LAB REPORT

Design of Automatic Fan

Major	Mechanical & Control Engineering	**Subject**	Embedded Controller
Student ID	21700791	**Professor**	Kim Young Keun
Name	Hong Se Hyun	**Date**	2021.09.23.

Table of Contents

Design of Automatic Fan

Name

Hong Se Hyun

초 록

기술의 빠른 발전과 함께, 점차 소프트웨어대한 기본적인 이해가 요구되고 있다. 이에 본 실험에서는 기본적인 임베디드 시스템에 대한 이해를 위해 mbed API를 사용하여 간단한 디지털 애플리케이션을 구현하였다. 앞으로 있을 몇몇의 추가 실험을 하기 위해서는 해당 실험에 대한 이해가 바탕이 되어야 하기에 본 실험은 기본적인 임베디드 코딩에 대한 이해를 돕는 것을 목적으로 한다. 실험은 선행 연구와 서적 등 공인된 자료를 활용하여 신뢰성 및 타당성을 높였다.

Abstract

With the rapid development of technology, a basic understanding of software is gradually required. Therefore, in this experiment, a simple digital application was implemented using an mbed API to understand the basic embedded system. The purpose of this experiment is to help understand basic embedded coding, as some additional experiments in this semester based on an understanding of this experiment. The experiment increased reliability and validity by using authorized data such as previous studies and books.

I Introduction

1. Introduction

In this lab, I created a simple program that uses mbed API for implementing a simple embedded digital application. Refer to online mbed documentation for the full list of APIs

2. Procedure

Program needs to runs DC motor only when the distance of an object is within a certain value.

Example: An automatic mini-fan that runs only when the face is near the fan

1. As the button B1 is pressed, change the DC motor velocity
   - The mode is OFF(0%), MID(50%), HIGH(70%), V.HIGH(100%)
   - As the B1 is pressed, it should toggle from OFF mode to V.HIGH mode and so on
2. Automatically ReStart and Stop the DC motor when the mode is
   - RESTART: The distance is within about 50mm
   - STOP: The distance is beyond about 50mm
3. Print the distance and PWM duty ratio in Tera-Term console (every 2 sec).
4. When the DC is turned OFF temporarily then turned on again depending on the distance as in Condition (2), it should be turning at the previous speed.
5. When the mode is OFF, turn off the LED(LED1). Otherwise Blink the LED by 1 sec.

3. Configuration

Ultrasonic Distance Sensor

Trigger:

Generate a trigger pulse as PWM to the sensor (D10 or PB_6)

PWM out: 50ms period, 10us pulse-width, TIM4_CH1

Echo:

Receive echo pulses from the ultrasonic sensor

Input Capture: Input mode, Timer1_Ch1 (D7 or PA_8)

Measure the distance by calculating pulse-width of the echo pulse.

USART

Display measured distance in [cm] on serial monitor of Tera-Term.

Baudrate 9600

DC Motor

PWM: PWM1(Timer1 CH1N), set 10ms of period by default

Pin: D11 or PA_7

II Experiment

1. Experimental Equipment

Hardware

NUCLEO-F401RE or NUCLEO-F411RE

Ultrasonic distance sensor(HC-SR04), DC Motor(RK-280RA)

Software

Mbed OS

2. Flow-Chart

State	Motor-Mode	Motor-Pulse-Width	LED-Toggle
S_0	OFF	0	0
S_1	MID	0.5	T
S_2	HIGH	0.7	T
S_3	VERY HIGH	1	T
S_4	PAUSE	0	T
S_5	S_{prev}	S_{prev}	S_{prev}

