Mastering embedded system online diploma

www.learn-in-depth.com

First term (Final project 1)

Eng. Mo’men Amr Mohammed Amr

My profile: https://www.learn-in-depth.com/online-diploma/momen55amr@gmail.com

Abstract:

This report follows an embedded system architecture sequence using UML through the different stages from case study to system design, including also the c language code implementing the design with proteus simulation results.

1.Case study

The client wants to acquire a high-pressure alarm to be installed inside an air plane cabin to warn the crew if the pressure reaches a predetermined level that they can't handle, to take active measures regarding their safety.

I. Specifications

- A pressure controller: to inform the crew with an alarm if the pressure exceeds 20 pars inside the cabin.
- A timer: for the alarm to word for 60 seconds.

II. Assumptions

1. The controller setup and shutdown procedures are not modeled.
2. The controller maintenance is not modeled.
3. The pressure sensor never fails.
4. The alarm never fails.
5. The controller never faces a power cut.

2.Methodology

The system can follow any SDLC or STLC since it's not demanding so we chose v cycle just for purpose of demonstrating although technically not used.

3.Requirements

As stated early, we use UML to describe the system with easy going labels and identifiers to be simple as much as we can without going into engineering terms and complications, so the client can understand the concept of design clearly.

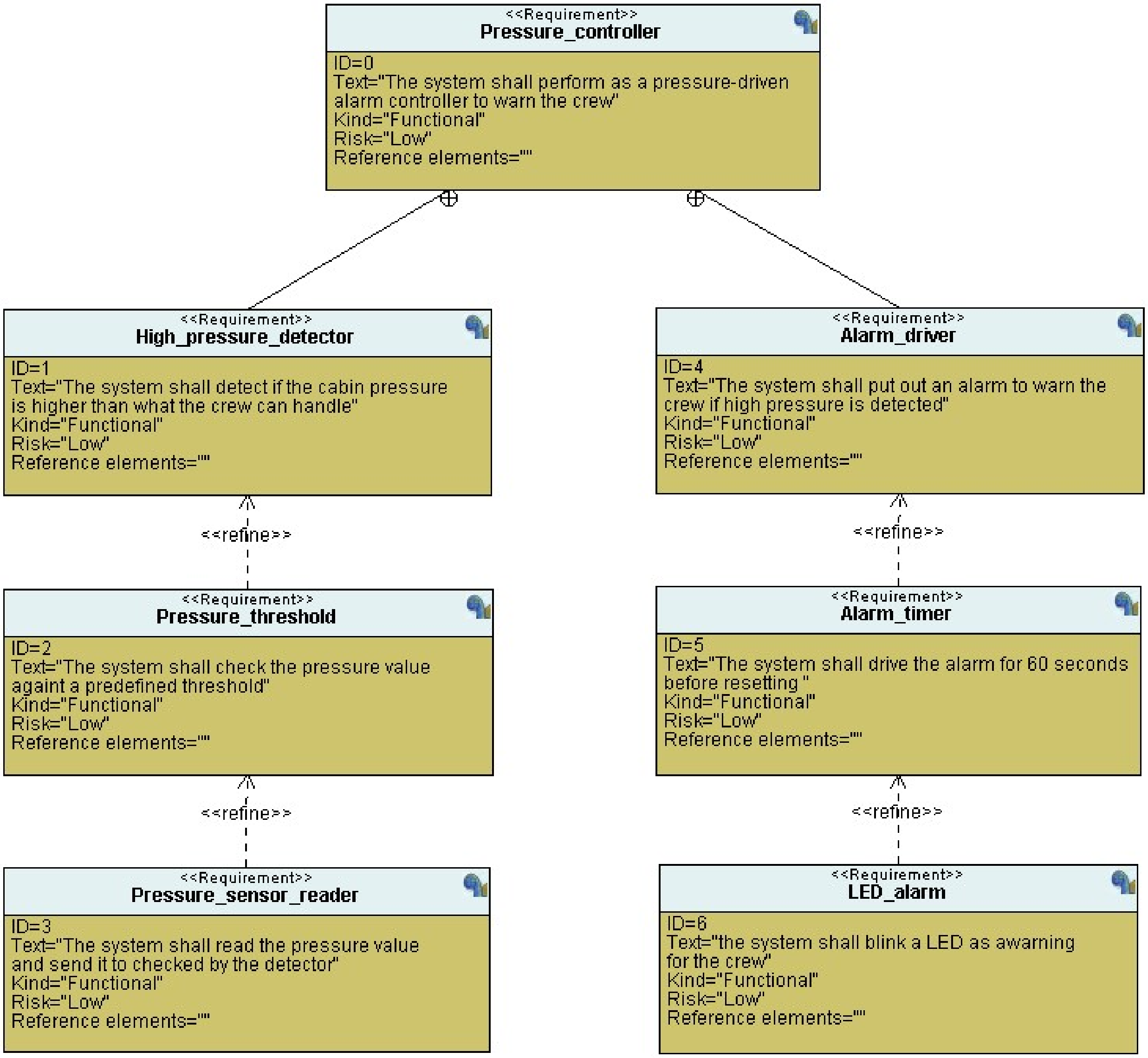


Figure 1 shows the UML requirement diagram

4.Space exploration

We can bypass all the steps from partitioning to performance since we must work with stm32F103 and the system is not demanding so we skipped through it.

5.System analysis

we use UML to describe the system with simple diagrams without going into engineering terms and complications, so the client can understand the concept of design clearly.

