Master Embedded System

Learn-in-depth.

Pressure Controller Project Report

By: Ibrahim Abo Elhassan

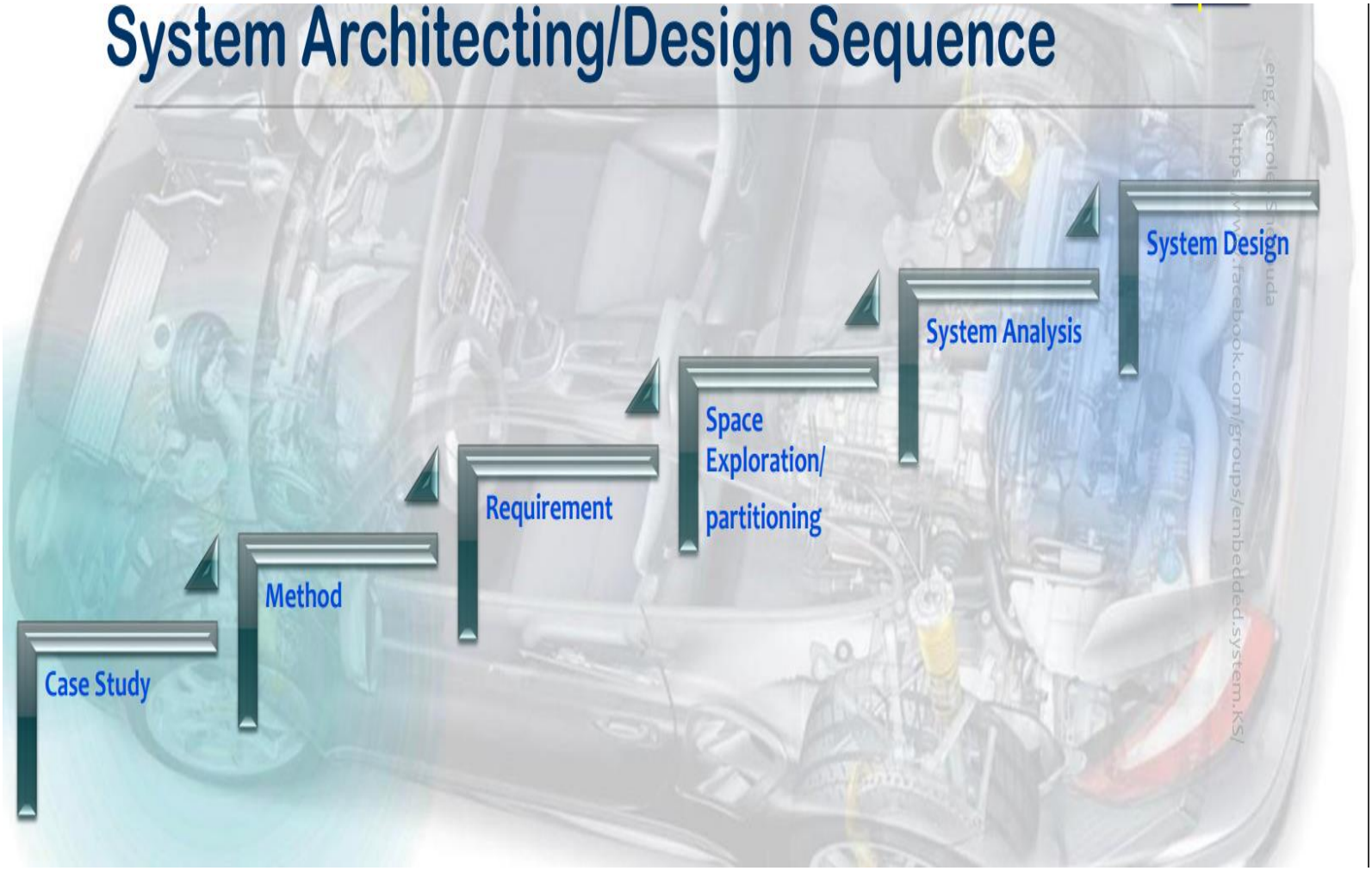
Report content

- Case Study
- Methodology
- Requirement Diagram
- Space Exploration
- System Analysis:

  1- Use Case Diagram

  2- Activity Diagram

  3- Sequence Diagram
- System Design
- System Design Simulation
- Code
- Map file
- Symbol table
- Simulation on Hardware “Proteus”.


System Architecting/Design Sequence
Case Study
Method
Requirement
Space Exploration/ partitioning
System Analysis
System Design

- Case Study

A pressure controller informs the crew of a cabin with an alarm when the pressure exceeds 20 bars in the cabin.

The alarm duration equals 60 seconds.

keeps track of the measured values.

Assumptions

The controller set up and shutdown procedures are not modeled.

The controller maintenance is not modeled.

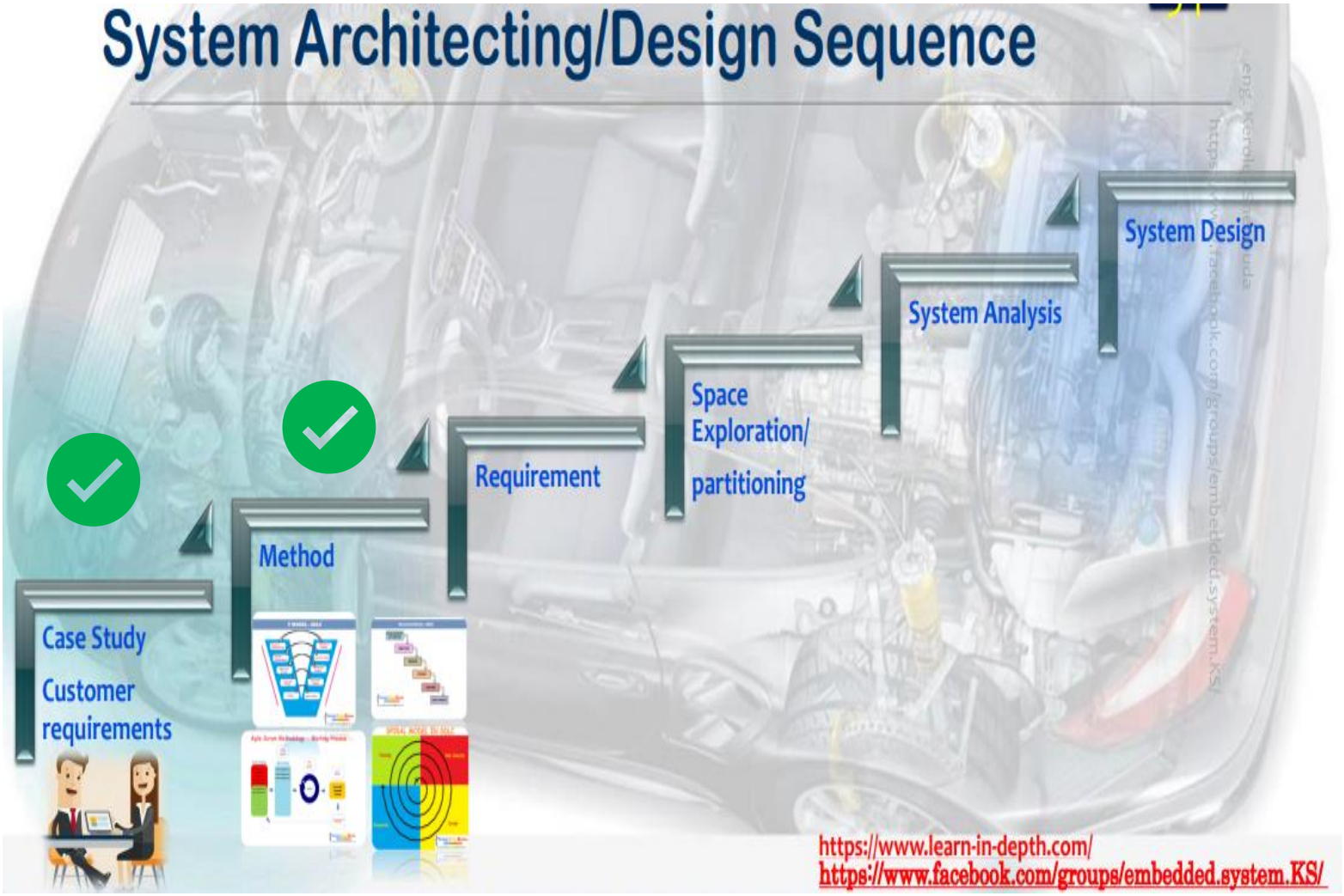
The pressure sensor never fails.

The alarm never fails.

