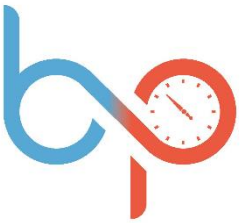
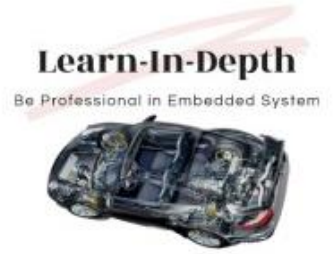




SafePressure Alert

Mastering Embedded System Online Diploma
www.learn-in-depth.com

Implementation Document

First Term (Final Project 1)

Eng. Abdallah Shabaan Ghazy
email:abdallah.shabaan.ghazy@gmail.com

My Profile:

https://www.learn-in-depth-store.com/certificate/abdallah.shabaan.ghazy%40gmail.com

Implementation for Pressure Control System (PCS)

Version 1.0 approved

Prepared by: Abdallah Shabaan Ghazy

Organization: learn in depth

Date Created: 8/13/2024

Implementation Document

1. Introduction

- **Objective:** This document describes the implementation of the Alarm Management System for pressure control. It aims to detail how the system was implemented and the specifics of each part to ensure compliance with the requirements.

- **Overview:** The system is designed to monitor pressure levels and alert users if thresholds are exceeded. It consists of multiple components that interact to ensure efficiency and accuracy.

- **Constraints:** Assumptions and constraints considered during implementation.

2. System Description

- **System Overview:** The system uses pressure sensors to read values and a control system to alert users based on these readings.

- **Key Components:**
  - Pressure sensors
  - Control unit
  - User interface

- **Technologies Used:**
  - Programming Language: C
  - Development Environment: GCC

3. Implementation Requirements

- **System Requirements:**
  - Read values from pressure sensors
  - Alert the user when thresholds are exceeded
  - Display values on the user interface

- **Performance Criteria:**
  - System response time less than 1 second
  - Pressure reading accuracy of 0.1 bar

- **Configuration Details:**
  - The system is configured to read pressure values every 0.5 seconds and update the user interface periodically.

4. System Design

- **Overall Structure:**
  - The system includes a control unit and a set of connected pressure sensors.

- **Module Design:**
  - Control Unit: Implements core logic and manages communications.
  - Pressure Sensors: Measure pressure and send data to the control unit.

- **Interfaces:**
  - Interface between the control unit and pressure sensors
  - Interface between the control unit and the user interface

5. System Implementation

- **Implementation Details:**
  - The code was developed in C and all modules were successfully integrated.

- **Procedures:**
  - Code development
  - Module testing
  - System integration

- **Issues and Solutions:**
  - A problem with sensor data reading was discovered and resolved by adjusting the code.

6. System Testing

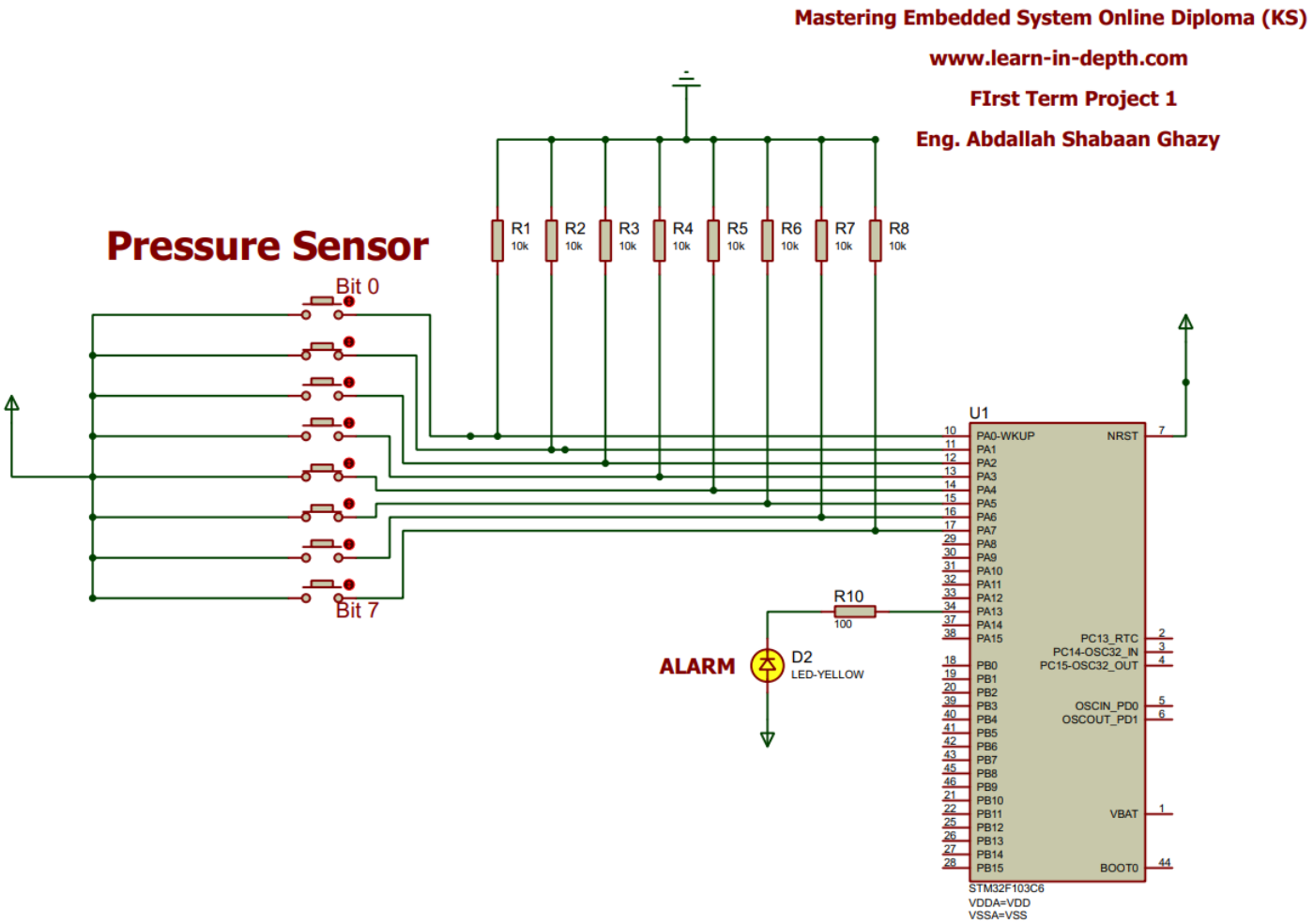
- **Test Plans:**
  - Test reading values from pressure sensors
  - Test the alert system

