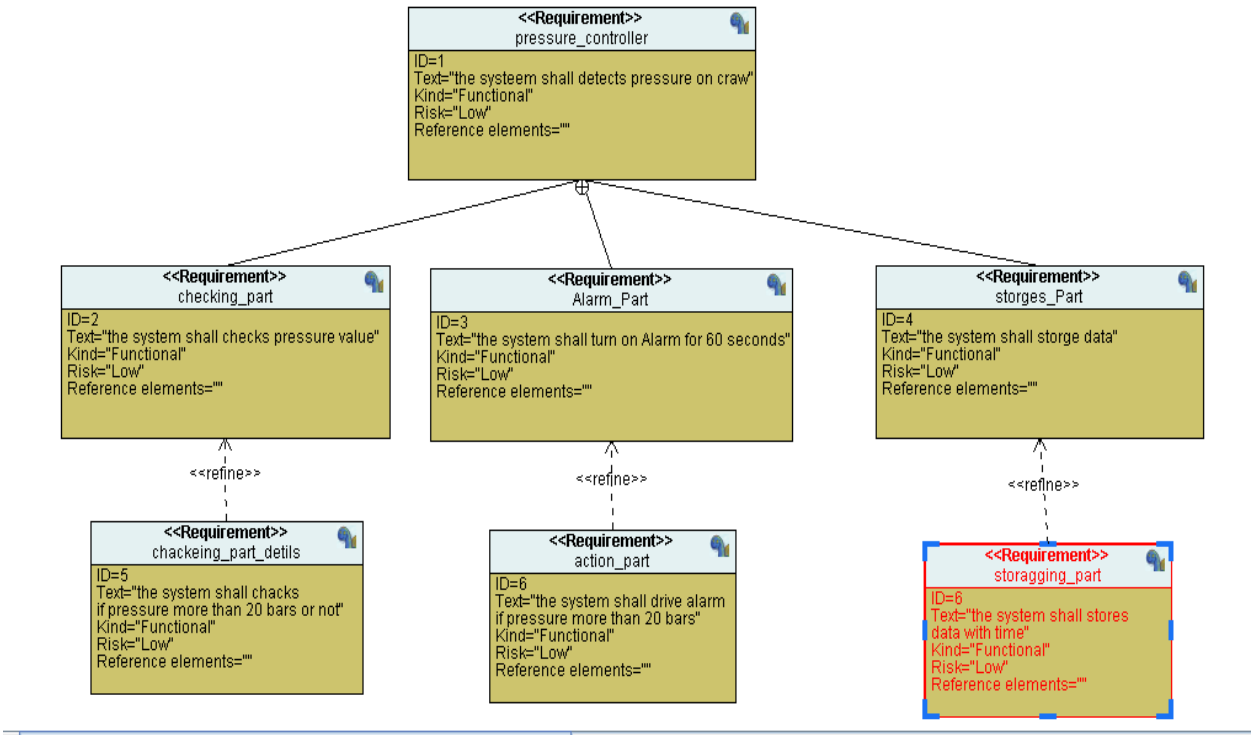
Contents

Introduction

In this project we will showing all step requirement , system analysis such as use case diagram ,activity diagram and sequence diagram and system design ,this will consist of application ,pressure sensor ,alarm and EEPROM and we will create our toolchain such as startup code, linker and make file.

Chapter 1: requirements

This is the first step in our project or anyone ,so in this step we will describe by details all part of requirement by requirement diagram.



1-pressure sensor will check pressure value on plane

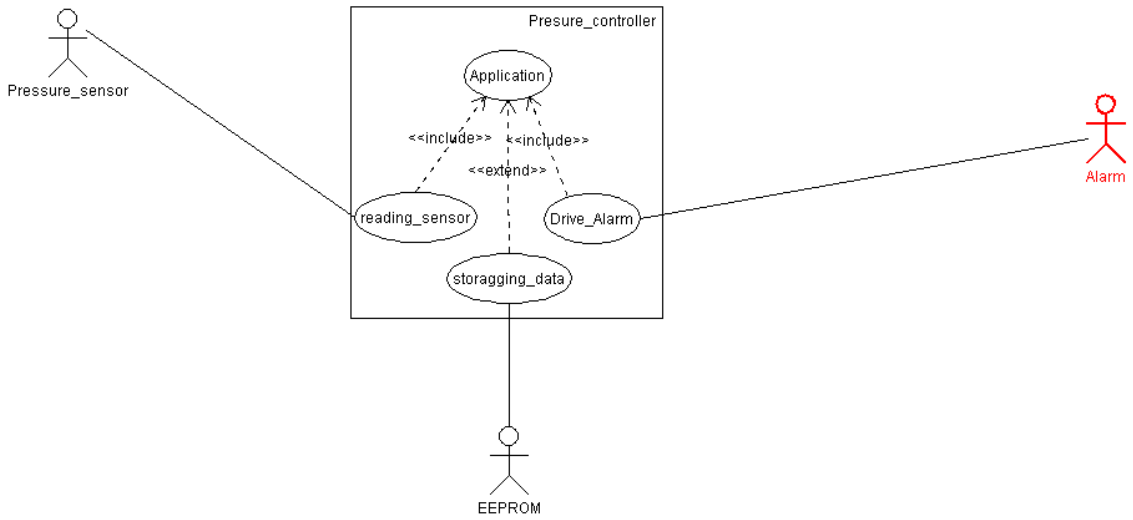
2-when pressure exceeds 20 bars Alarm will drive to 60 second