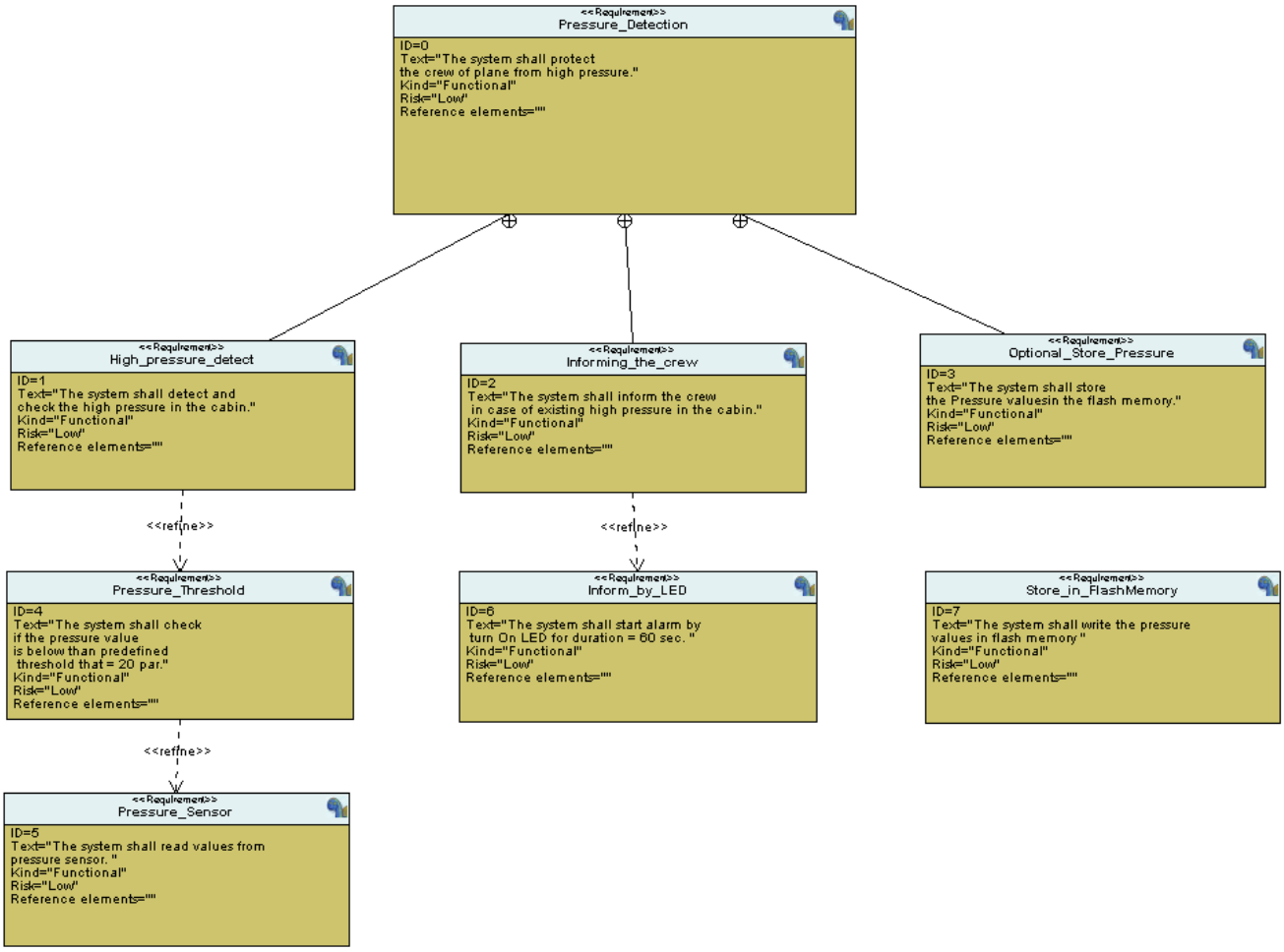
The controller never faces power cut.

- Methodology

I will use V-Model testing in implementation of this system.

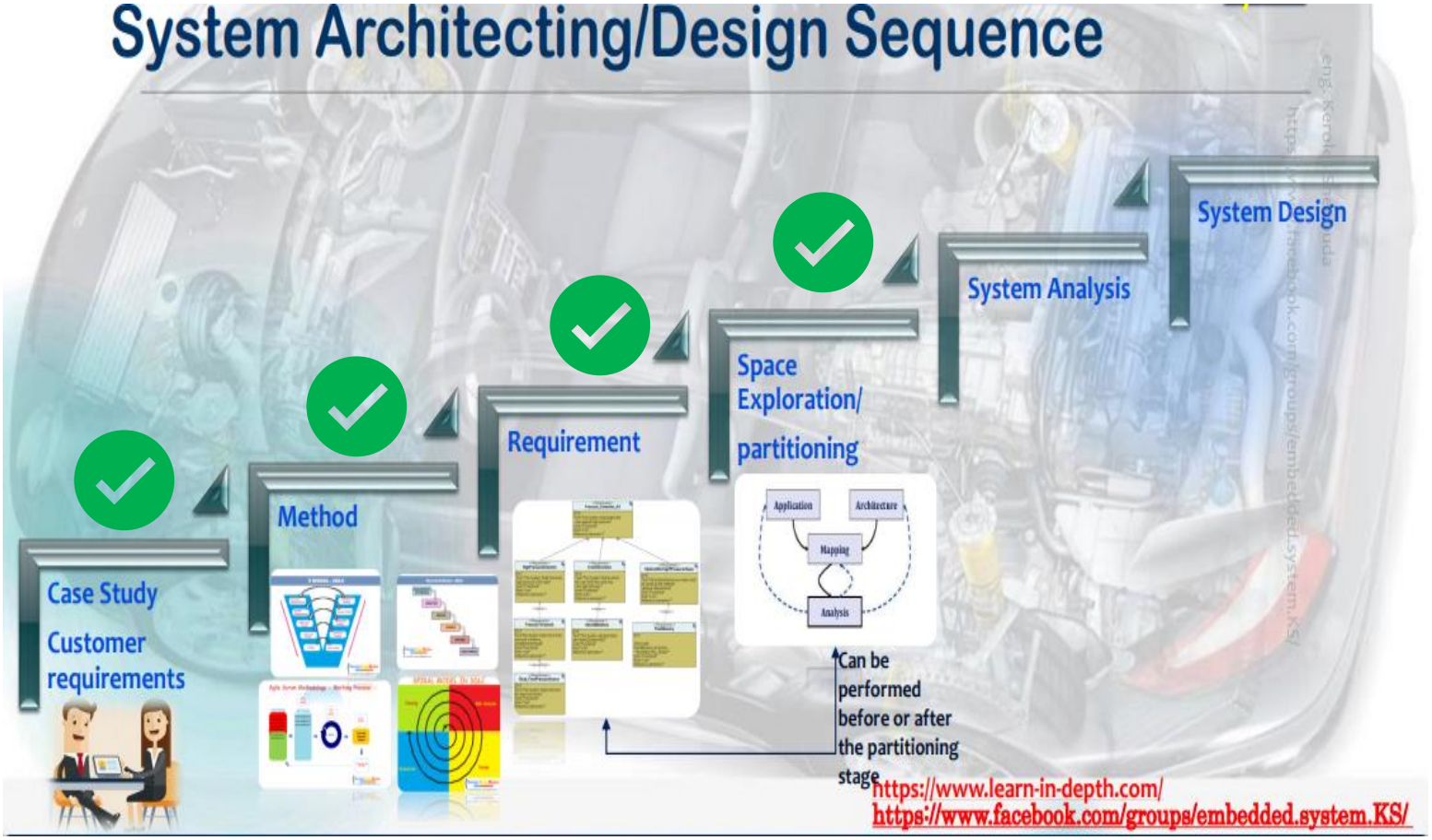

System Architecting/Design Sequence
Case Study
Customer requirements
Method
Requirement
Space Exploration/ partitioning
System Analysis
System Design
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

- Requirement Diagram



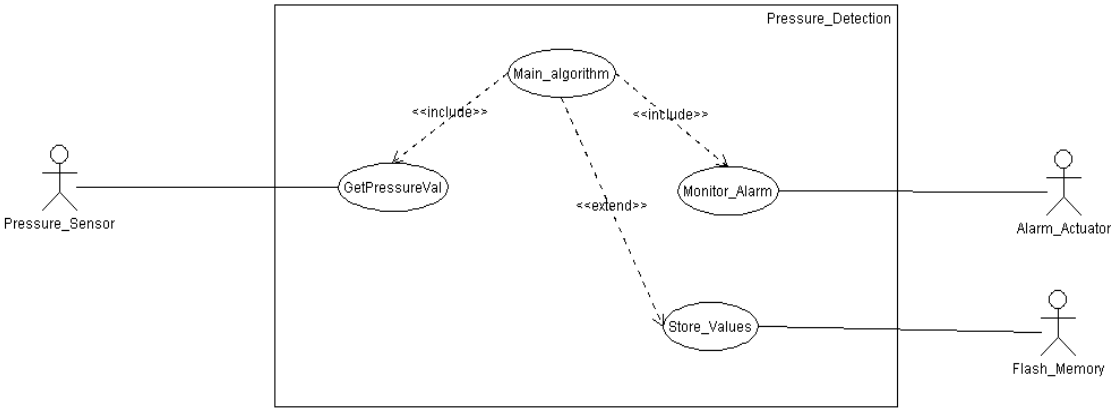
- Space Exploration

I will use STM32 microcontroller with cortex-m3 processor.