- **Test Results:**
  - All tests passed successfully, with minor improvements to system performance.

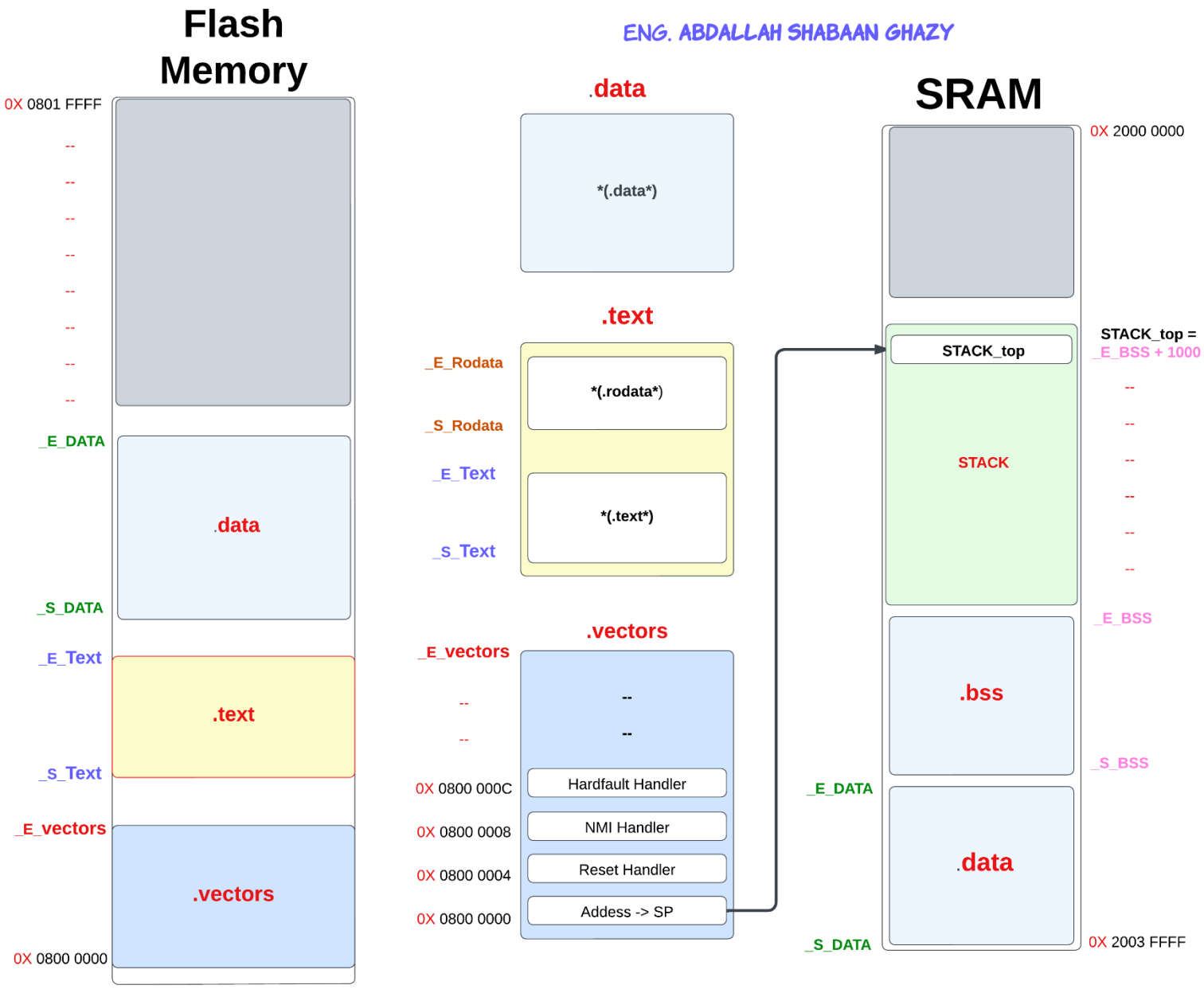
- **Issues and Tests:**
  - A slight delay in system response was detected and improved.