3-At this time ROM will gets a signal carries pressure value and time information.

Chapter 2: System Analysis

This step describes main components of systems and we have three methods of system analysis.

1- Use Case Diagram :



1- Hardware Components:

Pressure sensor , Alarm and EEPROM.

2- Software components:

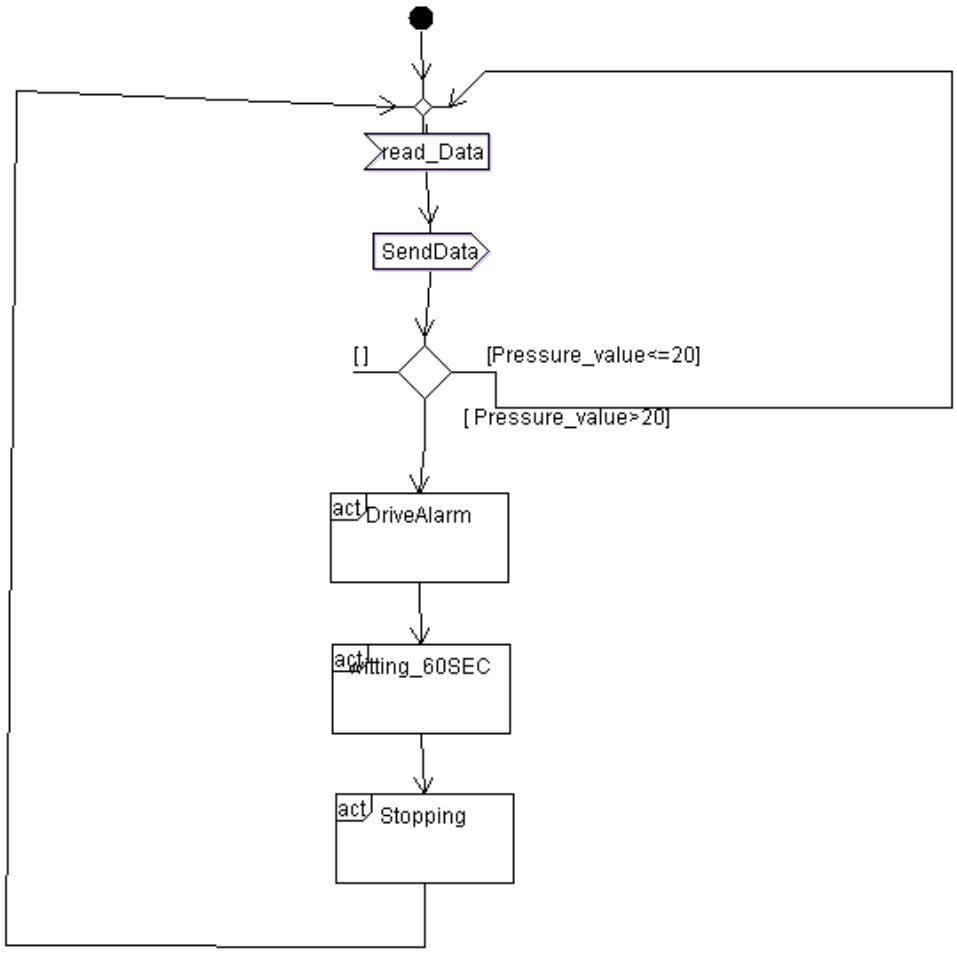
A- Include

Reading sensor

Alarm Driver

B- Extended

2- Activity Diagram



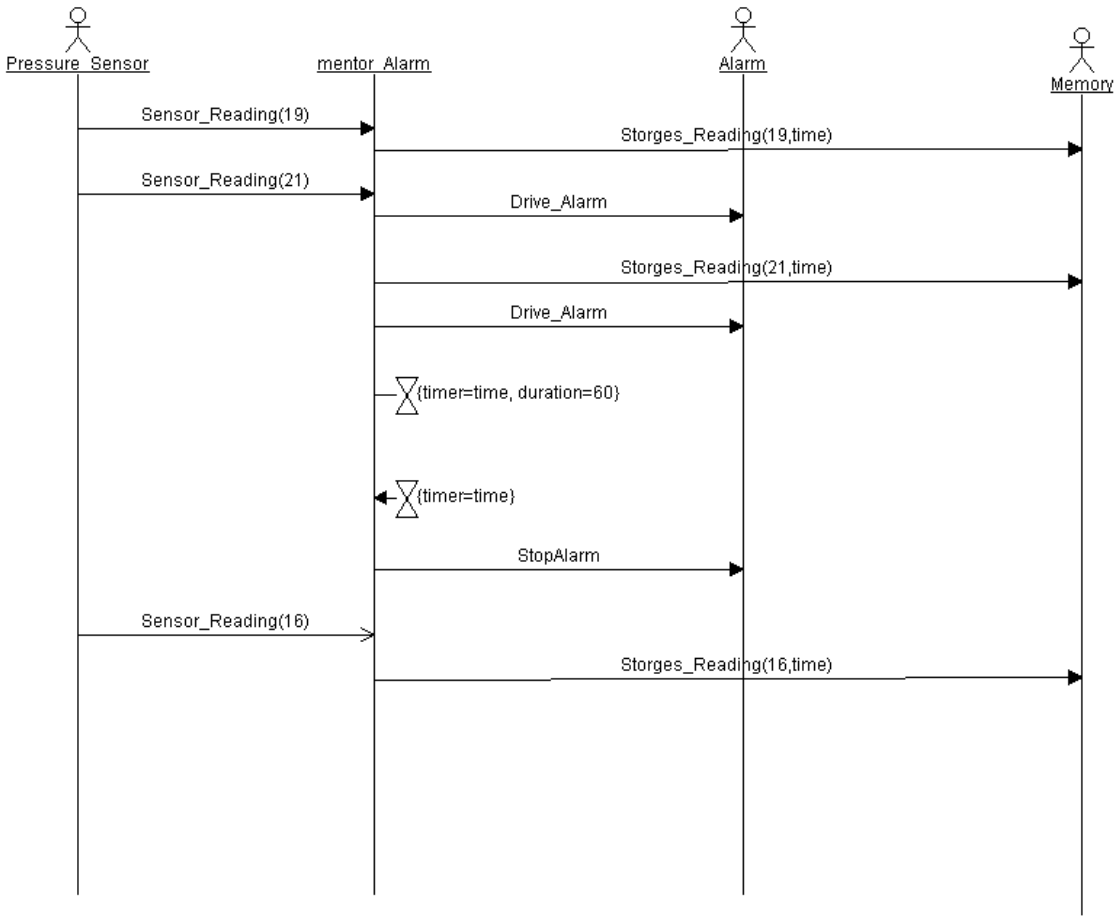
1- sensor shall checks pressure at cabin

2- system shall checks pressure value if it exceeds 20 bars .