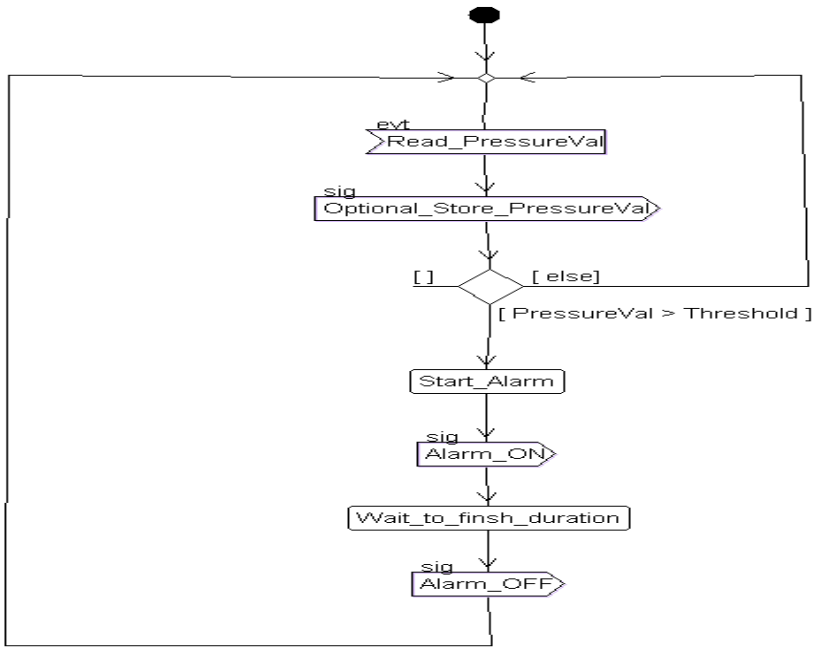

System Architecting/Design Sequence
Case Study
Customer requirements
Method
Requirement
Space Exploration/ partitioning
System Analysis
System Design
Application
Architecture
Mapping
Analysis
Can be performed before or after the partitioning stage
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

