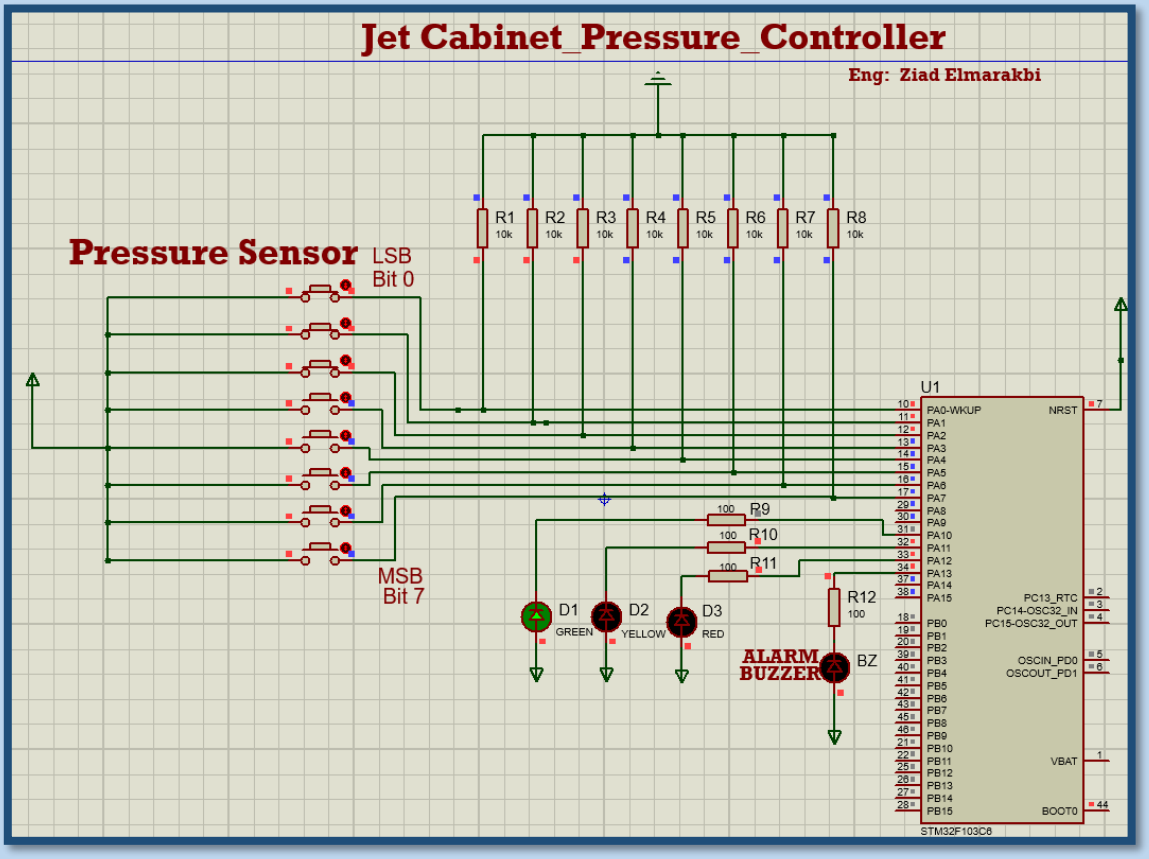
Over Loading Flash Memory



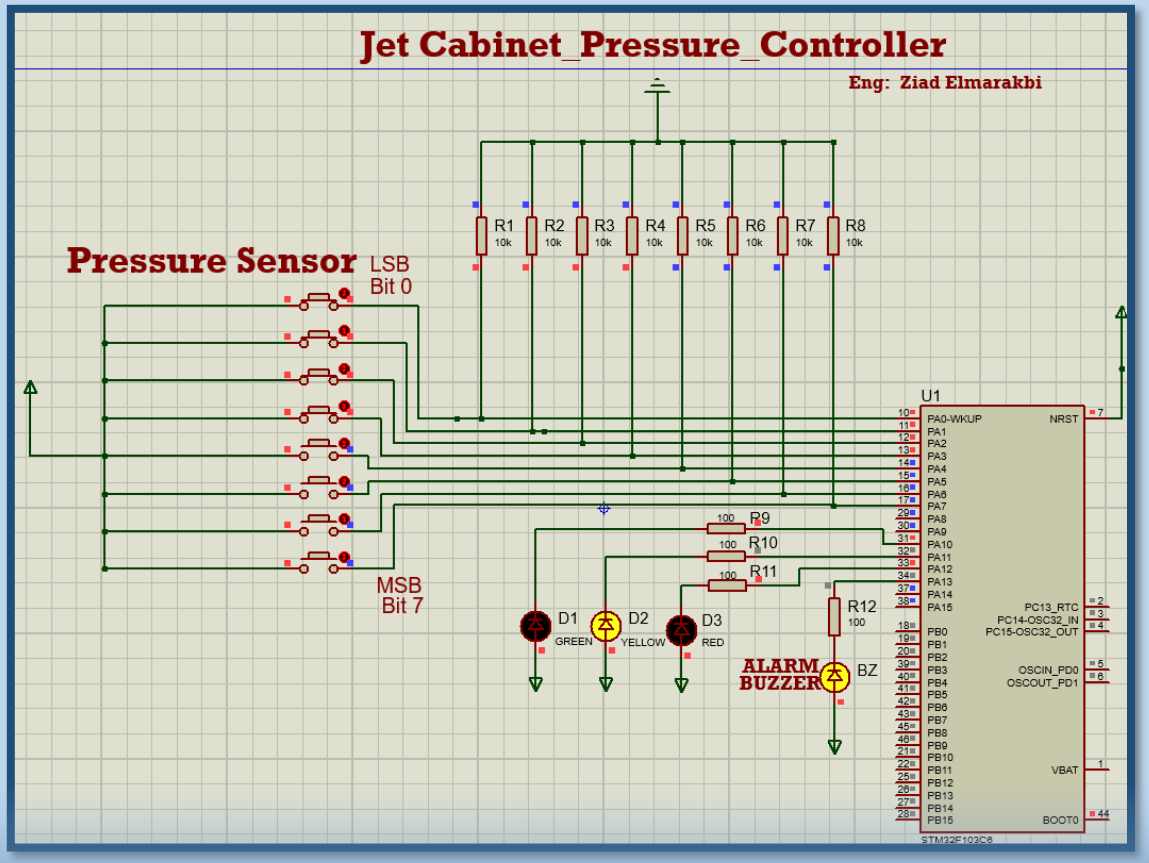
8.2 Hardware Proteus Simulation

Upon completing the software testing using the host machine native toolchain. The Arm Cross toolchain was then utilized on the code to generate a hex or binary file that can be comprehended by the target STM SoC. The pressure sensor was mimicked by 8 switches that read logical states “0 or 1” from the user then store those values in the Input Data Register (IDR) of the SoC. Typically, the IDR in 90% of microcontrollers has a width of 4 bytes. However, only 1 byte or 8 bits were needed to store the values obtained from the switches as 1 byte can hold a decimal value up to 256_{dec}. Recalling that the range of pressure values read by the sensor was conditioned to be between (8_{dec} - 21_{dec}). The switches were connected to Port A layed out pins from 0 – 7. Note that Port A, Port B, …Port X is a subset in one of the General-Purpose Input/Output (GPIO) peripherals within the ARM microcontroller. From the same GPIOA peripheral, the LED indicators and the Alarm buzzer modules were set as outputs which operate according to different input values from the pressure sensor. Those modules are connected to the Output Data Register (ODR) of the GPIOA peripheral. To configure which pins are inputs or outputs, the direction registers CRL and CRH has to be set accordingly. The following figures show the operation of the pressure detection system.