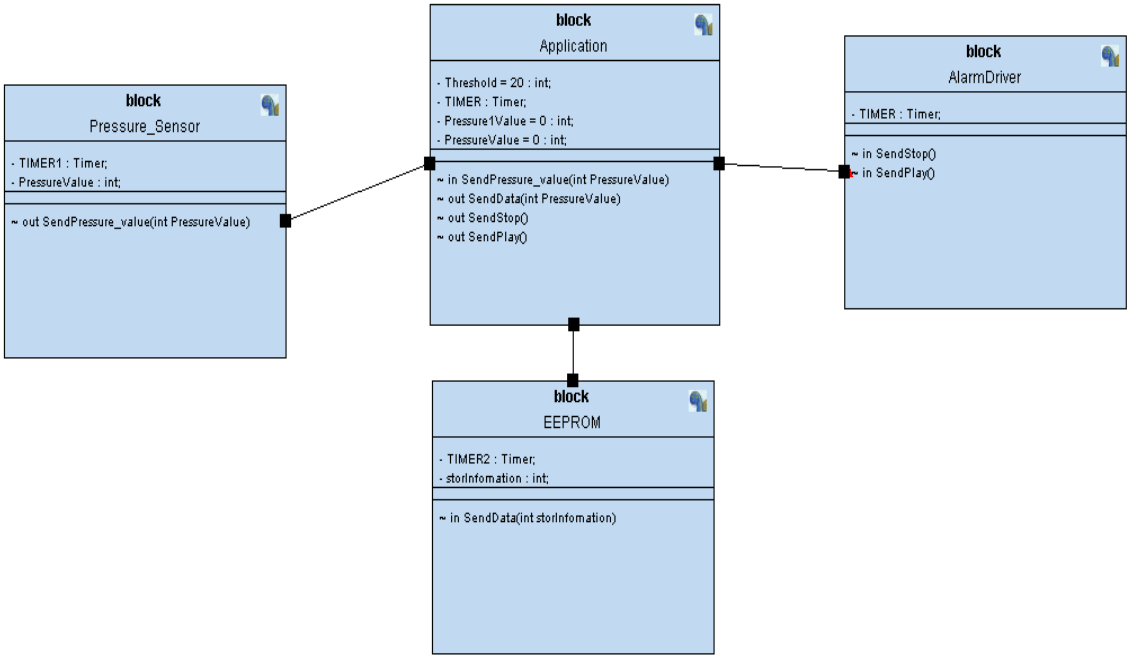
3- Alarm will be on to 60 sec and stop

3- Sequence Diagram:

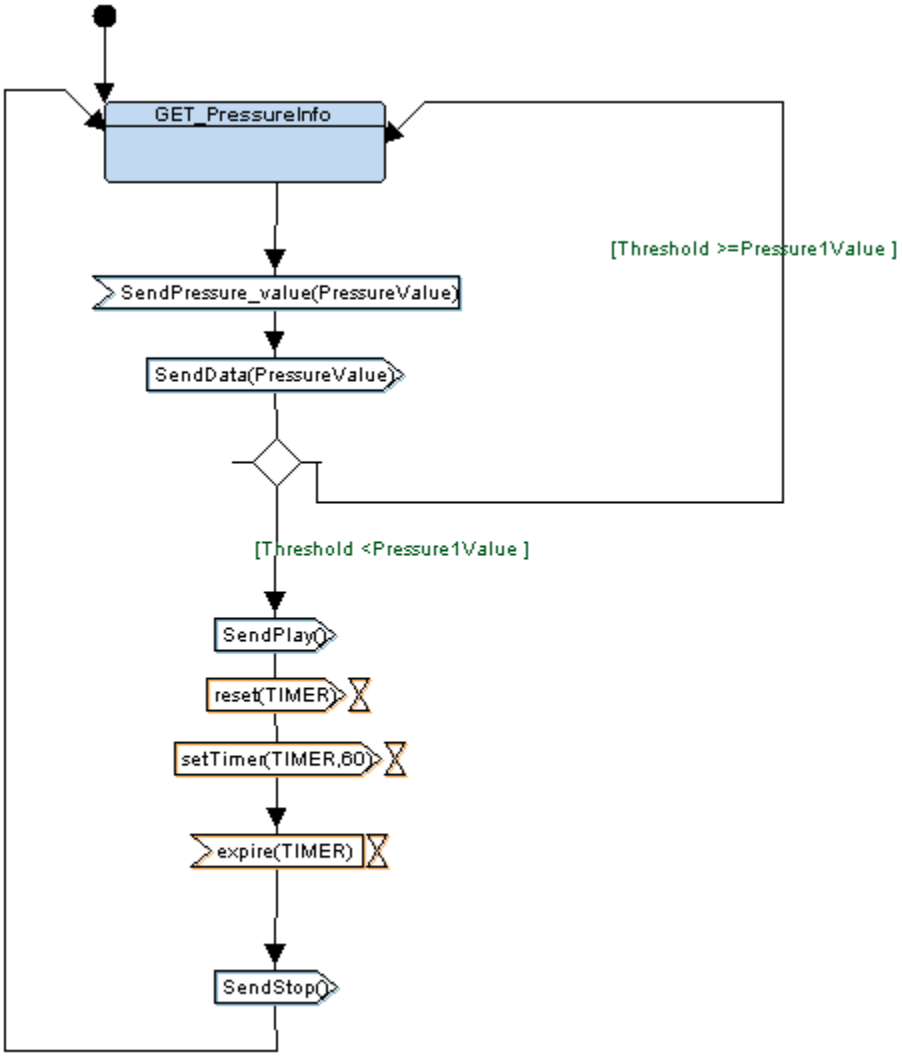
This sequence describes Scenario of system when is driving.