Table 1.1 State Table

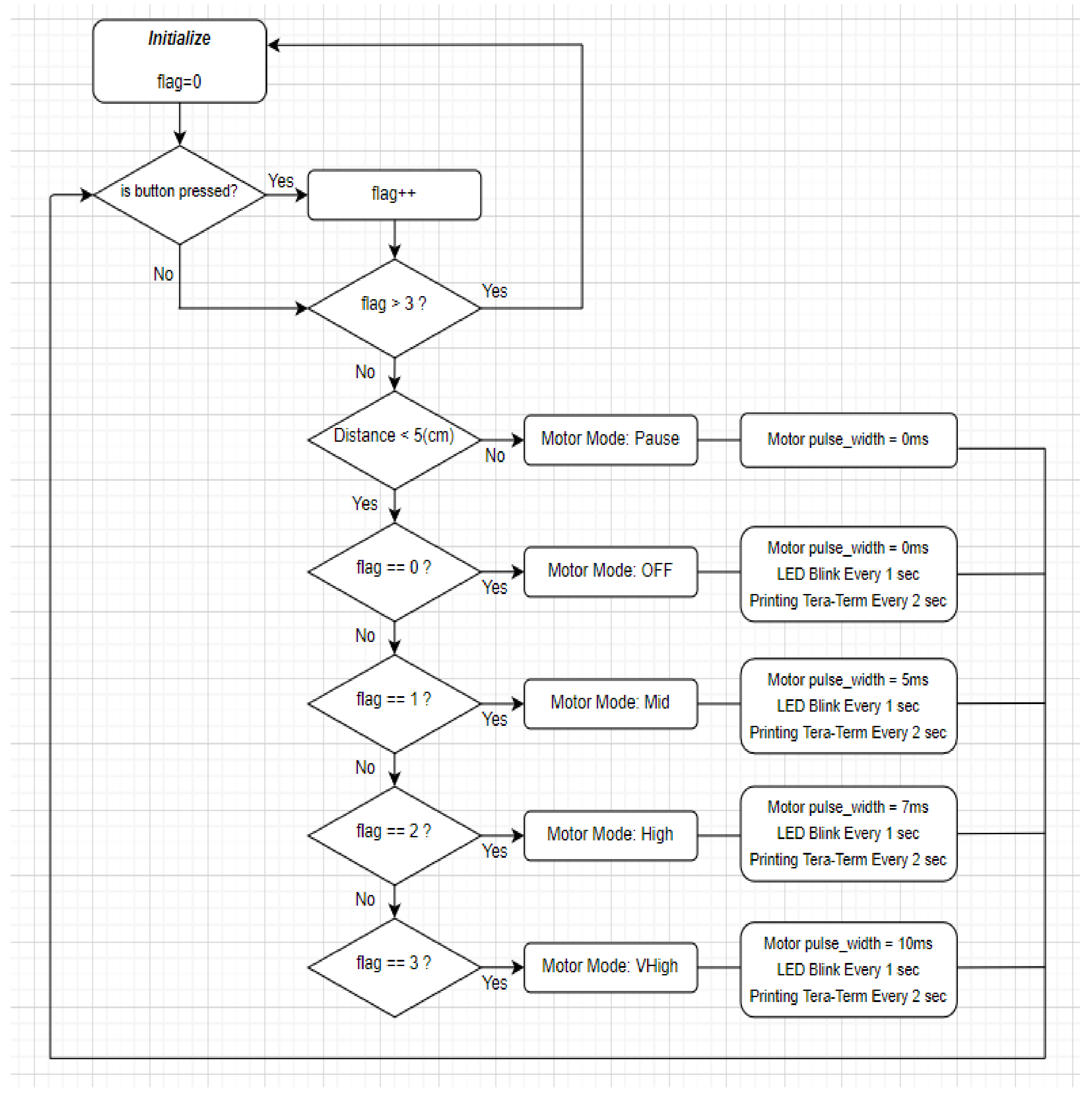


Figure 1.1 Flow-Chart of Automatic Fan

Flow-chart was completed based on the state table above. S_0 state when flag is 0, S_1 state when flag is 1, S_2 state when flag is 2, and S_3 state when flag is 3.

The special state, S_4, is a state in which the motor stops for a while when the distance exceeds $50mm$ while the motor is operating. This was processed through the $'Distance < 5(cm)'$ conditional sentence. Another special state, S_5, is a state in which the motor outputs the same state just before it is stopped when it is temporarily stopped and then operated again. Through the $'Flag'$ operation processing, this state was implemented in a way that remembers and maintains the existing motor state unless an $'Interrupt'$ called $'Button.fall'$ occurs.

