Mastering Embedded System Online Diploma

www.learn-in-depth.com

First Term (Final Project1)

Report for "High_Pressure_Detection" project

Name: Mostafa Mohamed Mourad

Mail: mostafamourad108@gmail.com

Github repo:
https://github.com/mostafamohamedmahmoudmourad/Embedded_System_Online_Diploma/tree/master

My profile:

https://www.learn-in-depth.com/certificates-2023

Case Study

The system should be in a plain cabin, if the pressure in the cabin is higher we will put a sensor -to sense the high pressure- and connected with an Alarm to tell the crew in the cabin that there is a high pressure to be safe
The Sensor should make Alarm if the pressure is greater than 20 bar.

Then the Alarm -The led- will turn on 60s and the Alarm still turn on if the pressure is greater than 20bar to tell the crew or the Pilot that there are a danger on the plain.

Files

1. State.h

```c
/*
 * state.h
 *
 *  Created on: 12 Aug 2024
 *      Author: Mostafa Mourad
 */

#ifndef STATE_H_  // Prevent multiple inclusions of this header
#define STATE_H_

// Macro to define a state function
#define STATE_define(_statFUN_)     void ST_##_statFUN_()

// Macro to call a state function
#define STATE(_statFUN_)            ST_##_statFUN_

// Function prototypes for handling pressure and alarms
void set_pressure_val(int Pval);
void HighPressure_detected();
void StartAlarm();
void StopAlarm();

#endif /* STATE_H_ */
```

2. Main.c

```c
 *
 *  Created on: 12 Aug 2024
 *      Author: Mostafa Mourad
 */
#include <stdint.h>
#include <stdio.h>
#include "driver.h"
#include "MainAlg.h"
#include "AlarmMonitor.h"
#include "AlarmActuatorDriver.h"
#include "PressureSensorDriver.h"

void setup()
{
    AlarmActuatorDriver_init();                                 // Initialize the alarm actuator driver
    PressureSensorDriver_state = STATE(PressureSensorDriver_reading);  // Set the initial state for the alarm actuator driver
    MainAlg_state = STATE(MainAlg_high_pressure_delection);     // Set the initial state for the main algorithm
    AlarmMonitor_state = STATE(AlarmMonitor_alarmOff);          // Set the initial state for the alarm monitor
    StopAlarm();                                                // Ensure the alarm is off at the start
}

int main()
{
    //GPIO_INITIALIZATION();  // Initialize the GPIOs
    //GPIOA_BRR = 0x8000;
    setup();  // Call the setup function to initialize states and drivers

    while (1)  // Main loop
    {
        PressureSensorDriver_state();       // Execute the current state of the alarm actuator driver
        MainAlg_state();                    // Execute the current state of the main algorithm
        AlarmMonitor_state();               // Execute the current state of the alarm monitor
    }
}
```

3. Driver.h

```h
#include <stdint.h>
#include <stdio.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 Delay(int nCount);
int getPressureVal();
void Set_Alarm_actuator(int i);
void GPIO_INITIALIZATION ();
```

4.Driver.c

```c
#include "driver.h"
#include <stdint.h>
#include <stdio.h>


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

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

void 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;
}
```

5.mainAlgorith.h

```c
/*
 * MainAlg.h
 *
 *  Created on: 12 Aug 2024
 *      Author: Mostafa Mourad
 */

#ifndef MAINALG_H_  // Prevent multiple inclusions of this header
#define MAINALG_H_

#include "state.h"
#include "driver.h"



// Enum to define the states for the Main Algorithm
enum
{
    MainAlg_high_pressure_delection  // State ID for high-pressure detection
} MainAlg_state_id;


// Declare the state function for high-pressure detection
STATE_define(MainAlg_high_pressure_delection);

// Pointer to the current state function of the Main Algorithm
extern void (* MainAlg_state)();

#endif /* MAINALG_H_ */
```