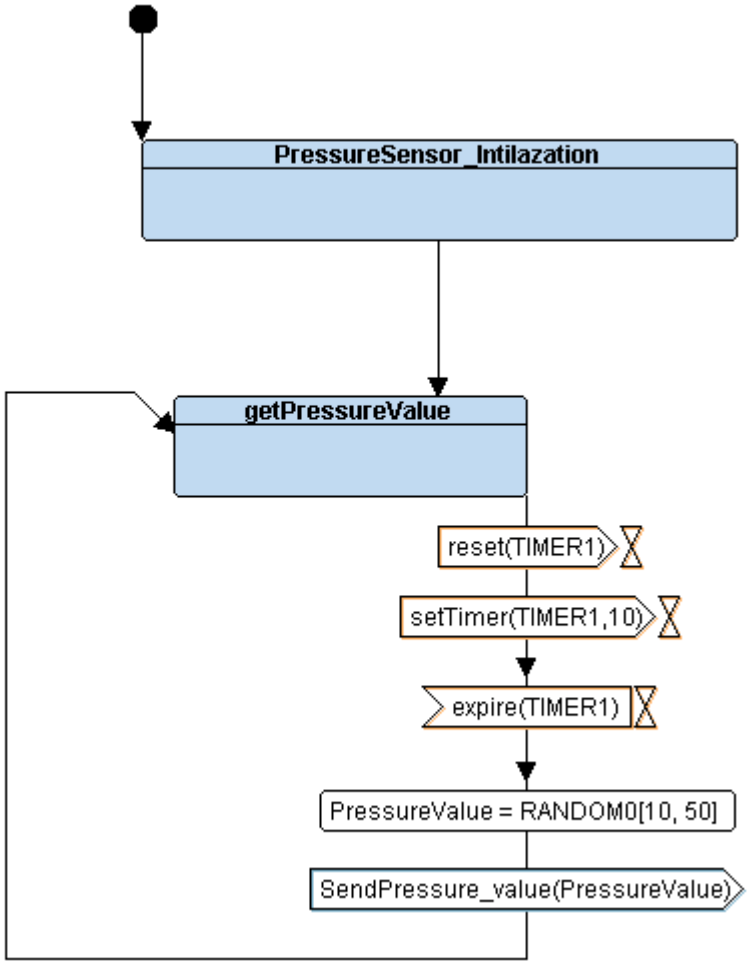
Chapter 3: System Design


block
Application
- Threshold = 20 : int;
- TIMER : Timer;
- Pressure1Value = 0 : int;
- PressureValue = 0 : int;
~ in SendPressure_value(int PressureValue)
~ out SendData(int PressureValue)
~ out SendStop()
~ out SendPlay()
block
Pressure_Sensor
- TIMER1 : Timer;
- PressureValue : int;
~ out SendPressure_value(int PressureValue)
block
AlarmDriver
- TIMER : Timer;
~ in SendStop()
~ in SendPlay()
block
EEPROM
- TIMER2 : Timer;
- storInfomation : int;
~ in SendData(int storInfomation)

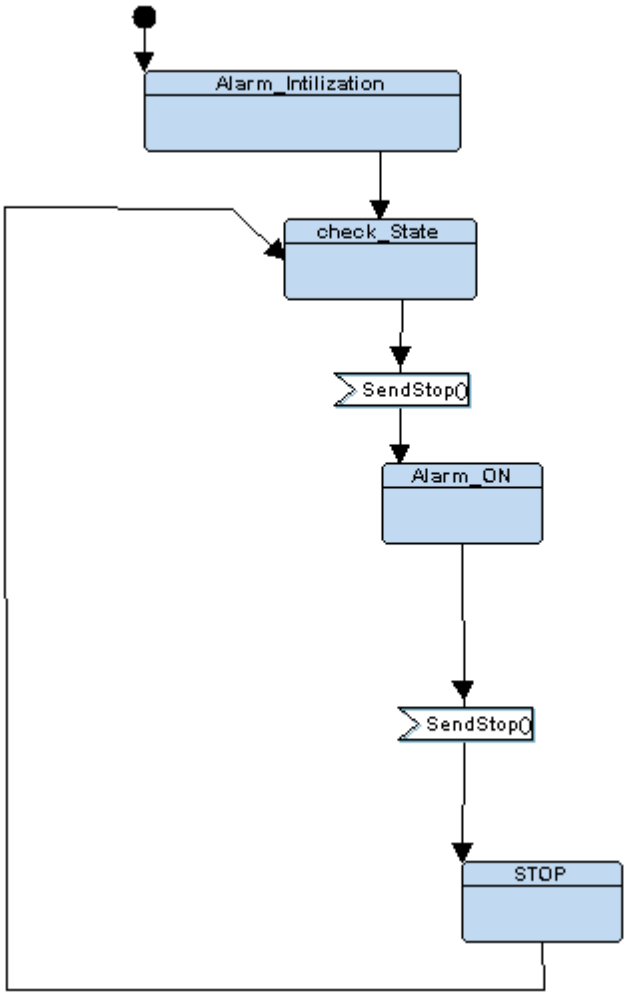
1- Application:



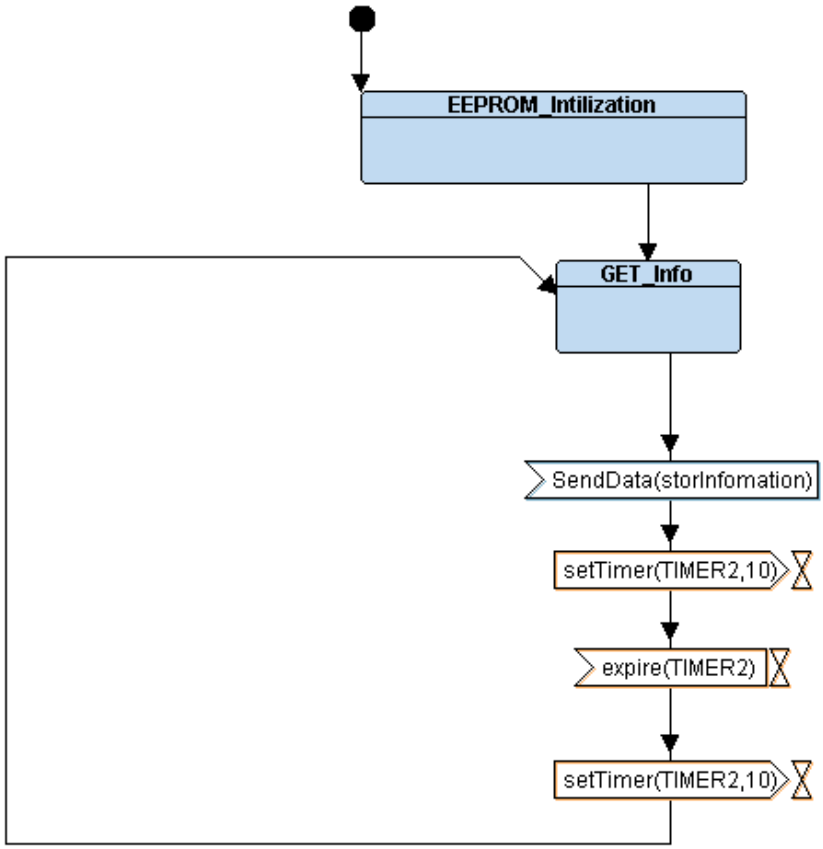
2- Pressure sensor



3- Alarm driver



4- EEPROM



Chapter4: toolchain

In this chapter we will create our toolchain that will help us to implement binary file or hex file

1- Startup code:

It is a part of startup code necessary to run our code ,It has vector table and it branches to main function.

The first address in this file is address of stack pointer stores in flash at first address and (_reset) symbol

Using when processor get reset massage by hardware it will branches to main.