I. Use case diagram

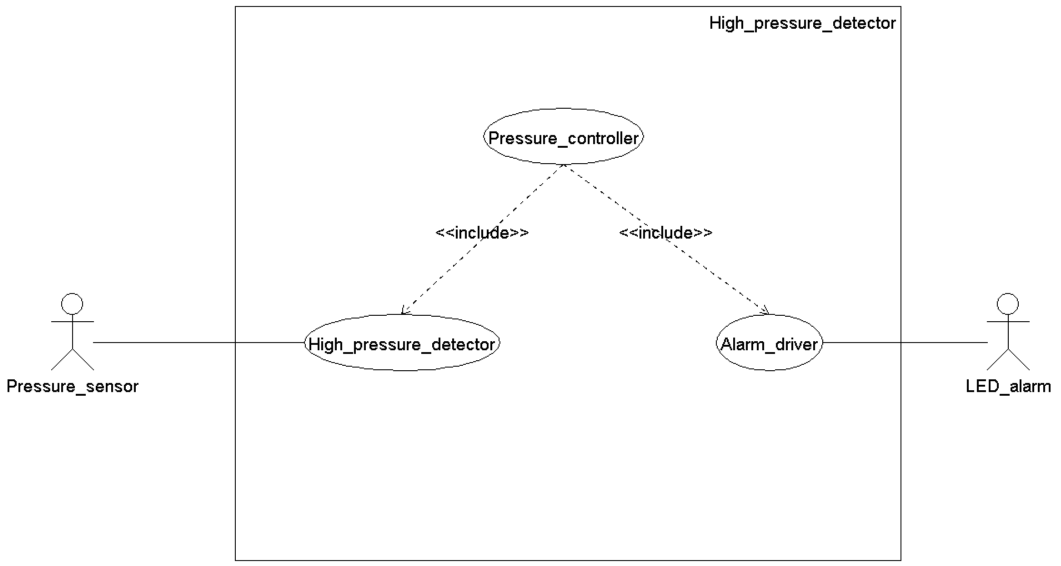


Figure 2 shows the UML use case diagram

II. Sequence diagram

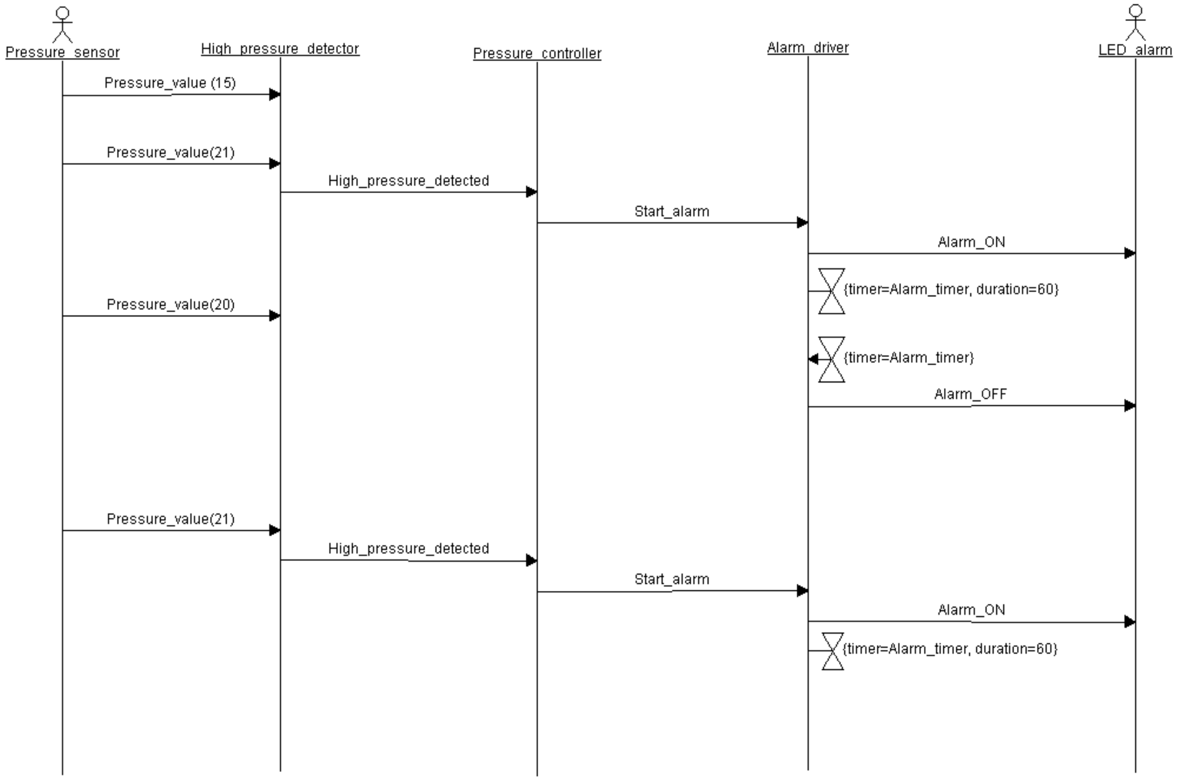


Figure 3 shows the UML sequence diagram

III. Activity diagram

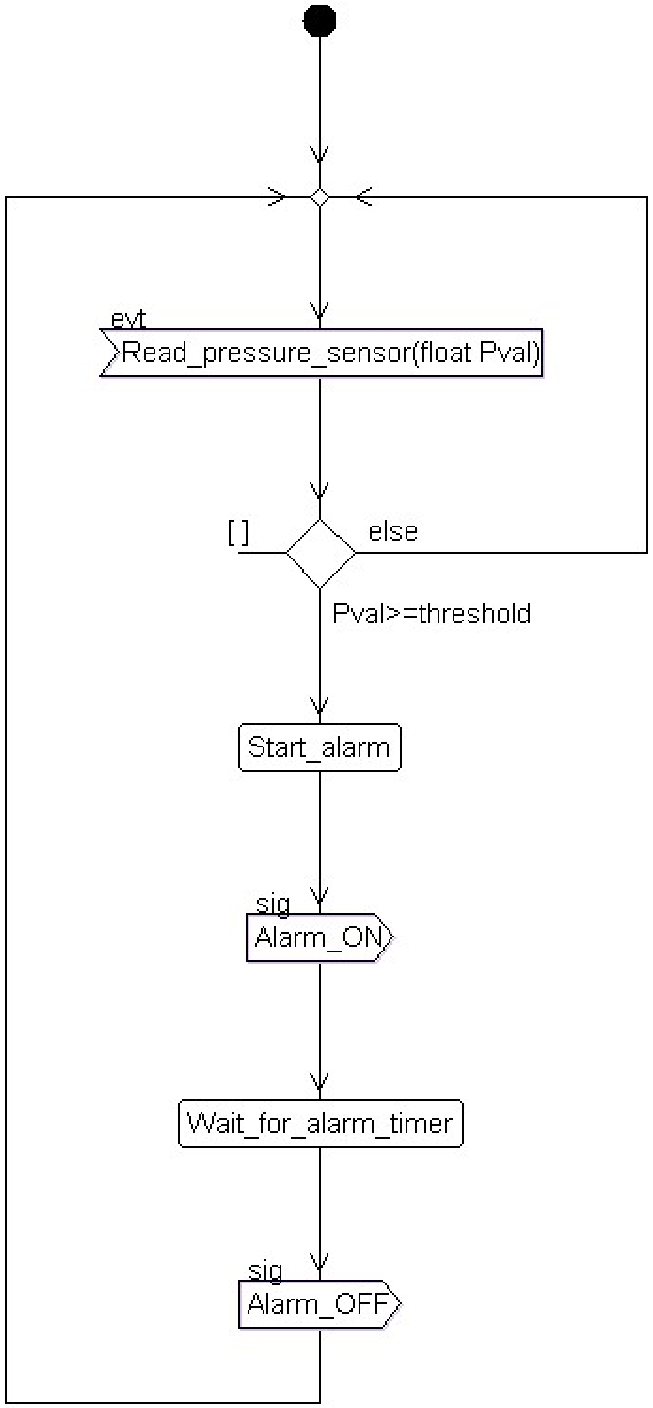


Figure 4 shows the UML activity diagram

6.System design

we use UML to describe the design with block and state diagrams without going into engineering terms and complications, so the client can understand the concept of design clearly.