7. Appendices

- **Diagrams:**



Memory Map Description




```
Pressure_Controller_CortexM3.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .vectors      0000003c  08000000  08000000  00008000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  1 .text         0000046c  0800003c  0800003c  0000803c  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .data         0000000c  20000000  080004a8  00010000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  3 .bss          00001028  2000000c  080004b4  0001000c  2**2
                  ALLOC
```

```
16
17  Name             Origin             Length             Attributes
18  flash            0x08000000         0x00008000         xr
19  sram             0x20000000         0x00005000         xrw
20  *default*        0x00000000         0xffffffff
21
22  Linker script and memory map
23
24
25  .vectors         0x08000000       0x3c
26   *(.vectors*)
27   .vectors        0x08000000       0x3c startup.o
28                   0x08000000                _init
29                   0x0800003c                _E_vectors = .
30
31  .text            0x0800003c      0x46c
32   *(.text*)
33   .text           0x0800003c       0x70 AlarmActyatorDriver.o
34                   0x0800003c                StartAlarm
35                   0x0800005c                StopeAlarm
36                   0x0800007c                STF_AAD_init
37                   0x08000094                STF_AAD_waiting
38   .text           0x080000ac       0x98 AlarmMonitor.o
39                   0x080000ac                HighPressureDetect
40                   0x080000c8                STF_AlarmOFF
41                   0x080000e0                STF_AlarmON
42                   0x0800010c                STF_waiting
43   .text           0x08000144       0x98 main.o
44                   0x08000144                setup
45                   0x0800019c                main
46   .text           0x080001dc       0x6c MainAlg.o
47                   0x080001dc                SetPressureValue
48                   0x08000230                STF_MA_HighPressureDetect
49                   0x0800023c                STF_MA_LowPressureDetect
50   .text           0x08000248       0x98 PressureSensorDriver.o
51                   0x08000248                STF_PSD_init
52                   0x08000260                STF_PSD_reading
53                   0x080002a8                STF_PSD_waiting
54   .text           0x080002e0       0xbc startup.o
55                   0x080002e0                DebugMon_Handler
56                   0x080002e0                SysTick_Handler
57                   0x080002e0                PendSV_Handler
58                   0x080002e0                UsageFault_Handler
59                   0x080002e0                NMIHandler
60                   0x080002e0                Default_Handler
61                   0x080002e0                MemManage_Handler
62                   0x080002e0                SVC_Handler
63                   0x080002e0                HardfaultHandler
64                   0x080002e0                BusFault_Handler
65                   0x080002ec                ResetHandler
66   .text           0x0800039c      0x10c driver\driver.o
67                   0x0800039c                Delay
68                   0x080003c0                getPressureVal
69                   0x080003d8                Set_Alarm_actuator
70                   0x08000428                GPIO_INITIALIZATION
71   *(.rodata*)
72                   0x080004a8                _E_text = .
73
```

Flash memory

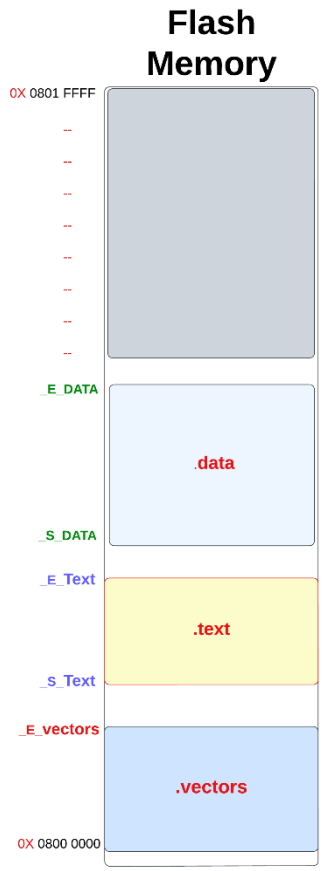
```
.data           0x20000000        0xc load address 0x080004a8
                0x20000000                _S_DATA = .
 *(.data*)
 .data          0x20000000        0x0 AlarmActyatorDriver.o
 .data          0x20000000        0x4 AlarmMonitor.o
                0x20000000                AlarmTime
 .data          0x20000004        0x0 main.o
 .data          0x20000004        0x4 MainAlg.o
                0x20000004                MA_threshold
 .data          0x20000008        0x4 PressureSensorDriver.o
                0x20000008                PSD_PrSensorTime
 .data          0x2000000c        0x0 startup.o
 .data          0x2000000c        0x0 driver\driver.o
                0x2000000c                . = ALIGN (0x4)
                0x2000000c              _E_DATA = .

.igot.plt       0x2000000c        0x0 load address 0x080004b4
 .igot.plt      0x00000000        0x0 AlarmActyatorDriver.o

.bss            0x2000000c     0x1028 load address 0x080004b4
                0x2000000c              _S_bss = .
 *(.bss*)
 .bss           0x2000000c        0x0 AlarmActyatorDriver.o
 .bss           0x2000000c        0x4 AlarmMonitor.o
                0x2000000c                AlarmPeriod
 .bss           0x20000010        0x0 main.o
 .bss           0x20000010        0x4 MainAlg.o
                0x20000010                MA_PressureVal
 .bss           0x20000014        0x0 PressureSensorDriver.o
 .bss           0x20000014        0x0 startup.o
 .bss           0x20000014        0x0 driver\driver.o
                0x20000014                . = ALIGN (0x4)
                0x20000014              _E_bss = .
                0x20001014                . = (. + 0x1000)
 *fill*         0x20000014     0x1000
                0x20001014              _stack_top = .
```

SRAM

Section	Start Address	End Address	Load Address	Description
.data	0x20000000	0x2000000c	0x080004a8	Contains initialized data loaded from flash to SRAM. Includes `_S_DATA` and `_E_DATA` symbols.
.bss	0x2000000c	0x20001014	0x080004b4	Contains uninitialized data initialized to zero at startup. Includes `_S_bss`, `_E_bss`, and `_stack_top` symbols.
Fill	0x20001014	0x20002014	-	Fills the remaining space after the `.bss` section to ensure no uninitialized data remains.
Stack	0x20001014	-	-	The stack starts at `_stack_top`, which is the end of the `.bss` section plus 0x1000 bytes of reserved space.