- System Analysis

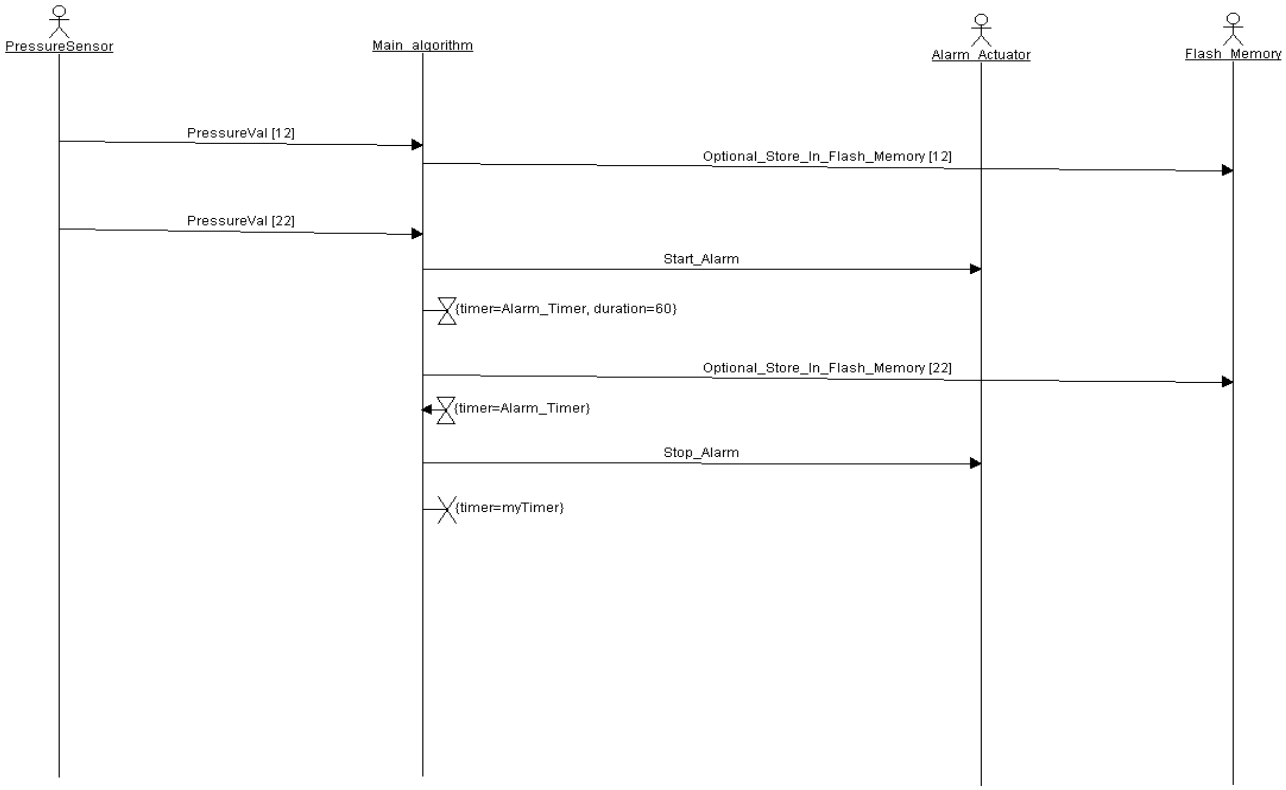
1- Use Case Diagram

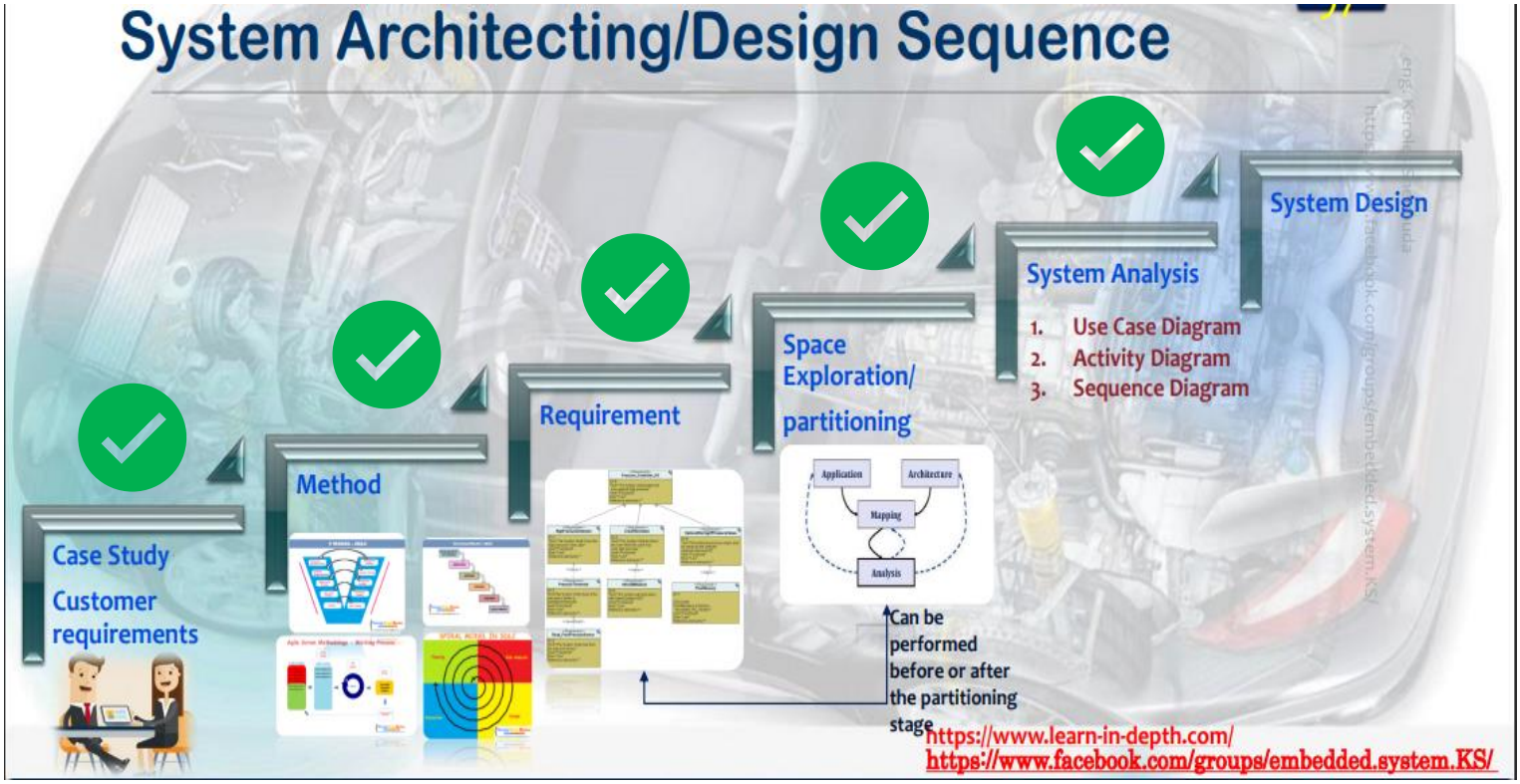


2- Activity Diagram

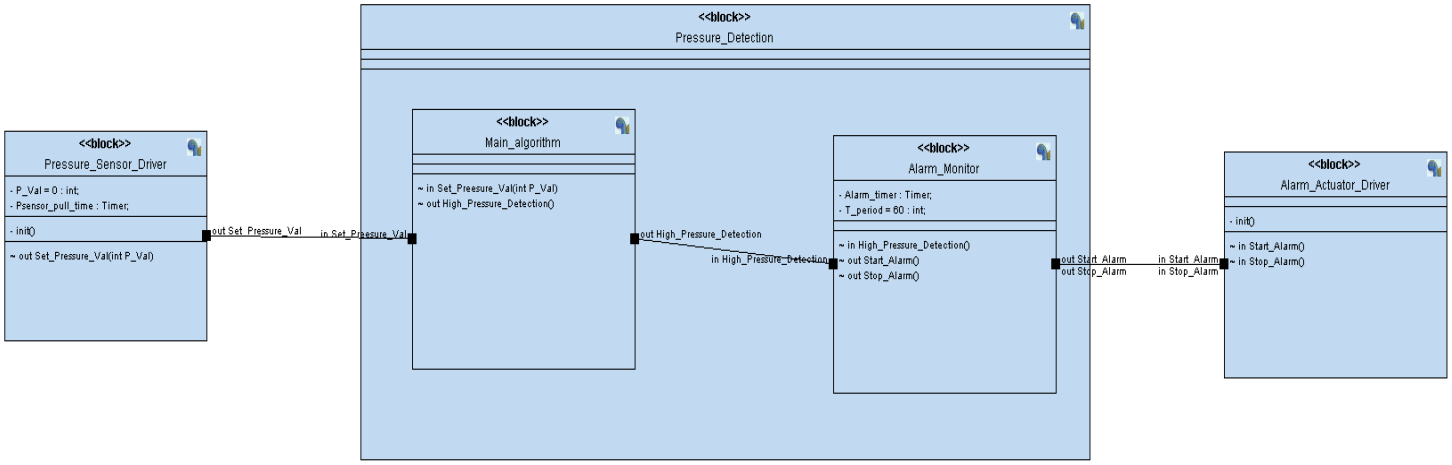


3- Sequence Diagram

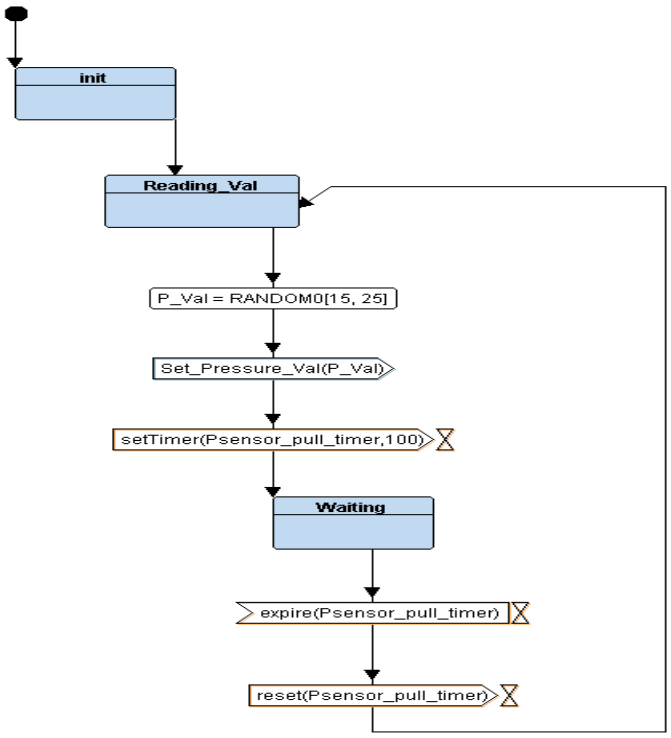



System Architecting/Design Sequence
Case Study
Customer requirements
Method
Requirement
Space Exploration/ partitioning
Application
Architecture
Mapping
Analysis
Can be performed before or after the partitioning stage
System Analysis
1. Use Case Diagram
2. Activity Diagram
3. Sequence Diagram
System Design
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