I. Block diagram

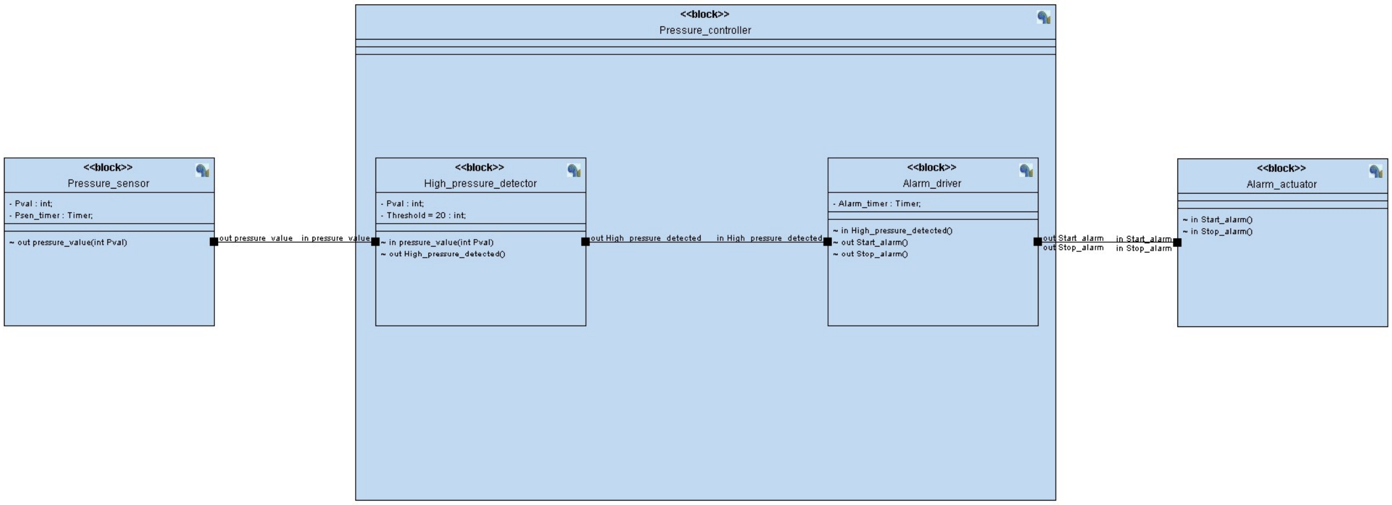


Figure 5 shows the UML design block diagram

II. Pressure sensor and high-pressure detector state diagrams

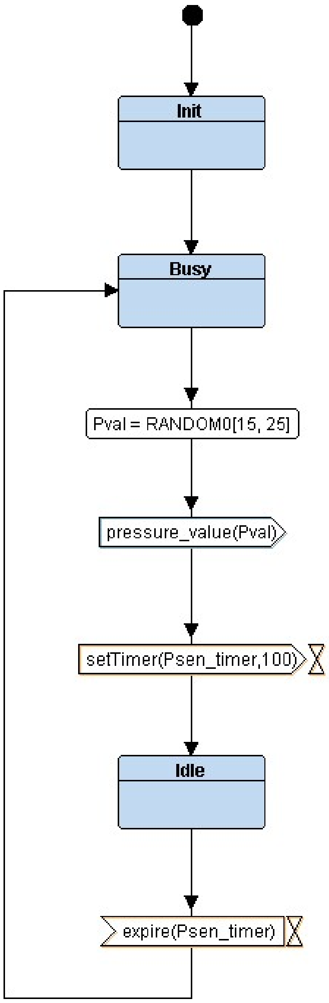


Figure 6shows the UML pressure sensor state diagram

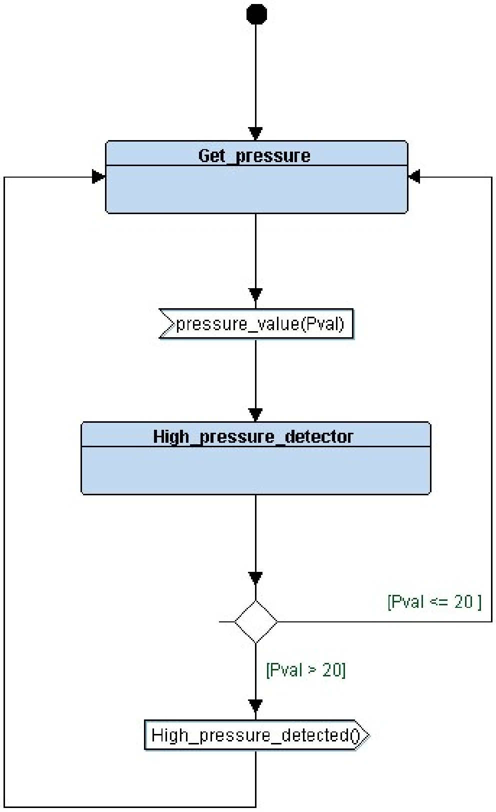


Figure 7 shows the UML high pressure detector state diagram

III. Alarm driver and actuator state diagrams

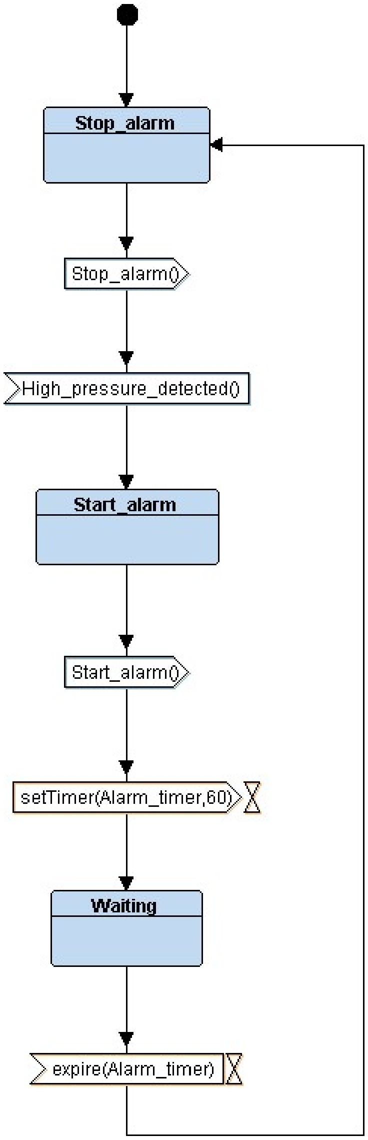


Figure 8 shows the UML alarm driver state diagram

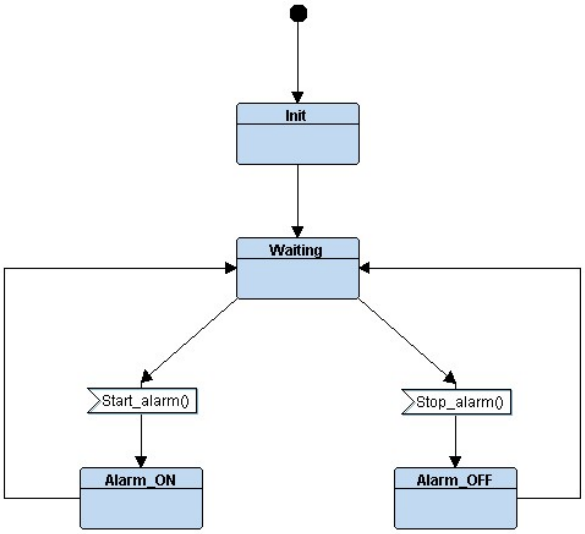


Figure 9 shows the UML alarm actuator state diagram

7.Interactive simulation

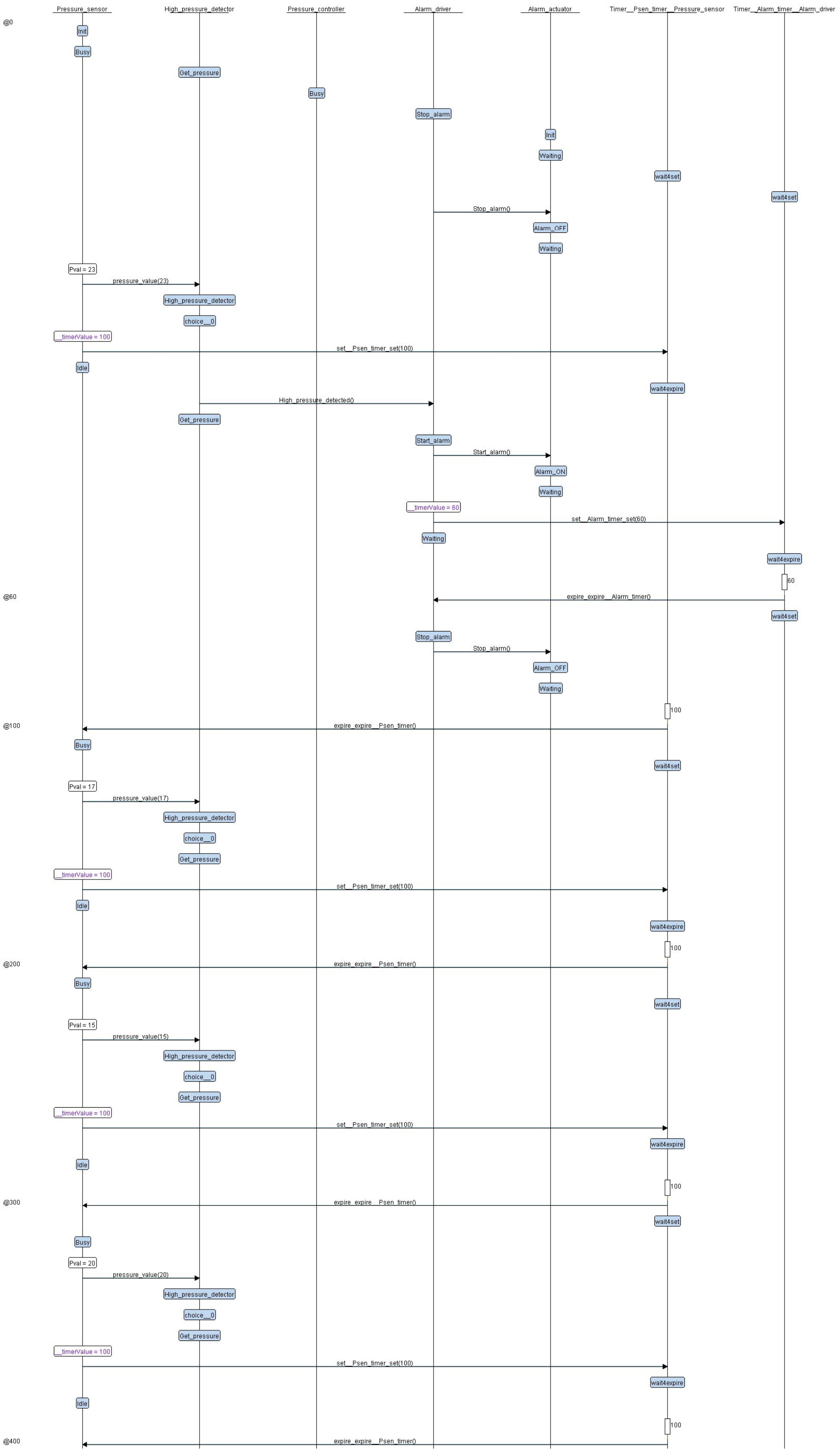


Figure 10 shows the software logic verification diagram

8.C code files

I. Pressure sensor c&h files

```
/*
 * Pressure_sensor.h
 *
 *  Created on: Jun 25, 2022
 *      Author: momen
 */

#ifndef PRESSURE_SENSOR_H_
#define PRESSURE_SENSOR_H_

void (*Psensor_STATE)();

STATE_define(PS_init);
STATE_define(PS_busy);
STATE_define(PS_idle);

#endif /* PRESSURE_SENSOR_H_ */
```

```
/*
 * Pressure_sensor.c
 *
 *  Created on: Jun 25, 2022
 *      Author: momen
 */

#include"state.h"
#include"Pressure_sensor.h"

int Pval;

STATE_define(PS_init)
{      state_Psensor_id = PS_init;
       // initialize parameters for pressure sensor
       Psensor_STATE = STATE(PS_busy);
}

STATE_define(PS_busy)
{      state_Psensor_id = PS_busy;
       //reading the gpio port set as 8bit input
       Pval = getPressureVal();
       //although it's lacking, sending the sensor reading through a channel to the detector
       //block, which is why we use global variable in the first place but this is for
       //demonstrating modularity only
       HPD_aquire_value(Pval);
       //after sending the reading to the detector we set the parameters for it to function
       //properly when it runs then return here so we don't call the detector function here