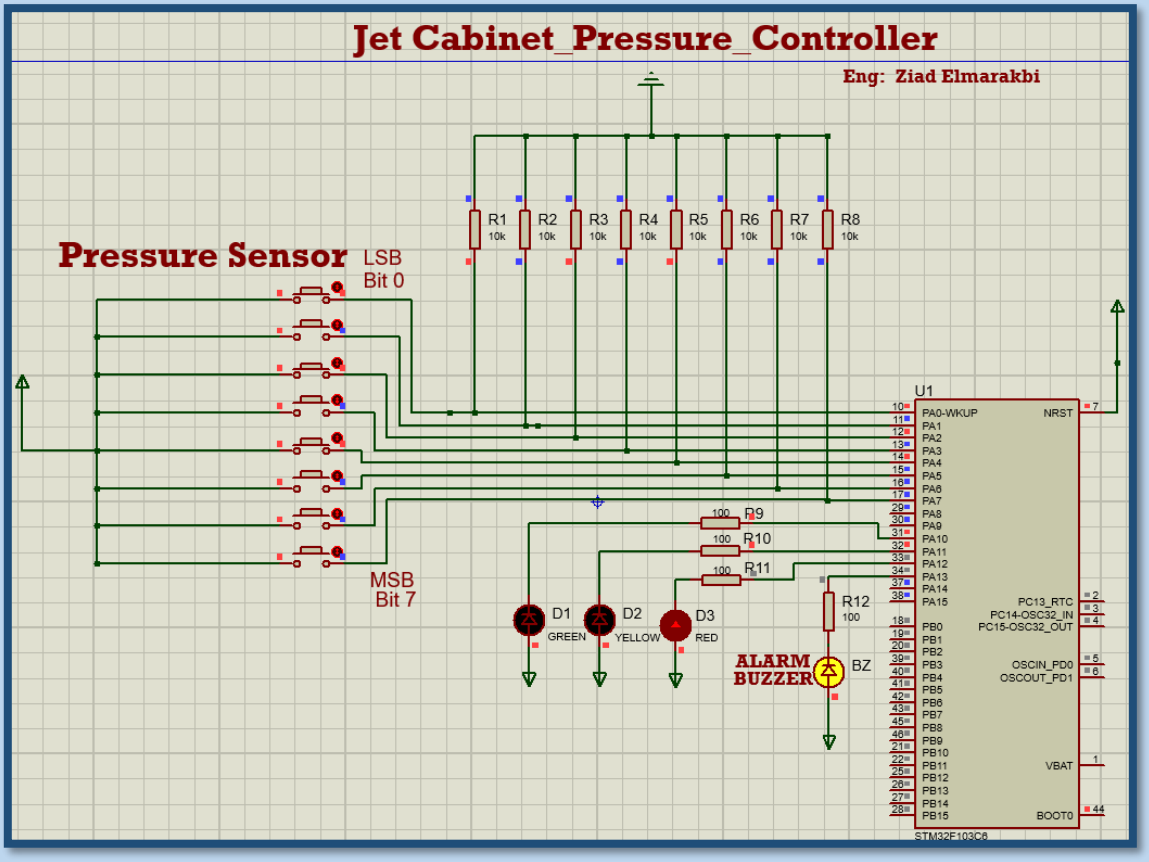
1st Case (pval < threshold 1), **pval = 7.** *Green Led is On, Buzzer is Off*



2nd Case (threshold 1 <= pval <= threshold 2), **pval = 15,** *Yellow Led is On, Buzzer is blinking.*



3rd Case (pval > threshold 2), **pval = 21,** *Red Led is On, Buzzer is On*



9.0 Misra-C Rules Violation Checker

Optimally, in order to check Misra C rules violation for all the files at one runtime, an Automated MakeFile was created.

9.1 Automated MakeFile

```
SRC= $(wildcard *.c)
OBJ= $(SRC:.c=.c.dump)


all: $(OBJ)


%.c.dump: %.c
	./cppcheck --dump $(SRC)
	python addons/misra.py $(OBJ)


clean_all:
	rm *.c.dump
	@echo "--------------------All Dumped Files Removed-----------------------"
```