Makefile | startup.s

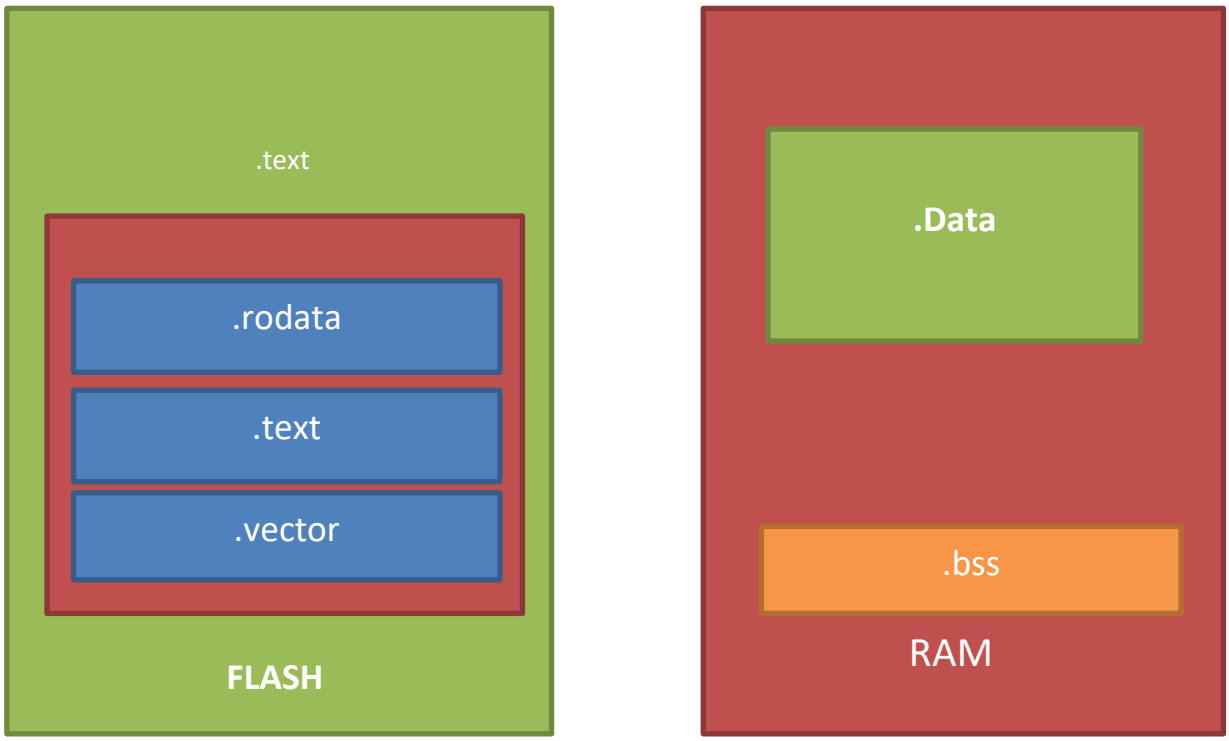
```
4
5   /*Vector section*/
6   .section .vector  /*this commend in processor means creat section calls vector and save this symbols on it*/
7   .word 0x20001000 /*stack top address this address of SP in ram and it=0x08000000 in flash*/
8   .word _reset      /*reset interrupt by hardware it jumming to main in fash=0x08000004*/
9   .word vector_handelar  /*Non maskable interrupt*/
10  .word vector_handelar  /*hard fault*/
11  .word vector_handelar  /*bus fault*/
12  .word vector_handelar  /*usage fault*/
13  .word vector_handelar  /*reserved*/
14  .word vector_handelar  /*reserved*/
15  .word vector_handelar  /*reserved*/
16  .word vector_handelar  /*reserved*/
17  .word vector_handelar  /*SV call*/
18  .word vector_handelar /*debug reserved*/
19  .word vector_handelar /*reserved*/
20  .word vector_handelar /*pendSv*/
21  .word vector_handelar /*Systick*/
22  .word vector_handelar /*IRQ0*/
23  .word vector_handelar /*IRQ1*/
24  .word vector_handelar /*IRQ2*/
25  .word vector_handelar /*On to IRQ7*/
26  /*to know address in flash add 4 bytes*/
27
28
29  /*this section .text has codes of main*/
30  .section .text
31  _reset:
32      bl main
33      b .              /*loop in main */
34
35  .thumb_func
36  vector_handelar:    /*when hardware send handlar interrupt tragger processor will vectes to _reset and reset will jumps to main */
37      b _reset
38
```

2-linker script

```
MEMORY
{
    FLASH (rx)   : ORIGIN = 0x08000000, LENGTH = 64K
    RAM    (rwx) : ORIGIN = 0x20000000, LENGTH = 20K
}


SECTIONS
{
   .text : {

   *(.vector*)
   *(.text*)
   *(.rodata*)
   } >FLASH

   .data : {
       *(.data*)

   } > FLASH

   .bss : {
   *(.bss*)} >RAM




    }
```

In this file we divided memory to some section

NOT in this project we will working in FLASH.