       //leading to stack //consumption
       Psensor_STATE = STATE(PS_idle);
}

STATE_define(PS_idle)
{
       //to use delay here is not effective for this to function as a guard, we must use
       //hardware timer and since we can't yet, we can implement the guard differently by
       //checking if an alarm is OFF
       state_Psensor_id = PS_idle;
       if(state_Alarm_actuator_id == AA_OFF) Psensor_STATE = STATE(PS_busy);
}
```

II. High pressure detector c&h files

```c
/*
 * High_pressure_detector.h
 *
 *  Created on: Jun 25, 2022
 *      Author: momen
 */

#ifndef HIGH_PRESSURE_DETECTOR_H_
#define HIGH_PRESSURE_DETECTOR_H_

void (*HPdetector_STATE)();

STATE_define(HPD_calculating);

#endif /* HIGH_PRESSURE_DETECTOR_H_ */

/*
 * HPD.c
 *
 *  Created on: Jun 25, 2022
 *      Author: momen
 */

#include"state.h"
#include"High_pressure_detector.h"

int Psensor_val;
int HPD_threshold = 20;

void HPD_aquire_value (int p)
{
	//this function acts as a channel between the sensor and the detector and sets the
	//detector to enter the
	//calculating state next time it's called to check the reading against the threshold
	state_HPdetector_id = HPD_aquiring;
	Psensor_val = p;
	HPdetector_STATE = STATE(HPD_calculating);
}

STATE_define(HPD_calculating)
{
	//this is a guard to not overwhelm the alarm driver when sending every time the function
	//is called as it can be called for two times already when the alarm driver tries to
	//enter the waiting state leading //it to go into start state again so the function is
	//called only once when the sensor sends a higher value than the threshold and if the
	//detector still reads the same reading because the sensor is blocked due to alarm being
	//activated then it doesn't call the function
	if((state_HPdetector_id == HPD_aquiring)&&(Psensor_val > HPD_threshold))
	High_pressure_detected ();
	state_HPdetector_id = HPD_calculating;
}
```