6.mainAlgoritm.c

```c
/*
 * MainAlg.c
 *
 *  Created on: 12 Aug 2024
 *      Author: Mostafa Mourad
 */

#include "MainAlg.h"

static int MainAlg_Pval = 0;  // Variable to store the pressure value in the main algorithm
static int MainAlg_threshold = 20;  // Threshold value for detecting high pressure



void (* MainAlg_state)();  // Function pointer to the current state function

// Function to set the pressure value
void set_pressure_val(int Pval)
{
    MainAlg_Pval = Pval;  // Update the pressure value
}



// State function for high-pressure detection
STATE_define(MainAlg_high_pressure_delection)
{
    MainAlg_state_id = MainAlg_high_pressure_delection;  // Set the current state ID

    // Check if the pressure value exceeds the threshold
    if (MainAlg_Pval > MainAlg_threshold)
        HighPressure_detected();  // Trigger high-pressure detection action
}
```

7.highpressre_detction.h

```h
/*
 * PressureSensorDriver.h
 *
 *  Created on: 12 Aug 2024
 *      Author: Mostafa Mourad
 */

#ifndef PRESSURESENSORDRIVER_H_  // Prevent multiple inclusions of this header
#define PRESSURESENSORDRIVER_H_

#include "state.h"
#include "driver.h"



// Enum to define the state of the pressure sensor driver
enum {
    PressureSensorDriver_reading  // State ID for reading sensor data
} PressureSensorDriver_id;


// Declare the state function for reading sensor data
STATE_define(PressureSensorDriver_reading);


// Function prototype to initialize the pressure sensor driver
void PressureSensorDriver_init();

// Pointer to the current state function of the pressure sensor driver
extern void (* PressureSensorDriver_state)();

#endif /* PRESSURESENSORDRIVER_H_ */
```

8.highpressre_detction.c

```c
/*
 * PressureSensorDriver.c
 *
 *  Created on: 12 Aug 2024
 *      Author: Mostafa Mourad
 */

#include "PressureSensorDriver.h"

static int PressureSensorDriver_pVal = 0;  // Variable to store the pressure sensor value

void (* PressureSensorDriver_state)();  // Function pointer to the current state function



void PressureSensorDriver_init()
{
    GPIO_INITIALIZATION();  // Initialize GPIO for the pressure sensor
}



STATE_define(PressureSensorDriver_reading)
{
    PressureSensorDriver_id = PressureSensorDriver_reading;      // Set the current state ID
    PressureSensorDriver_pVal = getPressureVal();                // Read the pressure sensor value
    set_pressure_val(PressureSensorDriver_pVal);                 // Pass the value to the state management system
    Delay(10000);                                                // Delay for a specified time
}
```

9.MonitorAlarm.h

```c
/*
 * AlarmMonitor.h
 *
 *  Created on: 12 Aug 2024
 *      Author: Mostafa Mourad
 */

#ifndef ALARMMONITOR_H_  // Prevent multiple inclusions of this header
#define ALARMMONITOR_H_

#include "state.h"
#include "driver.h"



// Enum to define the states for the alarm monitor
enum {
    AlarmMonitor_alarmOff,  // State ID for when the alarm is off
    AlarmMonitor_alarmOn    // State ID for when the alarm is on
} AlarmMonitor_state_id;


// Declare state functions for alarm on and off states
STATE_define(AlarmMonitor_alarmOff);
STATE_define(AlarmMonitor_alarmOn);

// Pointer to the current state function of the alarm monitor
extern void (* AlarmMonitor_state)();

#endif /* ALARMMONITOR_H_ */
```

10.MonitorAlarm.c

```c
/*
 * AlarmMonitor.c
 *
 *  Created on: 12 Aug 2024
 *      Author: Mostafa Mourad
 */

#include "AlarmMonitor.h"

static int AlarmMonitor_periode = 60000;  // Duration for which the alarm stays on
void (* AlarmMonitor_state)();  // Function pointer to the current state function

// Function to handle high-pressure detection
void HighPressure_detected()
{
    AlarmMonitor_state = STATE(AlarmMonitor_alarmOn);  // Switch to the alarm on state
}

// State function for when the alarm is off
STATE_define(AlarmMonitor_alarmOff)
{
    AlarmMonitor_state_id = AlarmMonitor_alarmOff;  // Set the current state ID
    StopAlarm();  // Stop the alarm
}

// State function for when the alarm is on
STATE_define(AlarmMonitor_alarmOn)
{
    AlarmMonitor_state_id = AlarmMonitor_alarmOn;  // Set the current state ID

    StartAlarm();  // Start the alarm
    Delay(AlarmMonitor_periode);  // Keep the alarm on for a specified duration
    StopAlarm();  // Stop the alarm
    AlarmMonitor_state = STATE(AlarmMonitor_alarmOff);  // Switch to the alarm off state
}
```