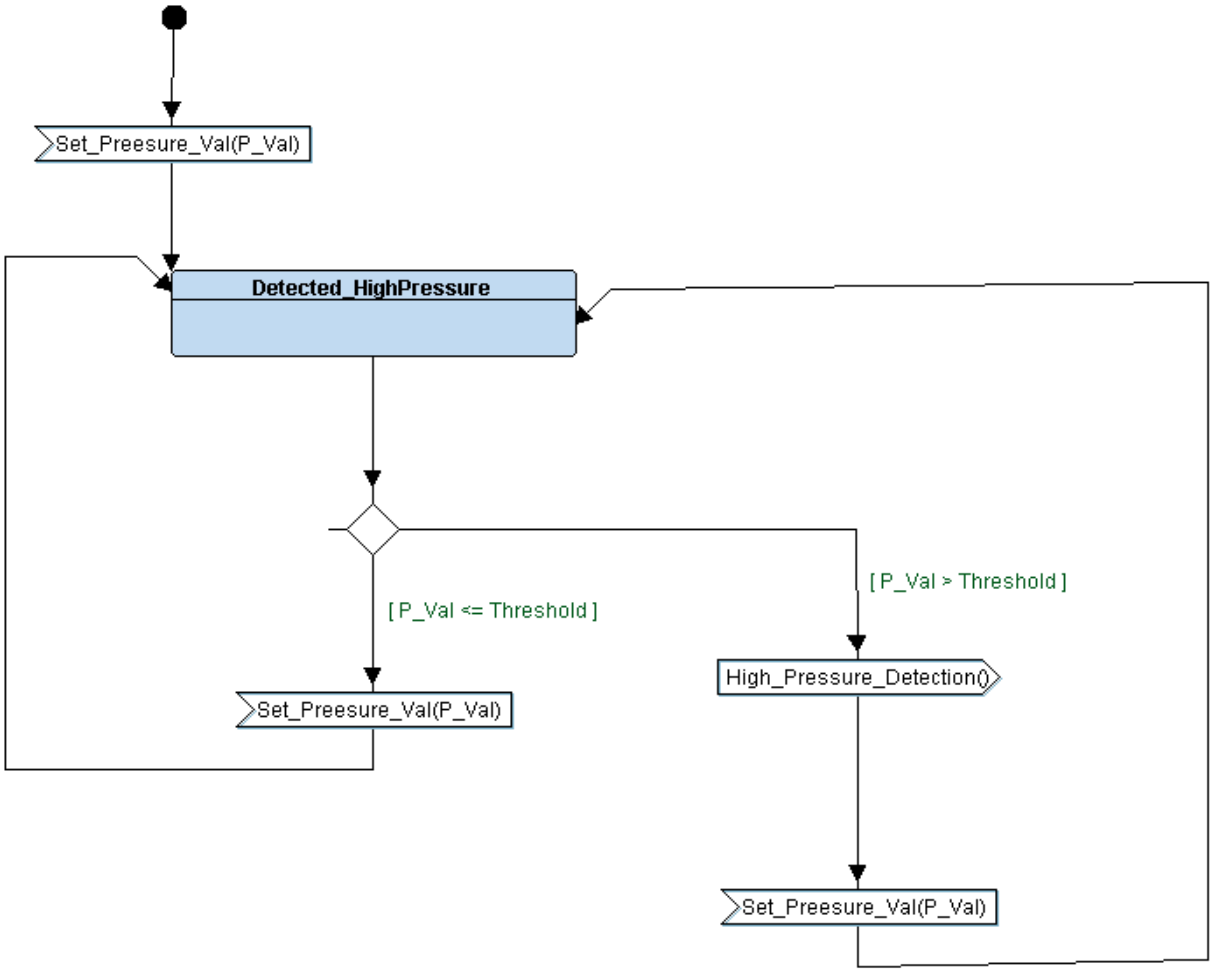
- System Design



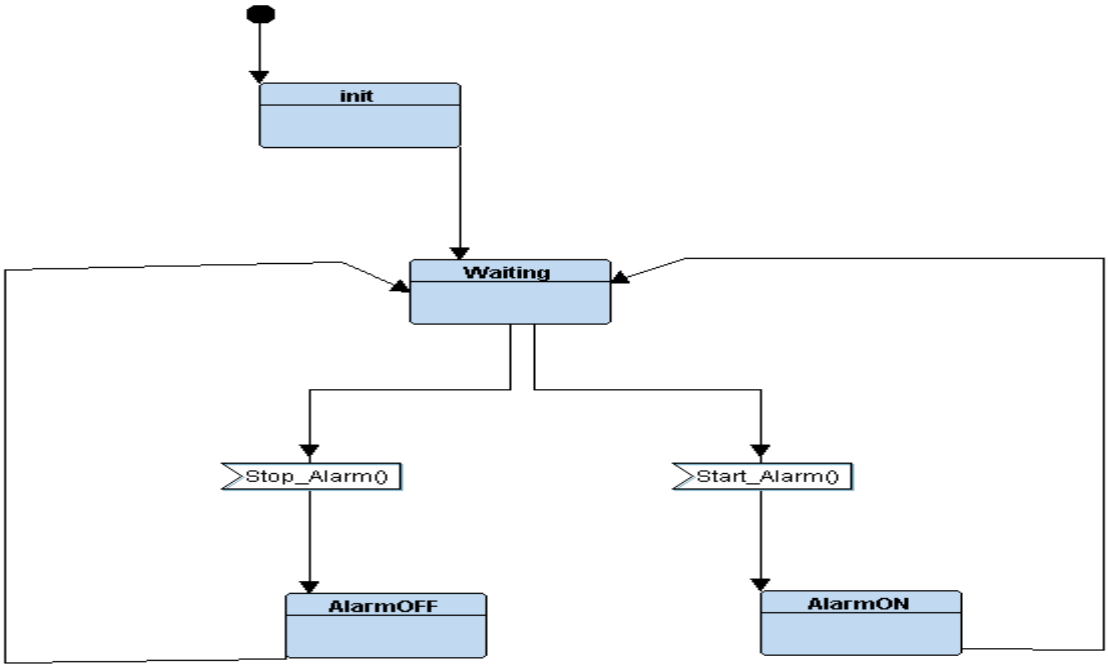
1- Pressure Sensor Driver state diagram



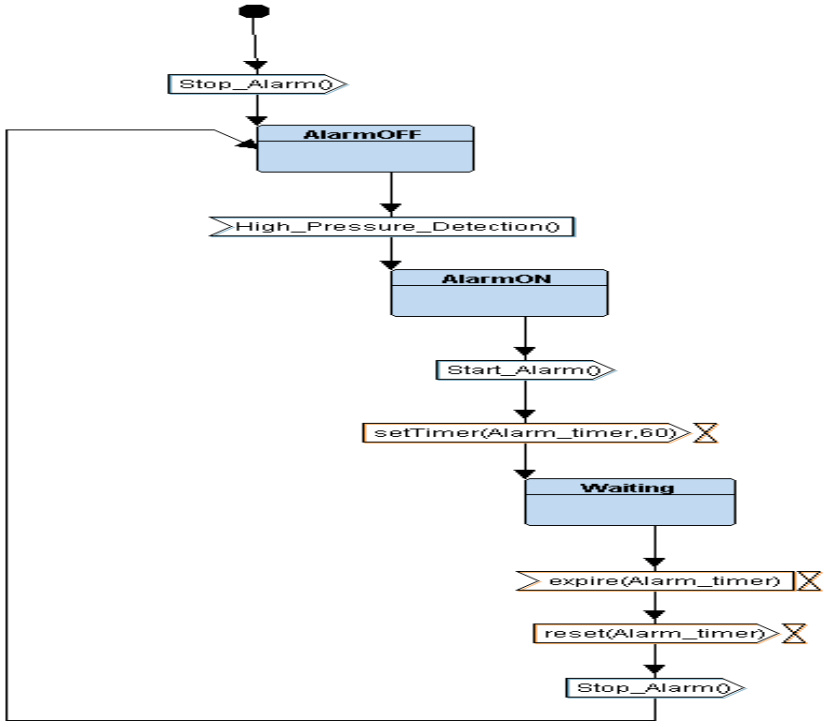
2- Main Algorithm state diagram



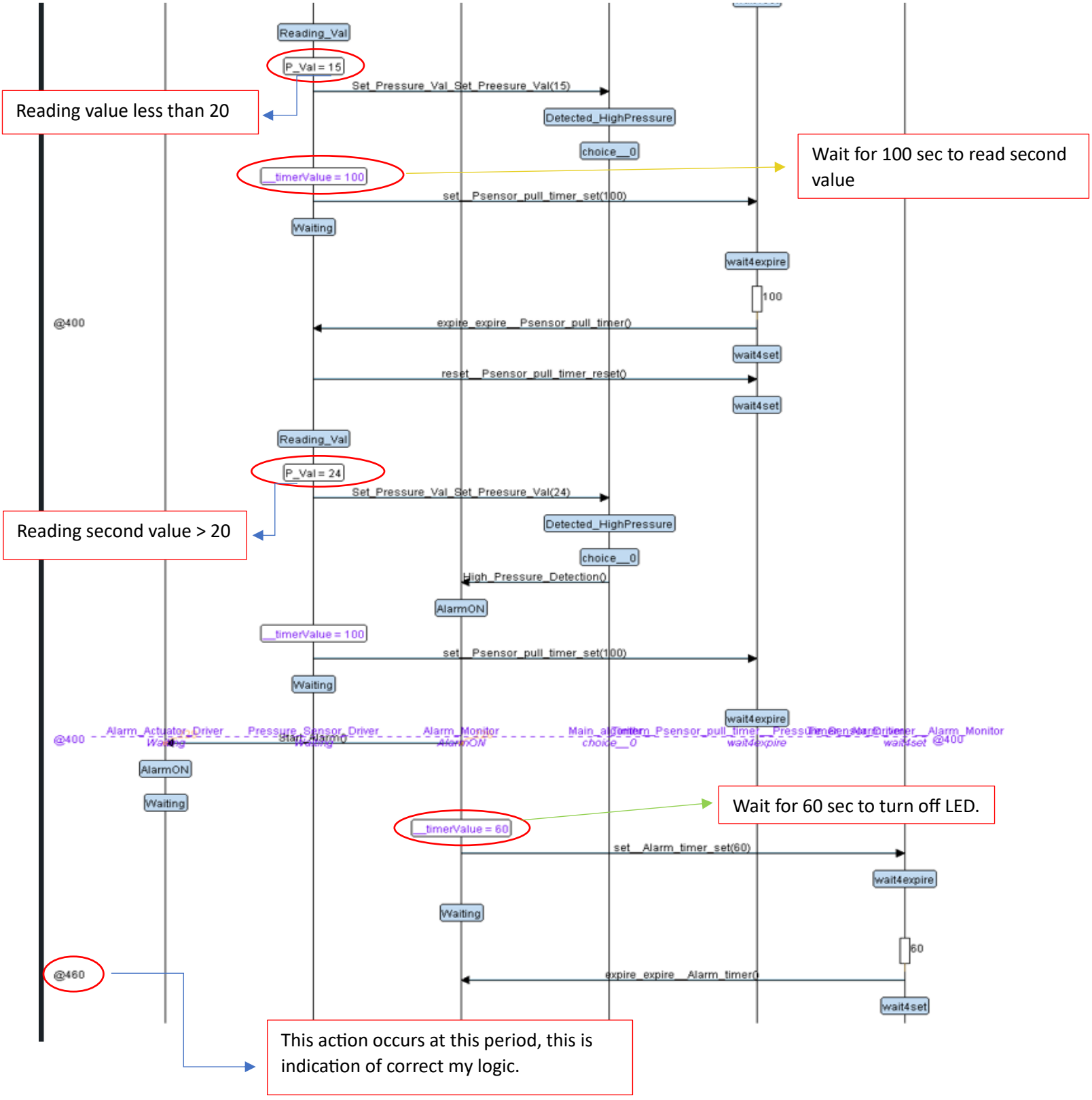
3-Alarm Actuator Driver state diagram

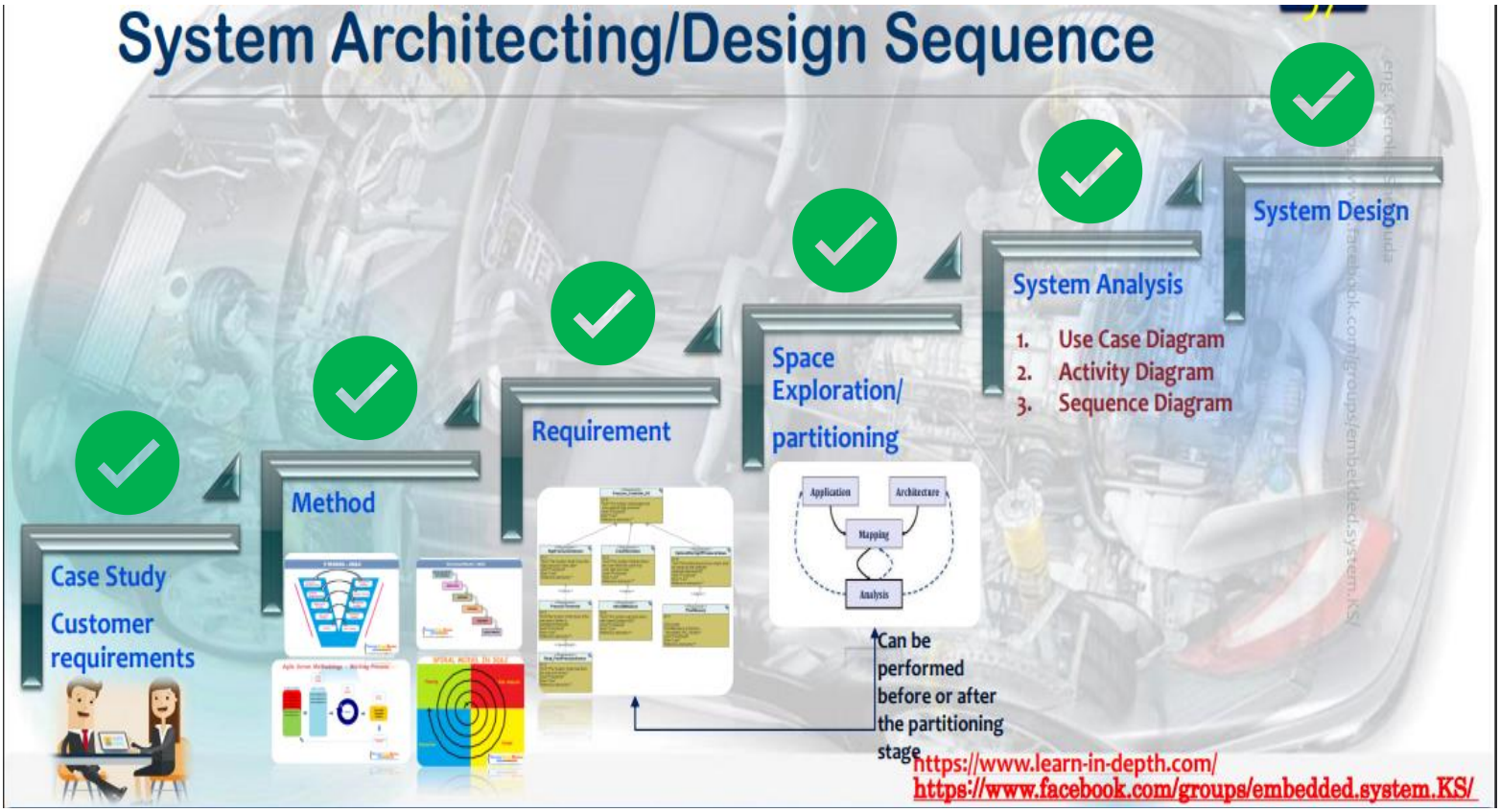


4- Alarm Monitor state diagram



- System Design Simulation




System Architecting/Design Sequence
Case Study
Customer requirements
Method
Requirement
Space Exploration/ partitioning
Application
Architecture
Mapping
Analysis
Can be performed before or after the partitioning stage
System Analysis
1. Use Case Diagram
2. Activity Diagram
3. Sequence Diagram
System Design
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

- **Code**

- Main.c

```c

/* main.c
*  By: Ibrahim Abo Elhassan
*/

#include <stdio.h>
#include <stdlib.h>

#include "Driver.h"
#include "P_SEN.h"
#include "Main_Alg.h"
#include "Alarm_Mon.h"
#include "Alarm_Actu.h"

void (*PS_state)() = STATE(PS_init);
void (*Aactu_state)() = STATE(Aactu_init);
void (*AM_state)() = STATE(AM_alarmOFF);
void (*MA_state)() = STATE(MA_High_Pressure);

int main()
{
    GPIO_INITIALIZATION();

    while(1)
    {
        PS_state();
        Aactu_state();
        AM_state();
        MA_state();

    }

    return 0;
}
```

- State.h

```c
/* state.h
*  By: Ibrahim Abo Elhassan
*/
#ifndef STATE_H_
#define STATE_H_

#include "Driver.h"

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

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

// Connections States

// Pressure Sensor =======> Main Algorithm
int PS_catch_Pressure_Val();

// Main Algorithm =======> Alarm Monitor
int MA_High_Pressure_Detect();

// Alarm Monitor =======> Alarm Actuator
void Aactu_Start_Alarm();

// Alarm Monitor =======> Alarm Actuator
void Aactu_Stop_Alarm();


#endif // STATE_H_
```

- Driver.c

```c

#include "driver.h"

#include <stdint.h>
#include <stdio.h>

#include "Driver.h"

void GPIO_Delay(int nCount)
{
    for(; nCount != 0; nCount--);
}

int GPIO_getPressureVal()
{
    return (GPIOA_IDR & 0xFF);
}

void GPIO_Set_Alarm_actuator(int i)
{
    if (i == 1){
        SET_BIT(GPIOA_ODR,13);
    }
    else if (i == 0){
        RESET_BIT(GPIOA_ODR,13);
    }
}

void GPIO_INITIALIZATION ()
{
    SET_BIT(APB2ENR, 2);
    GPIOA_CRL &= 0xFF0FFFFF;
    GPIOA_CRL |= 0x00000000;
    GPIOA_CRH &= 0xFF0FFFFF;
    GPIOA_CRH |= 0x22222222;
}
```

Driver.h

```c


#ifndef Driver_H_
#define Driver_H_


#define SET_BIT(ADDRESS,BIT)   ADDRESS |=  (1<<BIT)
#define RESET_BIT(ADDRESS,BIT) ADDRESS &= ~(1<<BIT)
#define TOGGLE_BIT(ADDRESS,BIT)  ADDRESS ^=  (1<<BIT)
#define READ_BIT(ADDRESS,BIT) ((ADDRESS) &   (1<<(BIT)))


#define GPIO_PORTA 0x40010800
#define BASE_RCC   0x40021000

#define APB2ENR   *(volatile uint32_t *)(BASE_RCC + 0x18)

#define GPIOA_CRL *(volatile uint32_t *)(GPIO_PORTA + 0x00)
#define GPIOA_CRH *(volatile uint32_t *)(GPIO_PORTA + 0X04)
#define GPIOA_IDR *(volatile uint32_t *)(GPIO_PORTA + 0x08)
#define GPIOA_ODR *(volatile uint32_t *)(GPIO_PORTA + 0x0C)


void GPIO_Delay(int nCount);
int  GPIO_getPressureVal();