9.2 Checking and Evaluating Misra C Violations

```
Ziad@FX506HC MINGW64 /d/Embedded SW Engineering/Programs & Tools/cppcheck
$ make
./cppcheck --dump Alarm_Buzzer_Driver.c Alarm_Indicator_Driver.c Alarm_manager.c
 BareMetalDriver.c Flash_Memory_Driver.c Main_Algorithm.c Pressure_Sensor.c Star
tup.c
Checking Alarm_Buzzer_Driver.c ...
1/8 files checked 7% done
Checking Alarm_Indicator_Driver.c ...
2/8 files checked 20% done
Checking Alarm_manager.c ...
3/8 files checked 31% done
Checking BareMetalDriver.c ...
4/8 files checked 44% done
Checking Flash_Memory_Driver.c ...
5/8 files checked 64% done
Checking Main_Algorithm.c ...
6/8 files checked 73% done
Checking Pressure_Sensor.c ...
7/8 files checked 83% done
Checking Startup.c ...
8/8 files checked 100% done
python addons/misra.py Alarm_Buzzer_Driver.c.dump Alarm_Indicator_Driver.c.dump
Alarm_manager.c.dump BareMetalDriver.c.dump Flash_Memory_Driver.c.dump Main_Algo
rithm.c.dump Pressure_Sensor.c.dump Startup.c.dump
[Alarm_Buzzer_Driver.c:26] (style) misra violation (use --rule-texts=<file> to g
et proper output) (Undefined) [misra-c2012-2.7]
[Alarm_Buzzer_Driver.c:31] (style) misra violation (use --rule-texts=<file> to g
et proper output) (Undefined) [misra-c2012-2.7]
[Alarm_Indicator_Driver.c:31] (style) misra violation (use --rule-texts=<file> t
o get proper output) (Undefined) [misra-c2012-15.6]
[Alarm_Indicator_Driver.c:34] (style) misra violation (use --rule-texts=<file> t
o get proper output) (Undefined) [misra-c2012-15.6]
[Alarm_Indicator_Driver.c:37] (style) misra violation (use --rule-texts=<file> t
o get proper output) (Undefined) [misra-c2012-15.6]
[Alarm_Indicator_Driver.c:45] (style) misra violation (use --rule-texts=<file> t
o get proper output) (Undefined) [misra-c2012-15.6]
[Alarm_Indicator_Driver.c:48] (style) misra violation (use --rule-texts=<file> t
o get proper output) (Undefined) [misra-c2012-15.6]
[Alarm_Indicator_Driver.c:51] (style) misra violation (use --rule-texts=<file> t
o get proper output) (Undefined) [misra-c2012-15.6]
[Alarm_Indicator_Driver.c:38] (style) misra violation (use --rule-texts=<file> t
o get proper output) (Undefined) [misra-c2012-15.7]
[Alarm_Indicator_Driver.c:52] (style) misra violation (use --rule-texts=<file> t
o get proper output) (Undefined) [misra-c2012-15.7]
[Alarm_manager.c:16] (style) misra violation (use --rule-texts=<file> to get pro
per output) (Undefined) [misra-c2012-2.7]
[Alarm_manager.c:51] (style) misra violation (use --rule-texts=<file> to get pro
per output) (Undefined) [misra-c2012-2.7]
[BareMetalDriver.c:7] (style) misra violation (use --rule-texts=<file> to get pr
oper output) (Undefined) [misra-c2012-15.6]
[BareMetalDriver.c:22] (style) misra violation (use --rule-texts=<file> to get p
roper output) (Undefined) [misra-c2012-15.7]
[BareMetalDriver.c:50] (style) misra violation (use --rule-texts=<file> to get p
roper output) (Undefined) [misra-c2012-15.7]
[BareMetalDriver.c:7] (style) misra violation (use --rule-texts=<file> to get pr
oper output) (Undefined) [misra-c2012-17.8]
```

```
[Flash_Memory_Driver.c:67] (style) misra violation (use --rule-texts=<file> to g
et proper output) (Undefined) [misra-c2012-12.1]
[Flash_Memory_Driver.c:59] (style) misra violation (use --rule-texts=<file> to g
et proper output) (Undefined) [misra-c2012-15.5]
[Flash_Memory_Driver.c:76] (style) misra violation (use --rule-texts=<file> to g
et proper output) (Undefined) [misra-c2012-15.5]
[Flash_Memory_Driver.c:72] (style) misra violation (use --rule-texts=<file> to g
et proper output) (Undefined) [misra-c2012-15.6]
[Pressure_Sensor.c:19] (style) misra violation (use --rule-texts=<file> to get p
roper output) (Undefined) [misra-c2012-2.7]
[Pressure_Sensor.c:31] (style) misra violation (use --rule-texts=<file> to get p
roper output) (Undefined) [misra-c2012-2.7]
[Startup.c:28] (style) misra violation (use --rule-texts=<file> to get proper ou
tput) (Undefined) [misra-c2012-11.4]
[Startup.c:49] (style) misra violation (use --rule-texts=<file> to get proper ou
tput) (Undefined) [misra-c2012-13.3]
[Startup.c:59] (style) misra violation (use --rule-texts=<file> to get proper ou
tput) (Undefined) [misra-c2012-13.3]
[Startup.c:62] (style) misra violation (use --rule-texts=<file> to get proper ou
tput) (Undefined) [misra-c2012-17.7]
[Startup.c:42] (style) misra violation (use --rule-texts=<file> to get proper ou
tput) (Undefined) [misra-c2012-18.4]
[Startup.c:54] (style) misra violation (use --rule-texts=<file> to get proper ou
tput) (Undefined) [misra-c2012-18.4]
Checking Alarm_Buzzer_Driver.c.dump...
Checking Alarm_Buzzer_Driver.c.dump, config ...
Checking Alarm_Indicator_Driver.c.dump...
Checking Alarm_Indicator_Driver.c.dump, config ...
Checking Alarm_manager.c.dump...
Checking Alarm_manager.c.dump, config ...
Checking BareMetalDriver.c.dump...
Checking BareMetalDriver.c.dump, config ...
Checking Flash_Memory_Driver.c.dump...
Checking Flash_Memory_Driver.c.dump, config ...
Checking Main_Algorithm.c.dump...
Checking Main_Algorithm.c.dump, config ...
Checking Pressure_Sensor.c.dump...
Checking Pressure_Sensor.c.dump, config ...
Checking Startup.c.dump...
Checking Startup.c.dump, config ...

MISRA rules violations found:
        Undefined: 28

MISRA rules violated:
        misra-c2012-2.7 (-): 6
        misra-c2012-11.4 (-): 1
        misra-c2012-12.1 (-): 1
        misra-c2012-13.3 (-): 2
        misra-c2012-15.5 (-): 2
        misra-c2012-15.6 (-): 8
        misra-c2012-15.7 (-): 4
        misra-c2012-17.7 (-): 1
        misra-c2012-17.8 (-): 1
        misra-c2012-18.4 (-): 2
```