11.AlarmLed.h

```c
/*
 * AlarmActuatorDriver.h
 *
 *  Created on: 12 Aug 2024
 *      Author: Mostafa Mourad
 */

#ifndef ALARMACTUATORDRIVER_H_  // Prevent multiple inclusions of this header
#define ALARMACTUATORDRIVER_H_

#include "state.h"
#include "driver.h"

// Enum to define the states for the alarm actuator driver
enum {
    AlarmActuatorDriver_waiting,  // State ID for waiting state
    AlarmActuatorDriver_AlarmOn,  // State ID for when the alarm is on
    AlarmActuatorDriver_AlarmOff  // State ID for when the alarm is off
} AlarmActuatorDriver_state_id;


// Declare state functions for the alarm actuator driver
STATE_define(AlarmActuatorDriver_waiting);
STATE_define(AlarmActuatorDriver_AlarmOn);
STATE_define(AlarmActuatorDriver_AlarmOff);


// Function to initialize the alarm actuator driver
void AlarmActuatorDriver_init();

// Pointer to the current state function of the alarm actuator driver
extern void (* AlarmActuatorDriver_state)();

#endif /* ALARMACTUATORDRIVER_H_ */
```

12.AlarmLed.c

```c
void (* AlarmActuatorDriver_state)();  // Function pointer to the current state function
// Function to initialize the alarm actuator driver
void AlarmActuatorDriver_init()
{
    GPIO_INITIALIZATION();  // Initialize GPIO for the alarm actuator
}
// Function to start the alarm
void StartAlarm()
{
    AlarmActuatorDriver_state = STATE(AlarmActuatorDriver_AlarmOn);  // Switch to the alarm on state
    AlarmActuatorDriver_state();  // Execute the current state function
}
// Function to stop the alarm
void StopAlarm()
{
    AlarmActuatorDriver_state = STATE(AlarmActuatorDriver_AlarmOff);  // Switch to the alarm off state
    AlarmActuatorDriver_state();  // Execute the current state function
}
STATE_define(AlarmActuatorDriver_waiting)
{
    AlarmActuatorDriver_state_id = AlarmActuatorDriver_waiting;  // Set the current state ID
}
// State function for when the alarm is on
STATE_define(AlarmActuatorDriver_AlarmOn)
{
    AlarmActuatorDriver_state_id = AlarmActuatorDriver_AlarmOn;  // Set the current state ID
    Set_Alarm_actuator(0);  // Activate the alarm actuator
    AlarmActuatorDriver_state = STATE(AlarmActuatorDriver_waiting);  // Switch to the waiting state
}
// State function for when the alarm is off
STATE_define(AlarmActuatorDriver_AlarmOff)
{
    AlarmActuatorDriver_state_id = AlarmActuatorDriver_AlarmOff;  // Set the current state ID
    Set_Alarm_actuator(1);  // Deactivate the alarm actuator
    AlarmActuatorDriver_state = STATE(AlarmActuatorDriver_waiting);  // Switch to the waiting state
}
```