Flash Memory Layout

Description: The Flash memory is divided into several sections that store different types of data. Below is a description of each section and its contents:



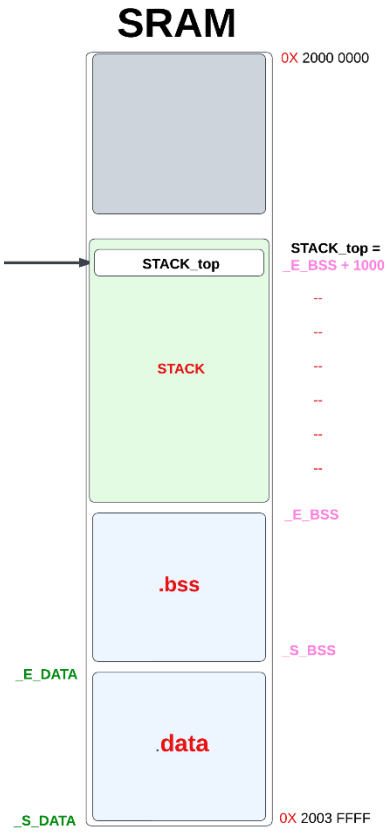
1. **Section .vectors**
   - **Address:** Starts at the beginning of Flash memory.
   - **Contents:**
     - **SP (Stack Pointer):** The initial stack pointer address.
     - **Reset Handler:** The address of the reset handler, which is the entry point of the program.
     - **Vector Table:** Contains pointers to various interrupt handlers.

2. **Section .text**
   - **Address:** Follows the .vectors section.
   - **Contents:**
     - **.text**: Contains the executable code of the application.
     - **.rodata**: Contains read-only data such as constants and static strings.

3. **Section .data**
   - **Address:** Located after the .text section.
   - **Contents:**
     - **.data**: Initialized global and static variables. This section is copied from Flash to SRAM during startup.

SRAM Initialization and Layout

Description: After the microcontroller starts, the following initializations occur in SRAM:

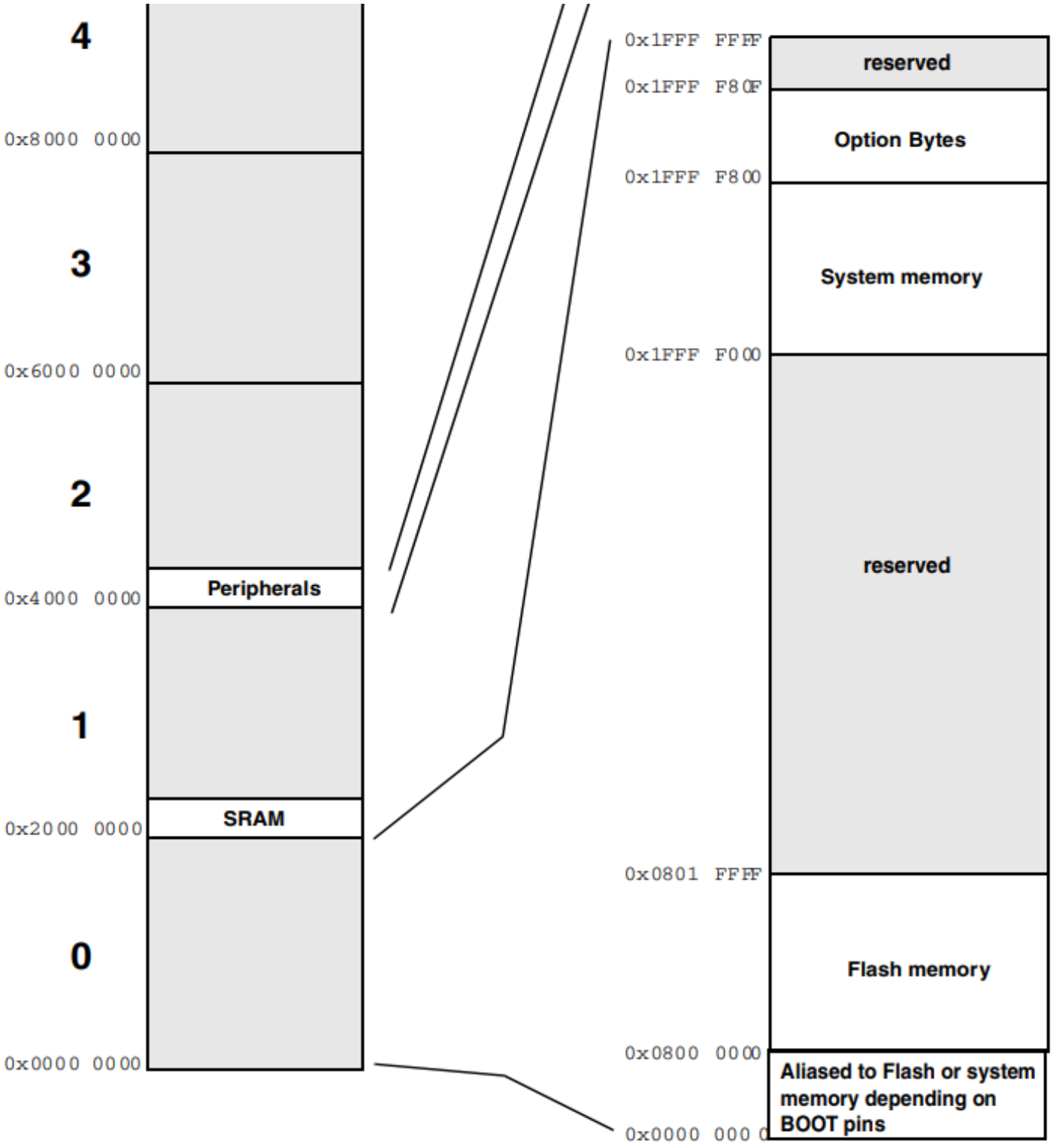


1. **Relocation of .data Section**
   - **From Address:** _S_DATA
   - **To Address:** _E_DATA
   - **Purpose:** Initializes .data section in SRAM with values from Flash.

2. **Initialization of .bss Section**
   - **From Address:** _S_BSS
   - **To Address:** _E_BSS
   - **Purpose:** Clears and initializes .bss section in SRAM to zero.

3. **Stack Initialization**
   - **Address:** STACK_top
   - **Calculated as:** _E_BSS + 1000 (or another size based on your stack requirements)
   - **Purpose:** Sets up the stack pointer.

Memory mapping



Vector table for other STM32F10xxx devices

Position	Priority	Type of priority	Acronym	Description	Address
-	-	-	-	Reserved	0x0000_0000
-	-3	fixed	Reset	Reset	0x0000_0004
-	-2	fixed	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000_0008
-	-1	fixed	HardFault	All class of fault	0x0000_000C
-	0	settable	MemManage	Memory management	0x0000_0010
-	1	settable	BusFault	Prefetch fault, memory access fault	0x0000_0014
-	2	settable	UsageFault	Undefined instruction or illegal state	0x0000_0018
-	-	-	-	Reserved	0x0000_001C - 0x0000_002B
-	3	settable	SVCall	System service call via SWI instruction	0x0000_002C
-	4	settable	Debug Monitor	Debug Monitor	0x0000_0030
-	-	-	-	Reserved	0x0000_0034
-	5	settable	PendSV	Pendable request for system service	0x0000_0038
-	6	settable	SysTick	System tick timer	0x0000_003C
0	7	settable	WWDG	Window watchdog interrupt	0x0000_0040
1	8	settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044
2	9	settable	TAMPER	Tamper interrupt	0x0000_0048
3	10	settable	RTC	RTC global interrupt	0x0000_004C
4	11	settable	FLASH	Flash global interrupt	0x0000_0050
5	12	settable	RCC	RCC global interrupt	0x0000_0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000_0068
11	18	settable	DMA1_Channel1	DMA1 Channel1 global interrupt	0x0000_006C
12	19	settable	DMA1_Channel2	DMA1 Channel2 global interrupt	0x0000_0070
13	20	settable	DMA1_Channel3	DMA1 Channel3 global interrupt	0x0000_0074

The Vector tables

Symbols Table

```
Abdallah Ghazy@DESKTOP-3DICVQM MINGW32 /f/Active courses/New folder/First Tearm/
FIRST TERM Project 1/FIRST_TERM_project1/implementation/code
$ arm-none-eabi-nm.exe Pressure_Controller_CortexM3.elf
20000014 B _E_bss
2000000c D _E_DATA
080004a8 T _E_text
0800003c D _E_vectors
08000000 D _init
2000000c B _S_bss
20000000 D _S_DATA
20001014 B _stack_top
20001018 B AAD_State
20001014 B AADStateID
2000000c B AlarmPeriod
20000000 D AlarmTime
20001020 B AM_State
2000101c B AMStateID
080002e0 W BusFault_Handler
080002e0 W DebugMon_Handler
080002e0 T Default_Handler
0800039c T Delay
080003c0 T getPressureVal
08000428 T GPIO_INITIALIZATION
080002e0 W HardfaultHandler
080000ac T HighPressureDetect
20000010 B MA_PressureVal
20001028 B MA_state
20000004 D MA_threshold
0800019c T main
20001024 B MAStateID
080002e0 W MemManage_Handler
080002e0 W NMIHandler
080002e0 W PendSV_Handler
20001030 B PSD_PressureVal
20000008 D PSD_PrSensorTime
2000102c B PSD_state
20001025 B PSDStateID
080002ec T ResetHandler
080003d8 T Set_Alarm_actuator
080001dc T SetPressureValue
08000144 T setup
0800003c T StartAlarm
0800007c T STF_AAD_init
08000094 T STF_AAD_waiting
080000c8 T STF_AlarmOFF
080000e0 T STF_AlarmON
08000230 T STF_MA_HighPressureDetect
0800023c T STF_MA_LowPressureDetect
08000248 T STF_PSD_init
08000260 T STF_PSD_reading
080002a8 T STF_PSD_waiting
0800010c T STF_waiting
0800005c T StopeAlarm
080002e0 W SVC_Handler
080002e0 W SysTick_Handler
080002e0 W UsageFault_Handler
```