Violated Misra C Rules:

R.2.7	Advisory	There should be no unused parameters in functions	1 D	Unused procedure parameter.
			15 D	Unused procedural parameter.
R.11.4	Advisory	A conversion should not be performed between a pointer to object and an integer type	439 S	Cast from pointer to integral type.
			440 S	Cast from integral type to pointer.
R.12.1	Advisory	The precedence of operators within expressions should be made explicit	49 S	Logical conjunctions need brackets.
			361 S	Expression needs brackets.
R.13.3	Advisory	A full expression containing an increment (++) or decrement (--) operator should have no other potential side effects other than that caused by the increment or decrement operator	30 S	Deprecated usage of ++ or -- operators found.
R.15.5	Advisory	A function should have a single point of exit at the end	7 C	Procedure has more than one exit point.
R.15.6	Required	The body of an iteration-statement or a selection-statement shall be a compound statement	11 S	No brackets to loop body (added by Testbed).
			12 S	No brackets to then/else (added by Testbed).
			428 S	No {} for switch (added by Testbed).
R.15.7	Required	All if . . else if constructs shall be terminated with an else statement	59 S	Else alternative missing in if.
			477 S	Empty else clause following else if.
R.17.7	Required	The value returned by a function having non-void return type shall be used	91 D	Function return value potentially unused.
			382 S	(void) missing for discarded return value.
R.17.8	Advisory	A function parameter should not be modified	14 D	Attempt to change parameter passed by value.
			149 S	Reference parameter to procedure is reassigned.
R.18.4	Advisory	The +, -, += and -= operators should not be applied to an expression of pointer type	87 S	Use of pointer arithmetic.
			567 S	Pointer arithmetic is not on array.

Clearing Out 89.29 % of the Misra C Violations

The following figures show the adjustments made on the defective lines of code in all the files to optimally clear out Misra C violations as much as possible. The captured samples of code on the left-hand-side show the original code before adjustments and the right-hand-side show them after adjustments.

Violations 2.7:

```
Define_State(BuzzAlarm){
    Alarm_Buzzer_status   = BuzzAlarm;
    Set_Alarm_Buzzer(0);
}
```

```
void BuzzAlarm_State(){
    Alarm_Buzzer_status   = BuzzAlarm;
    Set_Alarm_Buzzer(0);
}
```

Violations 15.6 & 15.7:

```
void AlarmIndication_State(Indicator_status LEDstatus){

    Alarm_Indicator_Status = IndicatorAlarm;

    if(LEDstatus == Green_On)
        Set_Alarm_Indicator(Green_On);

    else if(LEDstatus == Red_On)
        Set_Alarm_Indicator(Red_On);

    else if(LEDstatus == Yellow_On)
        Set_Alarm_Indicator(Yellow_On);
}
```

```
void AlarmIndication_State(Indicator_status LEDstatus){

    Alarm_Indicator_Status = IndicatorAlarm;
    if(LEDstatus == Green_On){
        Set_Alarm_Indicator(Green_On);
    }
    else if(LEDstatus == Red_On){
        Set_Alarm_Indicator(Red_On);
    }
    else {
        Set_Alarm_Indicator(Yellow_On);
    }
}
```