III. Pressure controller c file

```c
/*
 * main.c
 *
 *  Created on: Jun 25, 2022
 *      Author: momen
 */

#include"state.h"
#include"driver.h"
#include"Pressure_sensor.h"
#include"High_pressure_detector.h"
#include"Alarm_driver.h"
#include"Alarm_actuator.h"

void setup()
{
	//this function will be executed only once at the start which initialize all the modules
	//needed for the program
	GPIO_INITIALIZATION ();
	STATE(PS_init) ();
	STATE(AA_init) ();
	//no need to initialize all the function pointers as they will be initialized when the
	//preceding functions are called
	Alarm_driver_STATE = STATE(AD_stop);
}

void Pressure_controller()
{
	setup();
	while(1)
	{
		//instead of calling a function inside another we can use pointers to functions to
		//work in periodic blocking manner
		Psensor_STATE();
		HPdetector_STATE();
		Alarm_driver_STATE();
		Alarm_actuator_STATE();
	}
}
```

IV. Alarm driver c&h files

```c
/*
 * Alarm_driver.h
 *
 *  Created on: Jun 25, 2022
 *      Author: momen
 */

#ifndef ALARM_DRIVER_H_
#define ALARM_DRIVER_H_

void (*Alarm_driver_STATE)();

STATE_define(AD_stop);
STATE_define(AD_start);
STATE_define(AD_waiting);

#endif /* ALARM_DRIVER_H_ */


/*
 * Alarm_driver.c
 *
 *  Created on: Jun 25, 2022
 *      Author: momen
 */


#include"state.h"
#include"Alarm_driver.h"

void High_pressure_detected ()
{
       //this works as guard because the alarm driver has a delay before closing the alarm so
       //if the function gets called when it is in stop state when it starts the alarm and goes
       //to the waiting state which will be called the next time it's not the way to go and has
       //too many flows and is not recommended to implement but as instructed to work in an
       //emulated synchronous blocking process without calling functions inside functions to
       //guard the stack this is it

       if(state_Alarm_driver_id == AD_stop) Alarm_driver_STATE = STATE(AD_start);
}

STATE_define(AD_stop)
{      state_Alarm_driver_id = AD_stop;
       //sending a signal to the alarm actuator to turn off
       AD_set_alarm ('f');
}

STATE_define(AD_start)
{      state_Alarm_driver_id = AD_start;
       //sending a signal to the alarm actuator to turn on
       AD_set_alarm ('o');
       Alarm_driver_STATE = STATE(AD_waiting);
}

STATE_define(AD_waiting)
{      state_Alarm_driver_id = AD_waiting;
       //waiting for supposed 60 seconds before turning the alarm off, it's more like 6 seconds
       //not 60 but the idea works as intended
       Delay(3600000);
       Alarm_driver_STATE = STATE(AD_stop);
}
```

V. Alarm actuator c&h files

```c
/*
 * Alarm_actuator.h
 *
 *  Created on: Jun 25, 2022
 *      Author: momen
 */

#ifndef ALARM_ACTUATOR_H_
#define ALARM_ACTUATOR_H_

void (*Alarm_actuator_STATE)();

STATE_define(AA_init);
STATE_define(AA_ON);
STATE_define(AA_OFF);
STATE_define(AA_waiting);

#endif /* ALARM_ACTUATOR_H_ */

/*
 * Alarm_actuator.c
 *
 *  Created on: Jun 25, 2022
 *      Author: momen
 */

#include"state.h"
#include"Alarm_actuator.h"

void AD_set_alarm (char c)
{      //this works as a channel between the driver and the actuator so whenever a signal is
       //sent, the alarm goes from waiting state into closing or opening the alarm
       if(c=='o') Alarm_actuator_STATE = STATE(AA_ON);
       else if(c=='f') Alarm_actuator_STATE = STATE(AA_OFF);
}

STATE_define(AA_init)
{      state_Alarm_actuator_id = AA_init;
       //initializing the alarm module
       Set_Alarm_actuator(1);
       Delay(10000);
       Alarm_actuator_STATE = STATE(AA_waiting);
}

STATE_define(AA_waiting)
{      //do nothing
       state_Alarm_actuator_id = AA_waiting;
}

STATE_define(AA_ON)
{      state_Alarm_actuator_id = AA_ON;
       //drive the gpio pin set as output to low driving the external load to high
       Set_Alarm_actuator(0);
       Alarm_actuator_STATE = STATE(AA_waiting);
}

STATE_define(AA_OFF)
{      state_Alarm_actuator_id = AA_OFF;
       //drive the gpio pin set as output to high driving the external load to low
       Set_Alarm_actuator(1);
       Alarm_actuator_STATE = STATE(AA_waiting);
}
```

VI. State h file

```
/*
 * state.h
 *
 *  Created on: Jun 25, 2022
 *      Author: momen
 */

#ifndef STATE_H_
#define STATE_H_

#include"stdio.h"
#include"stdlib.h"
#include"driver.h"

enum
{
        PS_init, PS_busy, PS_idle
}
state_Psensor_id;


enum
{
        HPD_aquiring, HPD_calculating
}
state_HPdetector_id;

enum
{
        AD_stop, AD_start, AD_waiting
}
state_Alarm_driver_id;

enum
{
        AA_init, AA_ON, AA_OFF, AA_waiting
}
state_Alarm_actuator_id;


#define STATE_define(state_function) void ST_##state_function()
#define STATE(state_function) ST_##state_function

void HPD_aquire_value (int p);
void High_pressure_detected ();
void AD_set_alarm (char c);

#endif /* STATE_H_ */
```

VII. startup c file

```c
/*
 * startup.c
 *
 *  Created on: Jun 25, 2022
 *      Author: momen amr
 */

#include<stdint.h>

extern void Pressure_controller(void);
extern unsigned int _stack_top;
extern unsigned int _S_data;
extern unsigned int _E_data;
extern unsigned int _S_bss;
extern unsigned int _E_bss;
extern unsigned int _E_text;


void reset_handler(void)
{
	unsigned int data_size = (unsigned int *)&_E_data - (unsigned int *)&_S_data;
	unsigned int* P_src = (unsigned int *)&_E_text;
	unsigned int* P_dst = (unsigned int *)&_S_data;

	for(int i=0; i<data_size; i++)
	{
		*P_dst++ = *P_src++;
	}

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

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

	Pressure_controller();
}

void default_handler(void)
{
	reset_handler;
}

void NMI_handler(void) __attribute__ ((weak,alias ("default_handler")));;
void H_FAULT_handler(void) __attribute__ ((weak,alias ("default_handler")));;
void MM_FAULT_handler(void) __attribute__ ((weak,alias ("default_handler")));;
void BUS_FAULT_handler(void) __attribute__ ((weak,alias ("default_handler")));;
void USAGE_FAULT_handler(void) __attribute__ ((weak,alias ("default_handler")));;

uint32_t vectors[] __attribute__ ((section(".vectors")))=
{(uint32_t) & _stack_top,
 (uint32_t) & reset_handler,
 (uint32_t) & NMI_handler,
 (uint32_t) & H_FAULT_handler,
 (uint32_t) & MM_FAULT_handler,
 (uint32_t) & BUS_FAULT_handler,
 (uint32_t) & USAGE_FAULT_handler};
```