Symbol	Address	Type
`_E_bss`	0x20000014	B (BSS end)
`_E_DATA`	0x2000000c	D (Data end)
`_E_text`	0x080004a8	T (Text end)
`_E_vectors`	0x0800003c	D (Vectors end)
`_init`	0x08000000	D (Data)
`_S_bss`	0x2000000c	B (BSS start)
`_S_DATA`	0x20000000	D (Data start)
`_stack_top`	0x20001014	B (Stack top)
`AAD_State`	0x20001018	B (BSS)
`AADStateID`	0x20001014	B (BSS)
`AlarmPeriod`	0x2000000c	B (BSS)
`AlarmTime`	0x20000000	D (Data)
`AM_State`	0x20001020	B (BSS)
`AMStateID`	0x2000101c	B (BSS)
`BusFault_Handler`	0x080002e0	W (Weak)
`DebugMon_Handler`	0x080002e0	W (Weak)
`Default_Handler`	0x080002e0	T (Text)
`Delay`	0x0800039c	T (Text)
`getPressureVal`	0x080003c0	T (Text)
`GPIO_INITIALIZATION`	0x08000428	T (Text)
`HardfaultHandler`	0x080002e0	W (Weak)
`HighPressureDetect`	0x080000ac	T (Text)
`MA_PressureVal`	0x20000010	B (BSS)
`MA_state`	0x20001028	B (BSS)
`MA_threshold`	0x20000004	D (Data)
`main`	0x0800019c	T (Text)
`MAStateID`	0x20001024	B (BSS)
`PSD_PressureVal`	0x20001030	B (BSS)
`PSD_PrSensorTime`	0x20000008	D (Data)
`PSD_state`	0x2000102c	B (BSS)
`PSDStateID`	0x20001025	B (BSS)
`ResetHandler`	0x080002ec	T (Text)
`Set_Alarm_actuator`	0x080003d8	T (Text)
`SetPressureValue`	0x080001dc	T (Text)
`setup`	0x08000144	T (Text)
`StartAlarm`	0x0800003c	T (Text)

Startup Code

Code Description

This code is a basic startup code for an embedded system. It includes setting up initial memory configuration, defining interrupt handlers, and copying data sections from Flash to SRAM. Below is a description of the code components and a table summarizing the details.

Table Summary

Component	Description
External Variables	
`_E_text`	End address of the `.text` section in memory.
`_S_DATA`	Start address of the `.data` section in memory.
`_E_DATA`	End address of the `.data` section in memory.
`_S_bss`	Start address of the `.bss` section in memory.
`_E_bss`	End address of the `.bss` section in memory.
`_stack_top`	Top address of the stack.
Function Definitions	
`ResetHandler()`	Function that initializes memory sections and calls `main()`.
`Default_Handler()`	Default interrupt handler called if no specific handler is defined.
`NMIHandler()`, `HardfaultHandler()`, ...	Weakly defined interrupt handlers defaulting to `Default_Handler`.
Vector Table	
`_init[]`	Array of addresses for interrupt handlers and initial stack pointer in the `.vectors` section.
`ResetHandler` Function	
Copy `.data` Section	Copies initialized data from Flash to SRAM.
Initialize `.bss` Section	Clears `.bss` section in SRAM.
Call `main()`	Starts the main application after initialization.

```c
#include <stdint.h>

extern unsigned int _E_text;
extern unsigned int _S_DATA;
extern unsigned int _E_DATA;
extern unsigned int _S_bss;
extern unsigned int _E_bss;
extern unsigned int _stack_top;

extern int main();
void ResetHandler(void);
void Default_Handler(void);

void Default_Handler(void)
{
    ResetHandler();
}

// Weak aliases for interrupt handlers
void NMIHandler(void) __attribute__((weak, alias("Default_Handler")));
void HardfaultHandler(void) __attribute__((weak, alias("Default_Handler")));
void MemManage_Handler(void) __attribute__((weak, alias("Default_Handler")));
void BusFault_Handler(void) __attribute__((weak, alias("Default_Handler")));
void UsageFault_Handler(void) __attribute__((weak, alias("Default_Handler")));
void SVC_Handler(void) __attribute__((weak, alias("Default_Handler")));
void DebugMon_Handler(void) __attribute__((weak, alias("Default_Handler")));
void PendSV_Handler(void) __attribute__((weak, alias("Default_Handler")));
void SysTick_Handler(void) __attribute__((weak, alias("Default_Handler")));

uint32_t _init[] __attribute__((section(".vectors"))) = {
    (uint32_t) &_stack_top,            // Initial stack pointer
    (uint32_t) &ResetHandler,          // Reset Handler
    (uint32_t) &NMIHandler,            // NMI Handler
    (uint32_t) &HardfaultHandler,      // HardFault Handler
    (uint32_t) &MemManage_Handler,     // MemManage Handler
    (uint32_t) &BusFault_Handler,      // BusFault Handler
    (uint32_t) &UsageFault_Handler,    // UsageFault Handler
    (uint32_t) 0,                      // Reserved
    (uint32_t) 0,                      // Reserved
    (uint32_t) 0,                      // Reserved
    (uint32_t) 0,                      // Reserved
    (uint32_t) &SVC_Handler,           // SVC Handler
    (uint32_t) &DebugMon_Handler,      // DebugMon Handler
    (uint32_t) &PendSV_Handler,        // PendSV Handler
    (uint32_t) &SysTick_Handler        // SysTick Handler
};

void ResetHandler(void)
{
    unsigned int DATA_size = (unsigned char*)&_E_DATA - (unsigned char*)&_S_DATA;
    unsigned char* P_src = (unsigned char*)&_E_text;
    unsigned char* P_dst = (unsigned char*)&_S_DATA;

    for (unsigned int i = 0; i < DATA_size; i++) {
        *((uint8_t*)P_dst++) = *((uint8_t*)P_src++);
    }

    unsigned int bss_size = (unsigned char*)&_E_bss - (unsigned char*)&_S_bss;
    P_dst = (unsigned char*)&_S_bss;

    for (unsigned int i = 0; i < bss_size; i++) {
        *((uint8_t*)P_dst++) = (uint8_t)0;
    }

    main();
}
```

Linker Script Description

This linker script defines the memory layout and section placement for an embedded system. It specifies the memory regions, the sections within these regions, and the addresses for different sections and symbols. Below is a detailed description of each part of the script and a table summarizing the layout.