Violations 15.5 & 12.1:

```
        else                                    else{
            flashptr->Head++;                       flashptr->Head++;
                                                    }
        // DPRINTF("...Storing                  // DPRINTF("...Storing
        return Flash_No_error;              }
    }                                           return Flash_No_error;
}                                           }
```

Violation 17.8:

```
void Delay(int nCount)                  void Delay(int nCount)
{                                       {
    for(; nCount != 0; nCount--);           int DELAY = nCount;
                                            for(DELAY; DELAY != 0; DELAY--){}
}                                       }
```

Violation 11.4:

```
extern uint32_t stack_top; extern uint32_t &stack_top;
extern uint32_t _E_Text_ ; extern uint32_t &_E_Text_ ;
extern uint32_t _S_data_ ; extern uint32_t &_S_data_ ;
extern uint32_t _E_data_ ; extern uint32_t &_E_data_ ;
extern uint32_t _S_bss_ ;  extern uint32_t &_S_bss_ ;
extern uint32_t _E_bss_ ;  extern uint32_t &_E_bss_ ;
```

Violation 13.3:

```
for(i = 0; i < Data_Size; i++){             for(i = 0; i < Data_Size; i++){

*((uint8_t*)P_dest++) = *((uint8_t*)P_src++); *((uint8_t*)P_dest) = *((uint8_t*)P_src);
                                                P_dest++;
}                                               P_src++;
                                            }
```

Violation 17.7:

```
extern int main(void); extern void main(void);
}                      }
    main();                main(void);
}                      }
```

Re-Evaluating Violations:

```
ziad@FX506HC MINGW64 /d/Embedded SW Engineering/Programs & Tools/cppcheck
$ make clean_all
rm *.c.dump
--------------------All Dumped Files Removed----------------------

ziad@FX506HC MINGW64 /d/Embedded SW Engineering/Programs & Tools/cppcheck
$ make
./cppcheck --dump Alarm_Buzzer_Driver.c Alarm_Indicator_Driver.c Alarm_manager.c BareMetalDriver.c Flash_Memo
ry_Driver.c Main_Algorithm.c Pressure_Sensor.c Startup.c
Checking Alarm_Buzzer_Driver.c ...
1/8 files checked 7% done
Checking Alarm_Indicator_Driver.c ...
2/8 files checked 20% done
Checking Alarm_manager.c ...
3/8 files checked 31% done
Checking BareMetalDriver.c ...
4/8 files checked 43% done
Checking Flash_Memory_Driver.c ...
5/8 files checked 63% done
Checking Main_Algorithm.c ...
6/8 files checked 73% done
Checking Pressure_Sensor.c ...
7/8 files checked 83% done
Checking Startup.c ...
8/8 files checked 100% done
python addons/misra.py Alarm_Buzzer_Driver.c.dump Alarm_Indicator_Driver.c.dump Alarm_manager.c.dump BareMeta
lDriver.c.dump Flash_Memory_Driver.c.dump Main_Algorithm.c.dump Pressure_Sensor.c.dump Startup.c.dump
[Flash_Memory_Driver.c:59] (style) misra violation (use --rule-texts=<file> to get proper output) (Undefined)
 [misra-c2012-15.5]
[Startup.c:42] (style) misra violation (use --rule-texts=<file> to get proper output) (Undefined) [misra-c201
2-18.4]
[Startup.c:56] (style) misra violation (use --rule-texts=<file> to get proper output) (Undefined) [misra-c201
2-18.4]
Checking Alarm_Buzzer_Driver.c.dump...
Checking Alarm_Buzzer_Driver.c.dump, config ...
Checking Alarm_Indicator_Driver.c.dump...
Checking Alarm_Indicator_Driver.c.dump, config ...
Checking Alarm_manager.c.dump...
Checking Alarm_manager.c.dump, config ...
Checking BareMetalDriver.c.dump...
Checking BareMetalDriver.c.dump, config ...
Checking Flash_Memory_Driver.c.dump...
Checking Flash_Memory_Driver.c.dump, config ...
Checking Main_Algorithm.c.dump...
Checking Main_Algorithm.c.dump, config ...
Checking Pressure_Sensor.c.dump...
Checking Pressure_Sensor.c.dump, config ...
Checking Startup.c.dump...
Checking Startup.c.dump, config ...

MISRA rules violations found:
        Undefined: 3

MISRA rules violated:
        misra-c2012-15.5 (-): 1
        misra-c2012-18.4 (-): 2
```

Percentage of Violations Cleared: **((28 - 3 / 28) × 100%) ~ 89.29 %**