VII. linker script file

```
/*
 *  linker_script.ld
 *
 *  Created on: Jun 25, 2022
 *      Author: momen amr
 */

MEMORY
{
      Flash(rx) : ORIGIN = 0x08000000, LENGTH = 128K
      Sram(rwx) : ORIGIN = 0x20000000, LENGTH = 20K
}

SECTIONS
{
      .text :
      {
      *(.vectors*)
      *(.text*)
      *(.rodata)
      . = ALIGN(4);
      _E_text = .;
      }> Flash

      .data :
      {
      _S_data = .;
      *(.data)
      . = ALIGN(4);
      _E_data = .;
      }> Sram AT> Flash

      .bss :
      {
      _S_bss = .;
      *(.bss)
      *(COMMON)
      . = ALIGN(4);
      _E_bss = .;
      . = . + 0X1000;
      _stack_top = .;
      }> Sram

}
```

VIII. make file

```makefile
CC=arm-none-eabi-
CFLAGS=-mcpu=cortex-m3 -gdwarf-2
INCS=-I .
LIBS=
SRC=$(wildcard *.c)
COBJ=$(SRC:.c=.o)
project=pressure_controller

$(project).bin: $(project).elf
	$(CC)objcopy.exe -O binary $< $@
	@echo "========Build is Done========"

$(project).elf: $(ASOBJ) $(COBJ)
	$(CC)ld.exe -T linker_script.ld $(ASOBJ) $(COBJ) -o $@ -Map=Map_file.map

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

clean:
	rm *.o *.elf *.bin
	@echo "========Clean is Done========"
```