void GPIO_Set_Alarm_actuator(int i);
void GPIO_INITIALIZATION ();

#endif // Driver_H_
```

- Pressure Sensor.c

```c
#include "P_SEN.h"

static int P_Val = 0;

enum{
    PS_init,
    PS_Reading,
    PS_waiting,
}Status_PS;

extern void (*PS_state)();

STATE_define(PS_init)
{
    Status_PS = PS_init;

    PS_state = STATE(PS_Reading);
}

STATE_define(PS_Reading)
{
    Status_PS = PS_Reading;

    P_Val = GPIO_getPressureVal();

    PS_state = STATE(PS_waiting);

}

STATE_define(PS_waiting)
{
    Status_PS = PS_waiting;

    GPIO_Delay(1000);

    PS_state = STATE(PS_Reading);
}

int PS_catch_Pressure_Val()
{
    return P_Val;
}
```

Pressure Sensor.h

```c
#ifndef P_SEN_H_
#define P_SEN_H_

#include "state.h"


STATE_define(PS_init);
STATE_define(PS_Reading);
STATE_define(PS_waiting);


#endif // P_SEN_H_
```

- Main algorithm.c

```c
#include "Main_Alg.h"

enum {
    MA_High_Pressure
}MA_Status;

extern void (*MA_state)();

int MA_P_Val = 0;
int MA_Threshold = 20;

STATE_define(MA_High_Pressure)
{
    MA_Status = MA_High_Pressure;

    MA_P_Val = PS_catch_Pressure_Val();

    MA_state = STATE(MA_High_Pressure);
}

int MA_High_Pressure_Detect()
{
    return(MA_P_Val > MA_Threshold);
}
```

Main algorithm.h

```c

#ifndef MAIN_ALG_H_
#define MAIN_ALG_H_

#include"state.h"

STATE_define(MA_High_Pressure);

#endif // MAIN_ALG_H_
```

- Alarm monitor.c

```c
#include "Alarm_Mon.h"

enum{
    AM_alarmOFF,
    AM_alarmON,
    AM_waiting
}AM_Status;

extern void (*AM_state)();

STATE_define(AM_alarmOFF)
{
    AM_Status = AM_alarmOFF;

    Aactu_Stop_Alarm();

    if(MA_High_Pressure_Detect != 0)
    {
        AM_state = STATE(AM_alarmON);
    }

}
STATE_define(AM_alarmON)
{
    AM_Status = AM_alarmON;

    Aactu_Start_Alarm();

    AM_state = STATE(AM_waiting);

}
STATE_define(AM_waiting)
 {
     AM_Status = AM_waiting;

     GPIO_Delay(500);

     AM_state = STATE(AM_alarmOFF);

 }
```

Alarm monitor.h

```c

#ifndef _ALARM_MON_H_
#define _ALARM_MON_H_

#include "state.h"

STATE_define(AM_alarmON);
STATE_define(AM_alarmOFF);
STATE_define(AM_waiting);


#endif // _ALARM_MON_H_

```

- Alarm actuator.c

```c
#include "Alarm_Actu.h"

#define TRUE 1
#define FALSE 0

enum{
    Aactu_init,
    Aactu_waiting,
    Aactu_ACT_ON,
    Aactu_ACT_OFF
}A_Act_Status;

extern void (*Aactu_state)();

STATE_define(Aactu_init)
{
    A_Act_Status = Aactu_init;

    Aactu_state = STATE(Aactu_waiting);

}
STATE_define(Aactu_waiting)
{
    Aactu_state = STATE(Aactu_waiting);

}
STATE_define(Aactu_ACT_ON)
{
    A_Act_Status = Aactu_ACT_ON;

    GPIO_Set_Alarm_actuator(TRUE);

    Aactu_state = STATE(Aactu_waiting);
}
STATE_define(Aactu_ACT_OFF)
{
    A_Act_Status = Aactu_ACT_OFF;

    GPIO_Set_Alarm_actuator(FALSE);

    Aactu_state = STATE(Aactu_waiting);

}


void Aactu_Start_Alarm()
{
    Aactu_state = STATE(Aactu_ACT_ON);
}

void Aactu_Stop_Alarm()
{
    Aactu_state = STATE(Aactu_ACT_OFF);
}
```

Alarm actuator.h

```c
#ifndef ALARM_ACTU_H_
#define ALARM_ACTU_H_

#include "state.h"

STATE_define(Aactu_init);
STATE_define(Aactu_waiting);
STATE_define(Aactu_ACT_ON);
STATE_define(Aactu_ACT_OFF);


#endif // ALARM_ACTU_H_
```