3- Makefile

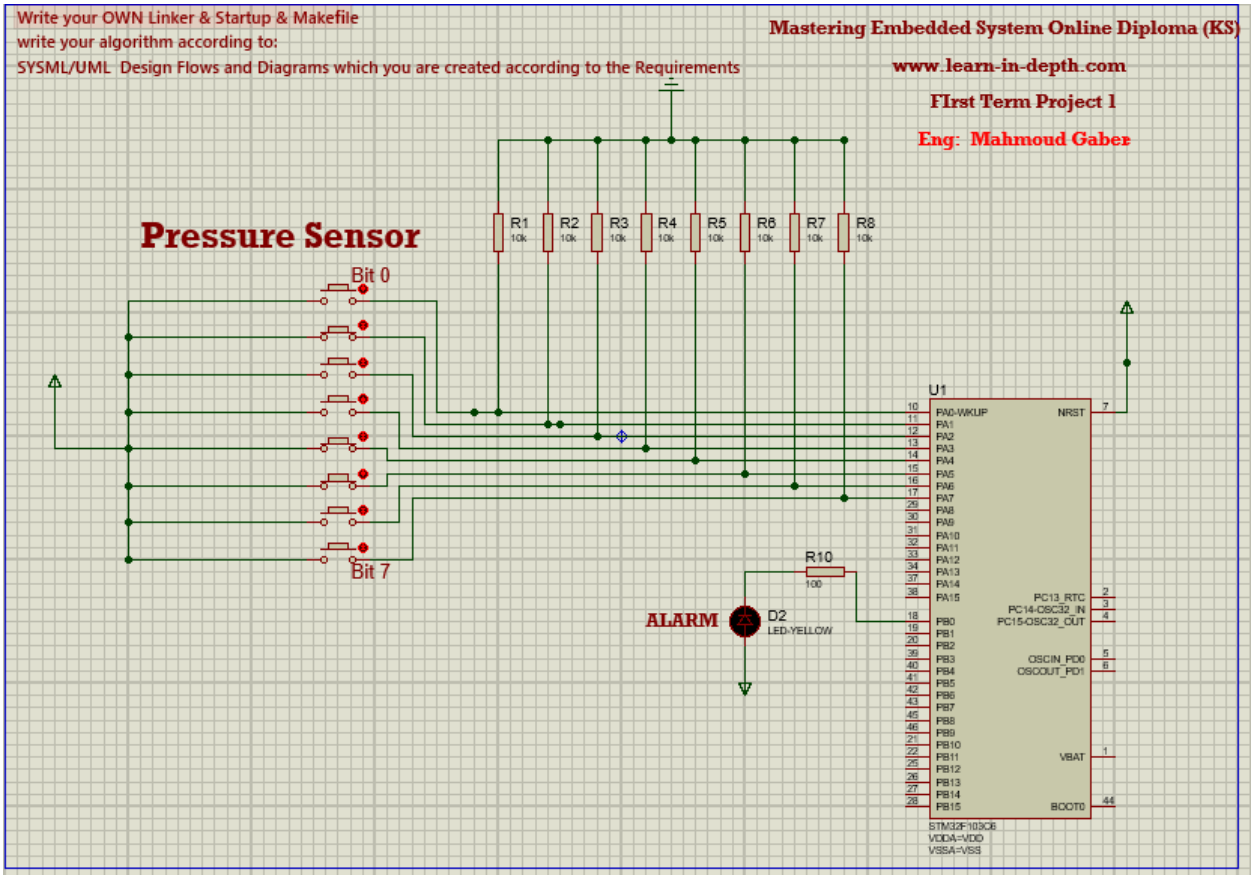
```
CC=arm-none-eabi-#replace all 'name of chip by $(CC)'
CFLAGS=-mcpu=cortex-m3 -gdwarf-2
INCS=-I
LIBS=#replace libs by 'LIBS'
SRC=$(wildcard *.c)#collect all $(PROJECT_NAME).c and replaces by 'SRC'
OBJ=$(SRC:.c=.o)#create same names of $(PROJECT_NAME)s.c to $(PROJECT_NAME)s.o
AS=$(wildcard *.s)#collect all $(PROJECT_NAME).s and replaces by 'AS'
ASOBJ=$(AS:.s=.o)#create same names of $(PROJECT_NAME)s.s to $(PROJECT_NAME)s.o
LINK=linker_script.ld
PROJECT_NAME=FILE
all: $(PROJECT_NAME).bin
    @echo "========bulid has been done=========="

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


%.o: %.s
    $(CC)as.exe  $(CFLAGS)  $< -o $@

$(PROJECT_NAME).elf: $(OBJ) $(ASOBJ)
    $(CC)ld.exe -T $(LINK) $(LIBS) $(OBJ) $(ASOBJ) -o $@ -Map=map_$(PROJECT_NAME).map
$(PROJECT_NAME).bin:$(PROJECT_NAME).elf
    $(CC)objcopy.exe  -O binary $< $@



clean obj:
    rm -rf *.o *~
    @echo "==================== All object $(PROJECT_NAME)s have been removed=================="
clean_all:
    rm -rf *.bin  *.elf *~
    @echo "==================== All execution $(PROJECT_NAME)s have been removed=================="

debug:
    $(CC)objdump.exe $(PROJECT_NAME).elf -h
```

Cahpter5: software

https://github.com/mahmoudgaber97/term_project/tree/main/pressure%20controller/CODE

chapter6: Hardware

1- circuit diagram



2- video

https://github.com/mahmoudgaber97/term_project/tree/main/pressure%20controller

ENG .Mahmoud Gaber