13.Startup.c

```c
//startup.c
//Eng: Mostafa Mouard

#include <stdint.h>

static unsigned long stack_top[256];
void(* const g_p_fn_Vectors[])() __attribute__ ((section(".vectors")));
extern int main();
void Reset_Handler(void);
void Default_Handler()
{
    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()     __attribute__((weak,alias("Default_Handler")));;
void Bus_Handler()          __attribute__((weak,alias("Default_Handler")));;
void Usage_Fault_Handler()  __attribute__((weak,alias("Default_Handler")));;


void(* const g_p_fn_Vectors[])() __attribute__ ((section(".vectors"))) =
{
    (void (*)())    ((unsigned long)stack_top + sizeof(stack_top)),
    &Reset_Handler,
    &NMI_Handler,
    &H_fault_Handler
};

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;
```

```c
void Reset_Handler(void)
{
    // Copy data from ROM to RAM
    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 (int i = 0; i < DATA_size; i++)
    {
        *((unsigned char*)P_dst++) = *((unsigned char*)P_src++);
    }

    // Initialize the .bss with zero
    unsigned int bss_size = (unsigned char*)&_E_bss - (unsigned char*)&_S_bss;
    P_dst = (unsigned char*)&_S_bss;
    for (int i = 0; i < bss_size; i++)
    {
        *((unsigned char*)P_dst++) = (unsigned char)0;
    }

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

14.Linker_script.ld

```
/* linker script CortexM3
Eng: Mostafa Mourad
*/

MEMORY
{
    flash(RX) : ORIGIN = 0x08000000, LENGTH = 128k
    sram(RWX) : ORIGIN = 0x20000000, LENGTH = 30k
}

SECTIONS
{
    .text : {
        *(.vectors*)
        *(.text*)
        *(.rodata)
        _E_text = . ;
    }>flash
    .data : {
        _S_DATA = . ;
        *(.data)
        . = ALIGN(4);
        _E_DATA = .;
    }> sram AT> flash
    .bss : {
        _S_bss = . ;
        *(.bss*)
        . = ALIGN(4);
        _E_bss = . ;
    }>sram
}
```

15.Makefile

```
#@copyright : Mostafa
#
CC=arm-none-eabi-
CFLAGS= -mcpu=cortex-m3 -mthumb -gdwarf-2 -std=c99
ASFLAGS= -mcpu=cortex-m3 -mthumb -gdwarf-2
INCS=-I .
LIBS=
As = $(wildcard *.s)
AsOBJ= $(As:.s=.o)
SRC = $(wildcard *.c)
OBJ = $(SRC:.c=.o) $(AsOBJ)
Project_name= PressureDetectionSystem

all: $(Project_name).bin
    @echo "----------Build is Done-----------"

%.o: %.s
    $(CC)as.exe $(ASFLAGS) $< -o $@ 2> log

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

$(Project_name).elf: $(OBJ)
    $(CC)ld.exe -T linker_script.ld $(LIBS) $(OBJ) -o $@ -Map=output.map
    cp  $(Project_name).elf $(Project_name).axf
$(Project_name).bin: $(Project_name).elf
    $(CC)objcopy.exe -O binary $< $@