- Linker script.ld

```
MEMORY
{
    flash(RX) : ORIGIN = 0x08000000, LENGTH = 128K
    sram(RWX) : ORIGIN = 0x20000000, LENGTH = 20K
}

SECTIONS
{
    .text :
    {
        *(.vectors*)
        *(.text*)
        *(.rodata*)
        _E_TEXT = . ; /* End of .text section*/
    }>flash

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

    .bss :
    {
        _S_BSS = . ;
        *(.bss*)
        . = ALIGN(4);
        _E_BSS = . ;

        . = ALIGN(4);
        . = . + 0x1000 ;
        _STACK_TOP = . ;
    }>sram
}
```

- Startup.c

```c
/*  startup.c
 *  CopyRight : ibrahim Abo Elhassan
 */


#include "Platform_Types.h"


extern uint32_t _STACK_TOP ;

extern int main(void);

void Reset_Hundler(void);

void Default_Hundler()
{
    Reset_Hundler();
}

void NMI_Handler(void)              __attribute__ ((weak, alias("Default_Hundler")));;
void H_Fault_Handler(void)          __attribute__ ((weak, alias("Default_Hundler")));;
void MM_Fault_Handler(void)         __attribute__ ((weak, alias("Default_Hundler")));;
void Bus_Fault(void)                __attribute__ ((weak, alias("Default_Hundler")));;
void Usage_Fault_Handler(void)      __attribute__ ((weak, alias("Default_Hundler")));;

uint32_t vectors[] __attribute__ ((section(".vectors"))) = {
    (uint32_t)  &_STACK_TOP,
    (uint32_t)  &Reset_Hundler,
    (uint32_t)  &NMI_Handler,
    (uint32_t)  &H_Fault_Handler,
    (uint32_t)  &MM_Fault_Handler,
    (uint32_t)  &Bus_Fault,
    (uint32_t)  &Usage_Fault_Handler
};
```

```c
extern uint32_t _E_TEXT ; // End of text section
extern uint32_t _S_DATA ; // Start of data section
extern uint32_t _E_DATA ; // End of data section
extern uint32_t _S_BSS ; // Start of bss section
extern uint32_t _E_BSS ; // End of bss section

void Reset_Hundler (void)
{
    //copy data from ROM to RAM
    uint32_t DATA_Size = (uint8_t*)&_E_DATA - (uint8_t*)&_S_DATA ;
    uint8_t* P_src = (uint8_t*)&_E_TEXT ;
    uint8_t* P_dst = (uint8_t*)&_S_DATA ;

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

    // init the .bss with zero
    uint32_t BSS_Size = (uint8_t*)&_E_BSS - (uint8_t*)&_S_BSS ;
    P_dst = (uint8_t*)&_S_BSS ;

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

    //jump to main
    main();
}
```

typedef unsigned int uint32_t

- Compilation code

```
ibrahim@DESKTOP-PF9T1AH MINGW32 /d/Pressure_Detection_P1
$ mingw32-make.exe
arm-none-eabi-gcc.exe -c -I . -g -mcpu=arm926ej-s Alarm_Actu.c -o Alarm_Actu.o
arm-none-eabi-gcc.exe -c -I . -g -mcpu=arm926ej-s Alarm_Mon.c -o Alarm_Mon.o
arm-none-eabi-gcc.exe -c -I . -g -mcpu=arm926ej-s Driver.c -o Driver.o
arm-none-eabi-gcc.exe -c -I . -g -mcpu=arm926ej-s Main_Alg.c -o Main_Alg.o
arm-none-eabi-gcc.exe -c -I . -g -mcpu=arm926ej-s P_SEN.c -o P_SEN.o
arm-none-eabi-gcc.exe -c -I . -g -mcpu=arm926ej-s main.c -o main.o
arm-none-eabi-gcc.exe -c -I . -g -mcpu=arm926ej-s startup.c -o startup.o
arm-none-eabi-ld.exe -T linker_script.ld  Alarm_Actu.o Alarm_Mon.o Driver.o Main
_Alg.o P_SEN.o main.o startup.o  -o learn-in-depth.elf -Map=Map_file.map
arm-none-eabi-objcopy.exe -O binary learn-in-depth.elf learn-in-depth.bin
=== Bulid is done ===
```