Section	Purpose	Placement	Symbols
`.vectors`	Interrupt vector table	`flash`	`_E_vectors`
`.text`	Executable code and read-only data	`flash`	`_E_text`
`.data`	Initialized global and static variables	`sram` (from `flash`)	`_S_DATA`, `_E_DATA`
`.bss`	Uninitialized global and static variables (zeroed)	`sram`	`_S_bss`, `_E_bss`, `_stack_top`

```
MEMORY
{
    flash (rx) : ORIGIN = 0x08000000, LENGTH = 32K
    sram  (rwx) : ORIGIN = 0x20000000, LENGTH = 20K
}

SECTIONS
{
    .vectors : {
        *(.vectors*)
        _E_vectors = .;
    } >flash

    .text : {
        *(.text*)
        *(.rodata*)
        _E_text = .;
    } >flash

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

    .bss :
    {
        _S_bss = .;
        *(.bss*)
        . = ALIGN(4);
        _E_bss = .;
        . = . + 0x1000;
        _stack_top = .;
    } >sram
}
```

Makefile Explanation

This Makefile is used to compile and link a project for an ARM Cortex-M3 microcontroller. It specifies the build process, including compiling source files, linking them, and generating output files.

Symbols Summary Table

Symbol	Description	Usage Example
`$<`	Refers to the first prerequisite in a rule.	In `%.o: %.c`, `$<` refers to the `.c` file being compiled.
`$@`	Refers to the target of the rule.	In `%.o: %.c`, `$@` refers to the `.o` file being created.
`$^`	Refers to all prerequisites in a rule, excluding duplicates.	In linking rules, `$^` refers to all object files.
`$*`	Refers to the stem (base name) of the target and prerequisite.	In `%.o: %.c`, `$*` refers to the base name of files, e.g., `file`.
`%`	Used in pattern rules to match multiple files.	In `%.o: %.c`, `%` matches any base name, e.g., `file`.

Symbol	Description
`CC`	Prefix for ARM toolchain commands (e.g., `arm-none-eabi-`).
`CFLAGS`	Compiler flags for source code compilation.
`LIBS`	Libraries to be linked (currently empty).
`TARGET`	Name of the output files (without extension).
`INCS`	Include directories for header files.
`SRC`	List of `.c` source files to be compiled.
`ASM_SRC`	List of `.s` assembly source files to be assembled.
`OBJ`	List of object files generated from source and assembly files.
`all`	Default target to build the `.bin` file and print a completion message.
`%.o: %.c`	Rule to compile `.c` files into `.o` object files using `gcc`.
`%.o: %.s`	Rule to assemble `.s` files into `.o` object files using `as`.
`$(TARGET).elf`	Rule to link object files into an ELF executable using `ld`.
`$(TARGET).bin`	Rule to convert the ELF file to a binary file using `objcopy`.
`clean`	Rule to remove all build artifacts (object files, ELF, binary, AXF, and map files).

```
# Toolchain variables
CC=arm-none-eabi-
CFLAGS= -mcpu=cortex-m3 -mthumb -std=c99 -gdwarf-2
LIBS=
TARGET=Pressure_Controller_CortexM3
INCS=-I .

# Source and object file lists
SRC = $(wildcard *.c) driver\\driver.c
ASM_SRC = $(wildcard *.s)
OBJ = $(SRC:.c=.o) $(ASM_SRC:.s=.o)

# Default target
all: $(TARGET).bin
     @echo "Build is done"

# Rule to build object files from C source
%.o: %.c
     $(CC)gcc -c $(CFLAGS) $(INCS) $< -o $@

# Rule to build object files from assembly source
%.o: %.s
     $(CC)as $(CFLAGS) $(INCS) $< -o $@

# Rule to build the ELF file
$(TARGET).elf: $(OBJ)
     $(CC)ld -T linker_script.ld $(LIBS) $(OBJ) -o $@ -Map=Map_file.map
     cp $(TARGET).elf $(TARGET).axf

# Rule to build the binary file
$(TARGET).bin: $(TARGET).elf
     $(CC)objcopy -O binary $< $@

# Clean rule to remove build artifacts
clean:
     rm -f *.o *.elf *.bin *.axf *.map
```

Main.c

```c
#include <stdint.h>
#include <stdio.h>
#include "AlarmActyatorDriver.h"
#include "AlarmMonitor.h"
#include "MainAlg.h"
#include "PressureSensorDriver.h"
#include "driver/driver.h"
#include "state.h"

extern void GPIO_INITIALIZATION();

void setup() {
    STATE_INIT(PSD_init);
    STATE_INIT(AAD_init);

    PSD_state = STATE(PSD_waiting);
    MA_state = STATE(MA_LowPressureDetect);
    AM_State = STATE(AlarmOFF);
    AAD_State = STATE(AAD_init);
}

int main() {
    GPIO_INITIALIZATION();
    setup();
    while (1) {
        PSD_state();
        MA_state();
        AAD_State();
        AM_State();
    }
}
```