clean_all:
    rm -f *.o *.elf *.bin

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

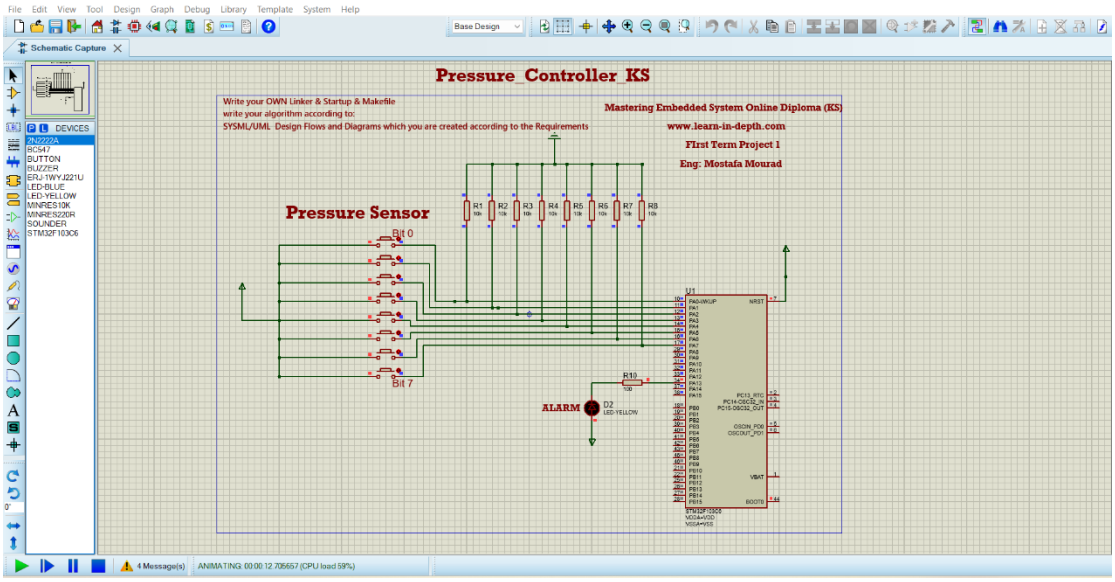
16.mapFile

```
                0x4                AlarmActuatorDriver.o
 AlarmMonitor_state_id
                0x1                AlarmMonitor.o
 AlarmActuatorDriver_state_id
                0x1                AlarmActuatorDriver.o

 Memory Configuration

 Name             Origin             Length             Attributes
 flash            0x08000000         0x00020000         xr
 sram             0x20000000         0x00007800         xrw
 *default*        0x00000000         0xffffffff

 Linker script and memory map


 .text           0x08000000      0x424
  *(.vectors*)
  .vectors       0x08000000       0x10 startup.o
                 0x08000000                g_p_fn_Vectors
  *(.text*)
  .text          0x08000010       0xcc AlarmActuatorDriver.o
                 0x08000010                AlarmActuatorDriver_init
                 0x0800001c                StartAlarm
                 0x08000040                StopAlarm
                 0x08000064                ST_AlarmActuatorDriver_waiting
                 0x0800007c                ST_AlarmActuatorDriver_AlarmOn
                 0x080000ac                ST_AlarmActuatorDriver_AlarmOff
  .text          0x080000dc       0x74 AlarmMonitor.o
                 0x080000dc                HighPressure_detected
                 0x080000f8                ST_AlarmMonitor_alarmOff
                 0x08000110                ST_AlarmMonitor_alarmOn
  .text          0x08000150      0x10c driver.o
                 0x08000150                Delay
                 0x08000174                getPressureVal
```

17.symbolTable

```
HP@LAPTOP-PPV7E6PE MINGW64 /e/Embedded System/Basic/Embedded_System_Online_Diplo
ma/FIRST TERM Project/1-HighPressure_Detection (master)
$ arm-none-eabi-nm.exe PressureDetectionSystem.elf
20000410 B _E_bss
20000008 D _E_DATA
08000424 T _E_text
20000008 B _S_bss
20000000 D _S_DATA
08000010 T AlarmActuatorDriver_init
20000410 B AlarmActuatorDriver_state
20000414 B AlarmActuatorDriver_state_id
20000000 d AlarmMonitor_periode
20000418 B AlarmMonitor_state
2000041c B AlarmMonitor_state_id
08000368 W Bus_Handler
08000368 T Default_Handler
08000150 T Delay
08000000 T g_p_fn_Vectors
08000174 T getPressureVal
080001dc T GPIO_INITIALIZATION
08000368 W H_fault_Handler
080000dc T HighPressure_detected
080002a0 T main
20000008 b MainAlg_Pval
20000420 B MainAlg_state
2000041e B MainAlg_state_id
20000004 d MainAlg_threshold
08000368 W MM_Fault_Handler
08000368 W NMI_Handler
2000041d B PressureSensorDriver_id
08000320 T PressureSensorDriver_init
2000000c b PressureSensorDriver_pVal
20000424 B PressureSensorDriver_state
08000374 T Reset_Handler
0800018c T Set_Alarm_actuator
080002d0 T set_pressure_val
0800025c T setup
080000ac T ST_AlarmActuatorDriver_AlarmOff
0800007c T ST_AlarmActuatorDriver_AlarmOn
08000064 T ST_AlarmActuatorDriver_waiting
080000f8 T ST_AlarmMonitor_alarmOff
08000110 T ST_AlarmMonitor_alarmOn
080002f0 T ST_MainAlg_high_pressure_detection
0800032c T ST_PressureSensorDriver_reading
20000010 b stack_top
0800001c T StartAlarm
08000040 T StopAlarm
08000368 W Usage_Fault_Handler
```

18.simulation before burn the bin file



19.simulation after burn the bin file