9.software analysis

I. map file

```
Allocating common symbols
Common symbol       size              file

HPdetector_STATE    0x4               High_pressure_detector.o
state_Alarm_actuator_id
                    0x1               Alarm_actuator.o
Psensor_val         0x4               High_pressure_detector.o
Pval                0x4               Pressure_sensor.o
Alarm_actuator_STATE
                    0x4               Alarm_actuator.o
Psensor_STATE       0x4               Pressure_controller.o
Alarm_driver_STATE  0x4               Alarm_driver.o
state_Alarm_driver_id
                    0x1               Alarm_actuator.o
state_HPdetector_id
                    0x1               Alarm_actuator.o
state_Psensor_id    0x1               Alarm_actuator.o

Memory Configuration

Name                Origin             Length             Attributes
Flash               0x08000000         0x00020000         xr
Sram                0x20000000         0x00005000         xrw
*default*           0x00000000         0xffffffff

Linker script and memory map


.text               0x08000000      0x428
 *(.vectors*)
 .vectors           0x08000000       0x1c startup.o
                    0x08000000                vectors
 *(.text*)
 .text              0x0800001c       0xd4 Alarm_actuator.o
                    0x0800001c                AD_set_alarm
                    0x08000058                ST_AA_init
                    0x08000088                ST_AA_waiting
                    0x080000a0                ST_AA_ON
                    0x080000c8                ST_AA_OFF
 .text              0x080000f0       0x94 Alarm_driver.o
                    0x080000f0                High_pressure_detected
                    0x08000118                ST_AD_stop
                    0x08000130                ST_AD_start
                    0x08000158                ST_AD_waiting
 .text              0x08000184       0xc4 driver.o
                    0x08000184                Delay
                    0x080001a4                getPressureVal
                    0x080001bc                Set_Alarm_actuator
                    0x080001f8                GPIO_INITIALIZATION
 .text              0x08000248       0x68 High_pressure_detector.o
                    0x08000248                HPD_aquire_value
                    0x0800027c                ST_HPD_calculating
 .text              0x080002b0       0x58 Pressure_controller.o
                    0x080002b0                setup
                    0x080002d4                Pressure_controller
 .text              0x08000308       0x8c Pressure_sensor.o
                    0x08000308                ST_PS_init
                    0x0800032c                ST_PS_busy
```

```
                0x08000364                ST_PS_idle
 .text          0x08000394       0x94 startup.o
                0x08000394                reset_handler
                0x0800041c                USAGE_FAULT_handler
                0x0800041c                H_FAULT_handler
                0x0800041c                BUS_FAULT_handler
                0x0800041c                default_handler
                0x0800041c                MM_FAULT_handler
                0x0800041c                NMI_handler
 *(.rodata)
                0x08000428                . = ALIGN (0x4)
                0x08000428                _E_text = .

.glue_7         0x08000428        0x0
 .glue_7        0x08000428        0x0 linker stubs

.glue_7t        0x08000428        0x0
 .glue_7t       0x08000428        0x0 linker stubs

.vfp11_veneer   0x08000428        0x0
 .vfp11_veneer  0x08000428        0x0 linker stubs

.v4_bx          0x08000428        0x0
 .v4_bx         0x08000428        0x0 linker stubs

.iplt           0x08000428        0x0
 .iplt          0x08000428        0x0 Alarm_actuator.o

.rel.dyn        0x08000428        0x0
 .rel.iplt      0x08000428        0x0 Alarm_actuator.o

.data           0x20000000        0x4 load address 0x08000428
                0x20000000                _S_data = .
 *(.data)
 .data          0x20000000        0x0 Alarm_actuator.o
 .data          0x20000000        0x0 Alarm_driver.o
 .data          0x20000000        0x0 driver.o
 .data          0x20000000        0x4 High_pressure_detector.o
                0x20000000                HPD_threshold
 .data          0x20000004        0x0 Pressure_controller.o
 .data          0x20000004        0x0 Pressure_sensor.o
 .data          0x20000004        0x0 startup.o
                0x20000004                . = ALIGN (0x4)
                0x20000004                _E_data = .

.igot.plt       0x20000004        0x0 load address 0x0800042c
 .igot.plt      0x20000004        0x0 Alarm_actuator.o

.bss            0x20000004     0x1020 load address 0x0800042c
                0x20000004                _S_bss = .
 *(.bss)
 .bss           0x20000004        0x0 Alarm_actuator.o
 .bss           0x20000004        0x0 Alarm_driver.o
 .bss           0x20000004        0x0 driver.o
 .bss           0x20000004        0x0 High_pressure_detector.o
 .bss           0x20000004        0x0 Pressure_controller.o
 .bss           0x20000004        0x0 Pressure_sensor.o
 .bss           0x20000004        0x0 startup.o
 *(COMMON)
 COMMON         0x20000004        0xb Alarm_actuator.o
                0x20000004                state_Alarm_actuator_id
                0x20000008                Alarm_actuator_STATE
                0x2000000c                state_Alarm_driver_id
                0x2000000d                state_HPdetector_id
                0x2000000e                state_Psensor_id
 *fill*         0x2000000f        0x1
 COMMON         0x20000010        0x4 Alarm_driver.o
                0x20000010                Alarm_driver_STATE
 COMMON         0x20000014        0x8 High_pressure_detector.o
                0x20000014                HPdetector_STATE
                0x20000018                Psensor_val
 COMMON         0x2000001c        0x4 Pressure_controller.o
                0x2000001c                Psensor_STATE
 COMMON         0x20000020        0x4 Pressure_sensor.o
                0x20000020                Pval
                0x20000024                . = ALIGN (0x4)
                0x20000024                _E_bss = .
                0x20001024                . = (. + 0x1000)
 *fill*         0x20000024     0x1000
                0x20001024                _stack_top = .
LOAD Alarm_actuator.o
LOAD Alarm_driver.o
LOAD driver.o
LOAD High_pressure_detector.o
LOAD Pressure_controller.o
LOAD Pressure_sensor.o
LOAD startup.o
OUTPUT(pressure_controller.elf elf32-littlearm)

.debug_info     0x00000000     0x417b
 .debug_info    0x00000000      0xae0 Alarm_actuator.o
 .debug_info    0x00000ae0      0xabd Alarm_driver.o
 .debug_info    0x0000159d      0xa05 driver.o
 .debug_info    0x00001fa2      0xac5 High_pressure_detector.o
 .debug_info    0x00002a67      0xac9 Pressure_controller.o
 .debug_info    0x00003530      0xaba Pressure_sensor.o
 .debug_info    0x00003fea      0x191 startup.o

.debug_abbrev   0x00000000      0xbfc
 .debug_abbrev  0x00000000      0x1fb Alarm_actuator.o
 .debug_abbrev  0x000001fb      0x1d4 Alarm_driver.o
 .debug_abbrev  0x000003cf      0x1de driver.o
 .debug_abbrev  0x000005ad      0x1e5 High_pressure_detector.o
 .debug_abbrev  0x00000792      0x1be Pressure_controller.o
 .debug_abbrev  0x00000950      0x1d4 Pressure_sensor.o
 .debug_abbrev  0x00000b24       0xd8 startup.o

.debug_loc      0x00000000      0x554
 .debug_loc     0x00000000      0x124 Alarm_actuator.o
 .debug_loc     0x00000124       0xc8 Alarm_driver.o
 .debug_loc     0x000001ec      0x140 driver.o
 .debug_loc     0x0000032c       0x88 High_pressure_detector.o
 .debug_loc     0x000003b4       0x58 Pressure_controller.o
 .debug_loc     0x0000040c       0xb4 Pressure_sensor.o
 .debug_loc     0x000004c0       0x94 startup.o

.debug_aranges  0x00000000       0xe0
 .debug_aranges
                0x00000000       0x20 Alarm_actuator.o
 .debug_aranges
                0x00000020       0x20 Alarm_driver.o
 .debug_aranges
                0x00000040       0x20 driver.o
 .debug_aranges
                0x00000060       0x20 High_pressure_detector.o
 .debug_aranges
                0x00000080       0x20 Pressure_controller.o
 .debug_aranges
```

```
                0x000000a0       0x20 Pressure_sensor.o
 .debug_aranges
                0x000000c0       0x20 startup.o

.debug_line     0x00000000      0xc12
 .debug_line    0x00000000      0x1cf Alarm_actuator.o
 .debug_line    0x000001cf      0x1b9 Alarm_driver.o
 .debug_line    0x00000388      0x1bb driver.o
 .debug_line    0x00000543      0x1ca High_pressure_detector.o
 .debug_line    0x0000070d      0x20e Pressure_controller.o
 .debug_line    0x0000091b      0x1bc Pressure_sensor.o
 .debug_line    0x00000ad7      0x13b startup.o

.debug_str      0x00000000      0x79a
 .debug_str     0x00000000      0x590 Alarm_actuator.o
                                0x658 (size before relaxing)
 .debug_str     0x00000590       0x5e Alarm_driver.o
                                0x657 (size before relaxing)
 .debug_str     0x000005ee       0x4e driver.o
                                0x58b (size before relaxing)
 .debug_str     0x0000063c       0x68 High_pressure_detector.o
                                0x661 (size before relaxing)
 .debug_str     0x000006a4       0x3e Pressure_controller.o
                                0x670 (size before relaxing)
 .debug_str     0x000006e2       0x38 Pressure_sensor.o
                                0x63f (size before relaxing)
 .debug_str     0x0000071a       0x80 startup.o
                                0x1d2 (size before relaxing)

.comment        0x00000000       0x7e
 .comment       0x00000000       0x7e Alarm_actuator.o
                                 0x7f (size before relaxing)
 .comment       0x0000007e       0x7f Alarm_driver.o
 .comment       0x0000007e       0x7f driver.o
 .comment       0x0000007e       0x7f High_pressure_detector.o
 .comment       0x0000007e       0x7f Pressure_controller.o
 .comment       0x0000007e       0x7f Pressure_sensor.o
 .comment       0x0000007e       0x7f startup.o

.ARM.attributes
                0x00000000       0x33
 .ARM.attributes
                0x00000000       0x33 Alarm_actuator.o
 .ARM.attributes
                0x00000033       0x33 Alarm_driver.o
 .ARM.attributes
                0x00000066       0x33 driver.o
 .ARM.attributes
                0x00000099       0x33 High_pressure_detector.o
 .ARM.attributes
                0x000000cc       0x33 Pressure_controller.o
 .ARM.attributes
                0x000000ff       0x33 Pressure_sensor.o
 .ARM.attributes
                0x00000132       0x33 startup.o

.debug_frame    0x00000000      0x32c
 .debug_frame   0x00000000       0xac Alarm_actuator.o
 .debug_frame   0x000000ac       0x84 Alarm_driver.o
 .debug_frame   0x00000130       0xa0 driver.o
 .debug_frame   0x000001d0       0x54 High_pressure_detector.o
 .debug_frame   0x00000224       0x48 Pressure_controller.o
 .debug_frame   0x0000026c       0x6c Pressure_sensor.o
 .debug_frame   0x000002d8       0x54 startup.o
```