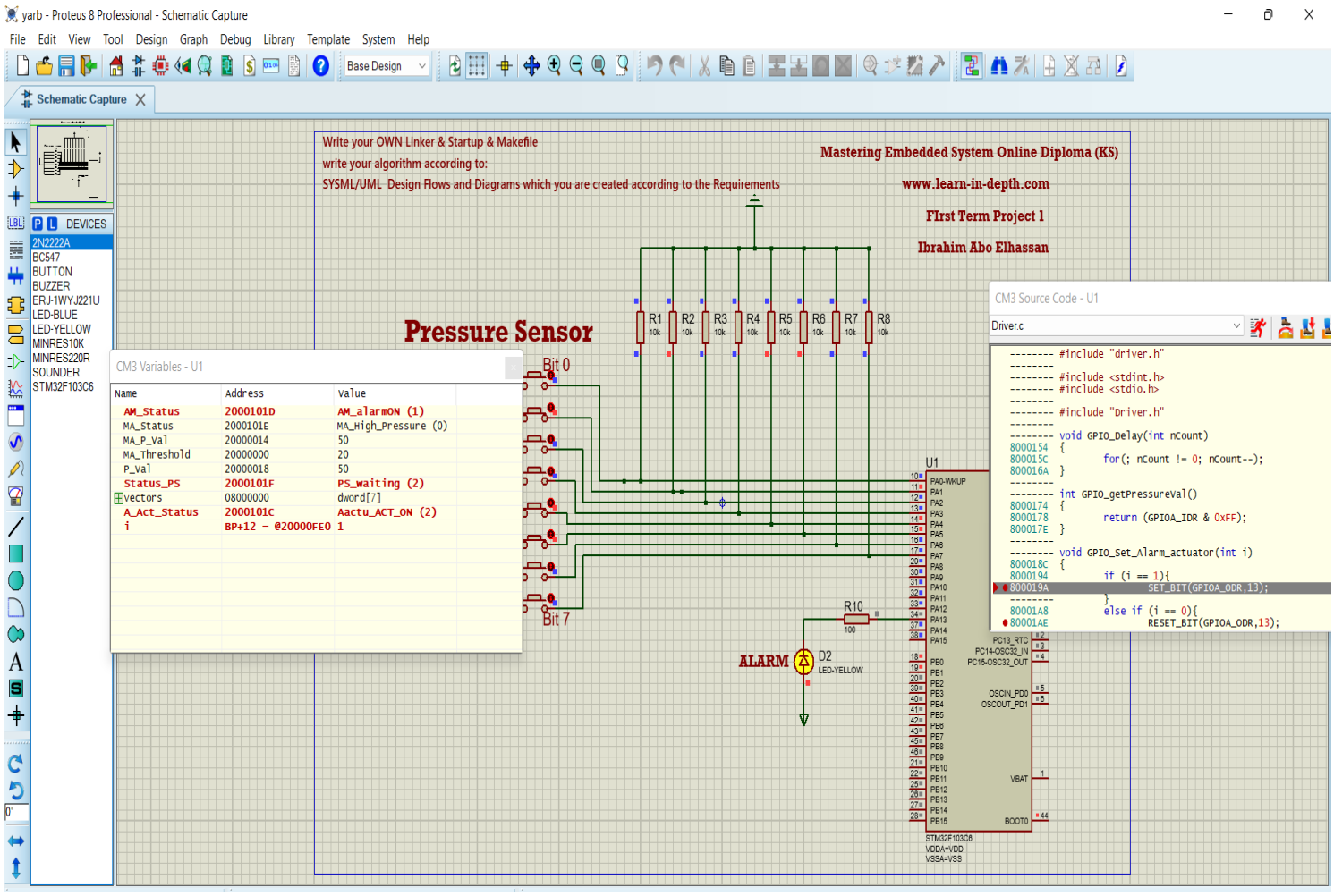
- Symbol table

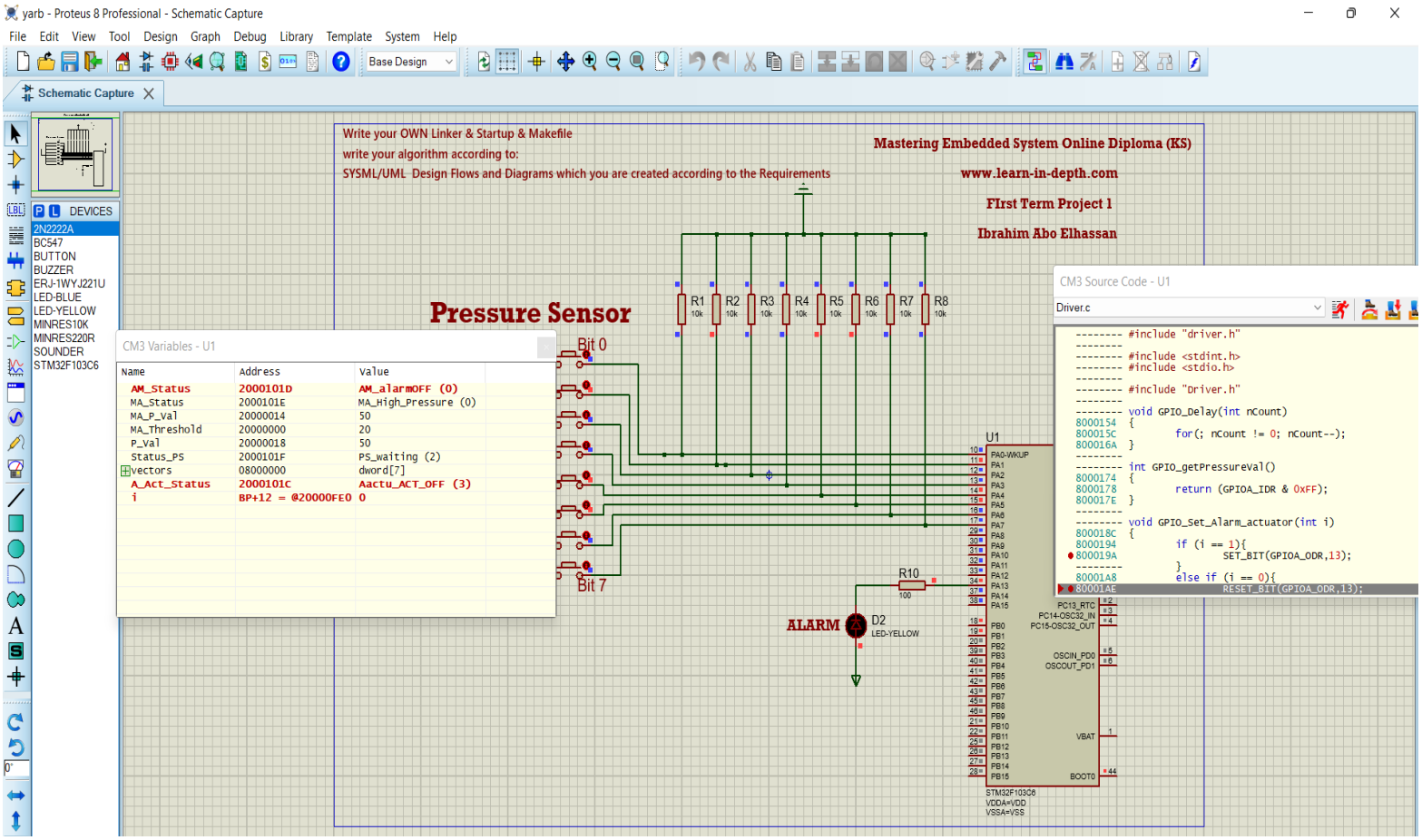
```
ibrahim@DESKTOP-PF9T1AH MINGW32 /d/Pressure_Detection_P1
$ arm-none-eabi-nm.exe Pressure_Detection_cortex_m3.elf
2000001c B _E_BSS
20000014 D _E_DATA
080003c4 T _E_TEXT
20000014 B _S_BSS
20000000 D _S_DATA
2000101c B _STACK_TOP
2000101c B A_Act_Status
080000ac T Aactu_Start_Alarm
20000008 D Aactu_state
080000c8 T Aactu_Stop_Alarm
2000000c D AM_state
2000101d B AM_Status
08000334 W Bus_Fault
08000334 T Default_Hundler
08000154 T GPIO_Delay
08000174 T GPIO_getPressureVal
080001c8 T GPIO_INITIALIZATION
0800018c T GPIO_Set_Alarm_actuator
08000334 W H_Fault_Handler
08000248 T MA_High_Pressure_Detect
20000014 B MA_P_Val
20000010 D MA_state
2000101e B MA_Status
20000000 D MA_Threshold
08000300 T main
08000334 W MM_Fault_Handler
08000334 W NMI_Handler
20000018 b P_Val
080002ec T PS_catch_Pressure_Val
20000004 D PS_state
08000340 T Reset_Hundler
08000084 T ST_Aactu_ACT_OFF
0800005c T ST_Aactu_ACT_ON
0800001c T ST_Aactu_init
08000040 T ST_Aactu_waiting
080000e4 T ST_AM_alarmOFF
08000108 T ST_AM_alarmON
0800012c T ST_AM_waiting
08000218 T ST_MA_High_Pressure
08000270 T ST_PS_init
08000294 T ST_PS_Reading
080002c4 T ST_PS_waiting
2000101f B Status_PS
08000334 W Usage_Fault_Handler
08000000 T vectors
```

- Map file

```
Memory Configuration

Name             Origin             Length             Attributes
flash            0x0000000008000000 0x0000000000020000 xr
sram             0x0000000020000000 0x0000000000005000 xrw
*default*        0x0000000000000000 0xffffffffffffffff

Linker script and memory map


.text           0x0000000008000000      0x614
 *(.vectors*)
 .vectors        0x0000000008000000       0x1c startup.o
                 0x0000000008000000                vectors
 *(.text*)
 .text           0x000000000800001c      0x138 Alarm_Actu.o
                 0x000000000800001c                ST_Aactu_init
                 0x0000000008000058                ST_Aactu_waiting
                 0x0000000008000084                ST_Aactu_ACT_ON
                 0x00000000080000c0                ST_Aactu_ACT_OFF
                 0x00000000080000fc                Aactu_Start_Alarm
                 0x0000000008000128                Aactu_Stop_Alarm
 .text           0x0000000008000154       0xac Alarm_Mon.o
                 0x0000000008000154                ST_AM_alarmOFF
                 0x000000000800018c                ST_AM_alarmON
                 0x00000000080001c4                ST_AM_waiting
 .text           0x0000000008000200      0x158 Driver.o
                 0x0000000008000200                GPIO_Delay
                 0x000000000800023c                GPIO_getPressureVal
                 0x0000000008000264                GPIO_Set_Alarm_actuator
                 0x00000000080002cc                GPIO_INITIALIZATION
 .text           0x0000000008000358       0x88 Main_Alg.o
                 0x0000000008000358                ST_MA_High_Pressure
                 0x00000000080003a0                MA_High_Pressure_Detect
 .text           0x00000000080003e0       0xe4 P_SEN.o
                 0x00000000080003e0                ST_PS_init
                 0x000000000800041c                ST_PS_Reading
                 0x0000000008000464                ST_PS_waiting
                 0x00000000080004a0                PS_catch_Pressure_Val
 .text           0x00000000080004c4       0x50 main.o
                 0x00000000080004c4                main
 .text           0x0000000008000514      0x100 startup.o
                 0x0000000008000514                NMI_Handler
                 0x0000000008000514                H_Fault_Handler
                 0x0000000008000514                Default_Hundler
                 0x0000000008000514                MM_Fault_Handler
                 0x0000000008000514                Bus_Fault
                 0x0000000008000514                Usage_Fault_Handler
                 0x0000000008000528                Reset_Hundler
 *(.rodata*)
                 0x0000000008000614                _E_TEXT = .

.glue_7         0x0000000008000614        0x0
 .glue_7         0x0000000008000614        0x0 linker stubs

.glue_7t        0x0000000008000614        0x0
 .glue_7t        0x0000000008000614        0x0 linker stubs

.vfp11_veneer   0x0000000008000614        0x0
 .vfp11_veneer   0x0000000008000614        0x0 linker stubs

.v4_bx          0x0000000008000614        0x0
 .v4_bx          0x0000000008000614        0x0 linker stubs

.iplt           0x0000000008000614        0x0
 .iplt           0x0000000008000614        0x0 Alarm_Actu.o

.rel.dyn        0x0000000008000614        0x0
 .rel.iplt       0x0000000008000614        0x0 Alarm_Actu.o

.data           0x0000000020000000       0x14 load address 0x0000000008000614
```

- Simulation on Proteus




yarb - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
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
Ibrahim Abo Elhassan
Pressure Sensor
DEVICES
2N2222A
BC547
BUTTON
BUZZER
ERJ-1WYJ221U
LED-BLUE
LED-YELLOW
MINRES10K
MINRES220R
SOUNDER
STM32F103C6
R1 10k
R2 10k
R3 10k
R4 10k
R5 10k
R6 10k
R7 10k
R8 10k
Bit 0
Bit 7
R10
100
ALARM
D2
LED-YELLOW
U1
STM32F103C6
VDDA=VDD
VSSA=VSS
CM3 Variables - U1
Name Address Value
AM_Status 2000101D AM_alarmOFF (0)
MA_Status 2000101E MA_High_Pressure (0)
MA_P_Val 20000014 50
MA_Threshold 20000000 20
P_Val 20000018 50
Status_PS 2000101F PS_waiting (2)
vectors 08000000 dword[7]
A_Act_Status 2000101C Aactu_ACT_OFF (3)
i BP+12 = @20000FE0 0
CM3 Source Code - U1
Driver.c
#include "driver.h"
#include <stdint.h>
#include <stdio.h>
#include "Driver.h"
void GPIO_Delay(int nCount)
{
for(; nCount != 0; nCount--);
}
int GPIO_getPressureVal()
{
return (GPIOA_IDR & 0xFF);
}
void GPIO_Set_Alarm_actuator(int i)
{
if (i == 1){
SET_BIT(GPIOA_ODR,13);
}
else if (i == 0){
RESET_BIT(GPIOA_ODR,13);