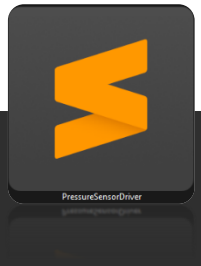
MainAlg.c

```c
#include "MainAlg.h"
#include "state.h"
#include <stdio.h>
#include "driver/driver.h"

int MA_PressureVal = 0;
int MA_threshold = 20;

void (*MA_state)();

void SetPressureValue(int p){
MA_PressureVal = p;
MA_state = (MA_PressureVal <= MA_threshold) ?  STATE(MA_LowPressureDetect) :
STATE(MA_HighPressureDetect);
      // printf("-------------- Pressure Value = %d -------------------------
\n",MA_PressureVal);

}

STATE_DEFINE(MA_HighPressureDetect){
      // printf("The CPU in MA_HighPressureDetect State .....\n");
    HighPressureDetect();
}

STATE_DEFINE(MA_LowPressureDetect) {
    // printf("The CPU is in MA_LowPressureDetect State .....\n");


}
```

MainAlg.h

```c
#ifndef MA_H_
#define MA_H_

#include "state.h"

enum {
    MA_HighPressureDetect,
    MA_LowPressureDetect
} MAStateID;

STATE_DEFINE(MA_HighPressureDetect);
STATE_DEFINE(MA_LowPressureDetect);

extern void (*MA_state)();

#endif
```

PressureSensorDriver.c




```c
#include "PressureSensorDriver.h"
#include "state.h"
#include <stdio.h>
#include "driver/driver.h"

int PSD_PressureVal ;
int PSD_PrSensorTime = 10000;
void (*PSD_state)();

// int PSD_Get_Pressure_random(int l, int r, int count) {
//     return (rand() % (r - l + 1)) + l;
// }

STATE_DEFINE(PSD_init){
    PSDStateID = PSD_init;
    // printf("The Sensor in init State ............\n");
}

STATE_DEFINE(PSD_reading){
    PSDStateID = PSD_reading;
    // printf("The Sensor in reading State ............\n");

   // PSD_PressureVal = PSD_Get_Pressure_random(15, 30, 1);
   // printf("PSD_Reading : PressureVal = %d\n", PSD_PressureVal);

    PSD_PressureVal = getPressureVal();
    SetPressureValue(PSD_PressureVal);

    PSD_state = STATE(PSD_waiting);
}

STATE_DEFINE(PSD_waiting){
    PSDStateID = PSD_waiting;

    // delay
    Delay(PSD_PrSensorTime);
    PSD_state = STATE(PSD_reading);
}
```

PressureSensorDriver.h


```
#ifndef PSD_H_
#define PSD_H_
#include "state.h"


enum {
    PSD_init,
    PSD_reading,
    PSD_waiting
} PSDStateID;

STATE_DEFINE(PSD_init);
STATE_DEFINE(PSD_reading);
STATE_DEFINE(PSD_waiting);

extern void (*PSD_state)();

#endif
```

AlarmActyatorDriver.c


```c
#include <stdio.h>
#include "state.h"
#include "AlarmActyatorDriver.h"

void (*AAD_State)();

void StartAlarm(void) {
    // printf("The Alarm in Start State ............\n");
    Set_Alarm_actuator(0);
    AAD_State = STATE(AAD_waiting);
}

void StopeAlarm(void) {
    //printf("The Alarm in Stope State ............\n");
    Set_Alarm_actuator(1);
    AAD_State = STATE(AAD_waiting);
}

STATE_DEFINE(AAD_init) {
    AADStateID = AAD_init;
    // printf("The Sensor in init State ............\n");
}

STATE_DEFINE(AAD_waiting) {
    AADStateID = AAD_waiting;
    // printf("The Alarm in waiting State ............\n");
}
```

AlarmActyatorDriver.h


```c
#ifndef AAD_H_
#define AAD_H_

#include "state.h"

enum {
    AAD_init,
    AAD_waiting,
    AAD_AlarmON,
    AAD_AlarmOFF
} AADStateID;

STATE_DEFINE(AAD_init);
STATE_DEFINE(AAD_waiting);

extern void (*AAD_State)();

#endif
```

AlarmMonitor.c


```c
#include <stdio.h>
#include "state.h"
#include "AlarmMonitor.h"

int AlarmTime = 10000;
int AlarmPeriod = 0;

void (*AM_State)();

void HighPressureDetect(void) {
    // printf("The CPU in High Pressure Detect ............\n");
    AM_State = STATE(AlarmON);
}

STATE_DEFINE(AlarmOFF) {
    AMStateID = AlarmOFF;
    // printf("The CPU in Alarm OFF State ............\n");
    StopeAlarm();
}

STATE_DEFINE(AlarmON) {
    AMStateID = AlarmON;
    // printf("The CPU in Alarm ON State ............\n");
    StartAlarm();
    AM_State = STATE(waiting);
}

STATE_DEFINE(waiting) {
    AMStateID = waiting;
    // printf("The CPU in waiting State ............\n");

    // delay
    Delay(AlarmTime);
    AM_State = STATE(AlarmOFF);
}
```

AlarmMonitor.h


```c
#ifndef AM_H_
#define AM_H_

#include "state.h"

enum {
    AlarmOFF,
    AlarmON,
    waiting
} AMStateID;

STATE_DEFINE(AlarmOFF);
STATE_DEFINE(AlarmON);
STATE_DEFINE(waiting);

extern void (*AM_State)();

#endif
```