II. sections file

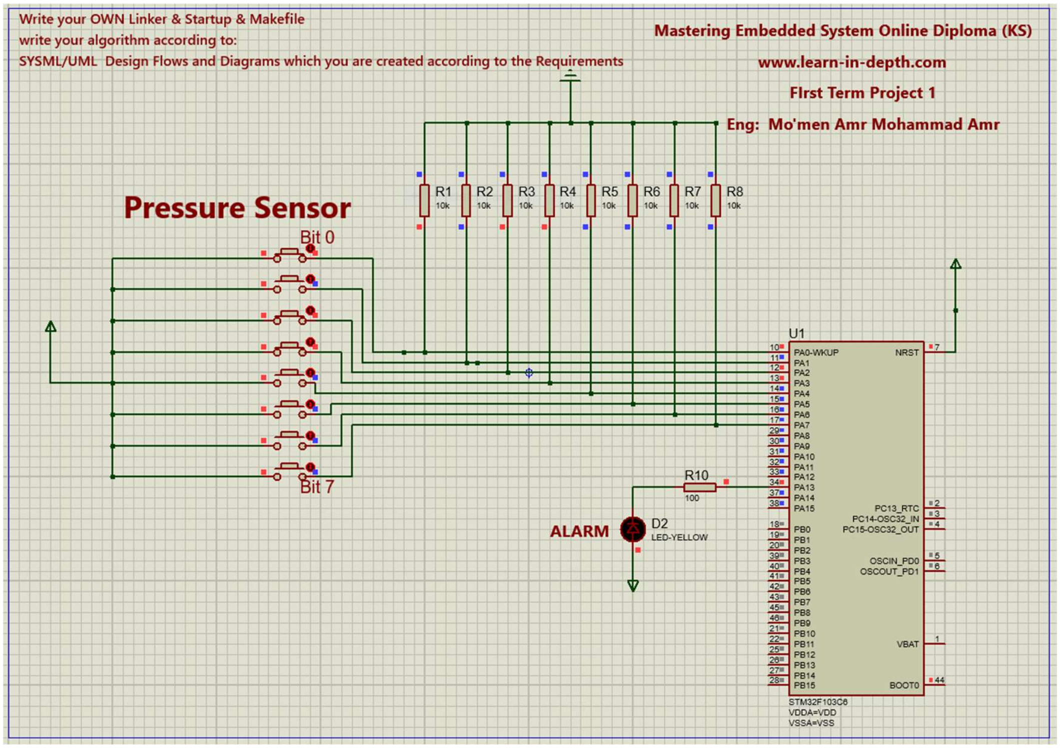
```
$ arm-none-eabi-objdump.exe -h pressure_controller.elf

pressure_controller.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000428  08000000  08000000  00010000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000004  20000000  08000428  00020000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00001020  20000004  0800042c  00020004  2**2
                  ALLOC
  3 .debug_info   0000417b  00000000  00000000  00020004  2**0
                  CONTENTS, READONLY, DEBUGGING
  4 .debug_abbrev 00000bfc  00000000  00000000  0002417f  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    00000554  00000000  00000000  00024d7b  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 000000e0  00000000  00000000  000252cf  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_line   00000c12  00000000  00000000  000253af  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .debug_str    0000079a  00000000  00000000  00025fc1  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      0000007e  00000000  00000000  0002675b  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  000267d9  2**0
                  CONTENTS, READONLY
 11 .debug_frame  0000032c  00000000  00000000  0002680c  2**2
                  CONTENTS, READONLY, DEBUGGING
```

III. symbols file

```
$ arm-none-eabi-nm.exe pressure_controller.elf
20000024 B _E_bss
20000004 D _E_data
08000428 T _E_text
20000004 B _S_bss
20000000 D _S_data
20001024 B _stack_top
0800001c T AD_set_alarm
20000008 B Alarm_actuator_STATE
20000010 B Alarm_driver_STATE
0800041c W BUS_FAULT_handler
0800041c T default_handler
08000184 T Delay
080001a4 T getPressureVal
080001f8 T GPIO_INITIALIZATION
0800041c W H_FAULT_handler
080000f0 T High_pressure_detected
08000248 T HPD_aquire_value
20000000 D HPD_threshold
20000014 B HPdetector_STATE
0800041c W MM_FAULT_handler
0800041c W NMI_handler
080002d4 T Pressure_controller
2000001c B Psensor_STATE
20000018 B Psensor_val
20000020 B Pval
08000394 T reset_handler
080001bc T Set_Alarm_actuator
080002b0 T setup
08000058 T ST_AA_init
080000c8 T ST_AA_OFF
080000a0 T ST_AA_ON
08000088 T ST_AA_waiting
08000130 T ST_AD_start
08000118 T ST_AD_stop
08000158 T ST_AD_waiting
0800027c T ST_HPD_calculating
0800032c T ST_PS_busy
08000364 T ST_PS_idle
08000308 T ST_PS_init
20000004 B state_Alarm_actuator_id
2000000c B state_Alarm_driver_id
2000000d B state_HPdetector_id
2000000e B state_Psensor_id
0800041c W USAGE_FAULT_handler
08000000 T vectors
```

10. Proteus simulation results

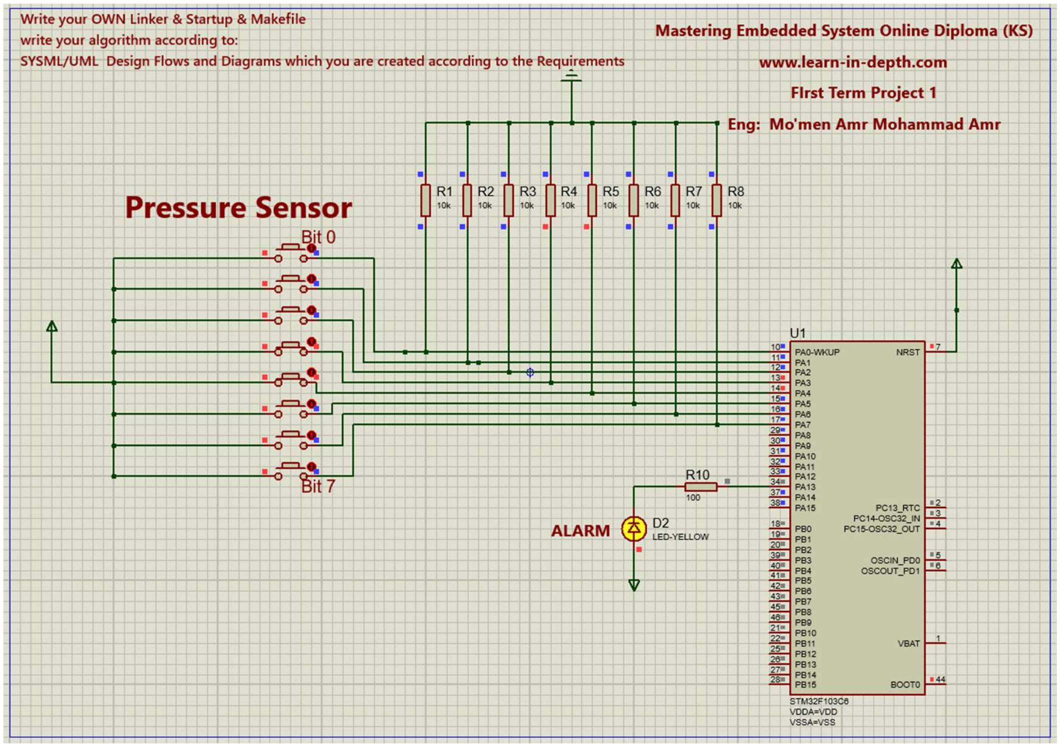
When we input Pval =13 alarm doesn't work



Write your OWN Linker & Startup & Makefile
write your algorithm according to:
SYSML/UML Design Flows and Diagrams which you are created according to the Requirements

Mastering Embedded System Online Diploma (KS)
www.learn-in-depth.com
FIrst Term Project 1
Eng: Mo'men Amr Mohammad Amr

When we input Pval =24 alarm works for 60 seconds



Write your OWN Linker & Startup & Makefile
write your algorithm according to:
SYSML/UML Design Flows and Diagrams which you are created according to the Requirements

Mastering Embedded System Online Diploma (KS)
www.learn-in-depth.com
FIrst Term Project 1
Eng: Mo'men Amr Mohammad Amr