3. Source Code

```
#include "mbed.h"

//Define Structure
DigitalOut      led(LED1);
InterruptIn     button(USER_BUTTON);

Serial          pc(USBTX, USBRX, 9600);
PwmOut          pwm1(D11);                  // DC Motor
PwmOut          trig(D10);                  // Trigger 핀
InterruptIn     echo(D7);                   // Echo 핀

Timer           tim;
Ticker          ledTick;
Ticker          printTick;


```

Figure 2.1 Define Structure

```
//Declare Functions
void printStatus();
void ledBLINK();
void rising();
void falling();
void buttonPush();
```

Figure 2.2 Declare Functions

```
//Define Varaibles
int      flag        = 0;
int      begin       = 0;
int      end         = 0;
float    distance    = 0;
unsigned int bPrint  = 0;
```

Figure 2.3 Define Variables

```
int main() {
    //Variable Initializing
    float     period_trig       = 50;           //ms
    float     pulsewidth_trig   = 10;           //us
    float     SensorDutyRatio   = pulsewidth_trig/period_trig;

    float     period_DC         = 10;           //ms
    float     OFF               = 0;            //period_DC*0;
    float     Half              = 5;            //period_DC*0.5;
    float     High              = 7;            //period_DC*0.7;
    float     VHigh             = period_DC;
    float     distance          = 0;

```

Figure 2.4 Variable Initializing

By treating both the ultrasonic sensor and the motor's Period and Pulse-width as variables, I designed the code more easily understandable and more easily moddable.

```
    trig.period_ms(period_trig);              // period      = 50ms
    trig.pulsewidth_us(pulsewidth_trig);      // pulse-width = 10us

    pwm1.period_ms(period_DC);                // period      = 10ms

    printTick.attach(&printStatus,2);
    ledTick.attach(&ledBLINK, 1);
    tim.start();
    echo.rise(&rising);
    echo.fall(&falling);

    //Screen Print Settings
    pc.printf("\n\n\n\n\t\r");
    pc.printf("Program START\t\n\r");
```

Figure 2.5 Define Period, Pulse-width, Toggle and Print

```
while(1){
    //Distance Measurement
    distance =  (float)(end - begin) / 58; // [cm]
    button.fall(&buttonPush);

    //Condition_Automatically ReStart and Pause the DC Motor
    //Condition_When the mode is OFF, turn off the LED. Otherwise Blink by 1 sec
    if(distance > 5) {
        pwm1.pulsewidth_us(OFF);
        if(bPrint) {
            pc.printf("Distance               = %.2f[cm]\r\n", distance);
            pc.printf("SENSOR PWM duty ratio = %.1f\r\n", SensorDutyRatio);
            pc.printf("MOTOR PWM duty ratio  = %.1f\r\n", OFF/period_DC);
            bPrint = 0;
            }
        }
```

Figure 2.6 Define State_4

```
        else {
            if(flag==0) {
                pwm1.pulsewidth_ms(OFF);
                if(bPrint) {
                    pc.printf("Distance               = %.2f[cm]\r\n", distance);
                    pc.printf("SENSOR PWM duty ratio = %.1f\r\n", SensorDutyRatio);
                    pc.printf("MOTOR PWM duty ratio  = %.1f\r\n", OFF/period_DC);
                    bPrint = 0;
                }
            }
```

Figure 2.7 Define State_0

```
            else if(flag==1) {
                pwm1.pulsewidth_ms(Half);
                if(bPrint) {
                    pc.printf("Distance               = %.2f[cm]\r\n", distance);
                    pc.printf("SENSOR PWM duty ratio = %.1f\r\n", SensorDutyRatio);
                    pc.printf("MOTOR PWM duty ratio  = %.1f\r\n", Half/period_DC);
                    bPrint = 0;
                }
            }
```

Figure 2.8 Define State_1

```
            else if(flag==2) {
                pwm1.pulsewidth_ms(High);
                if(bPrint) {
                    pc.printf("Distance               = %.2f[cm]\r\n", distance);
                    pc.printf("SENSOR PWM duty ratio = %.1f\r\n", SensorDutyRatio);
                    pc.printf("MOTOR PWM duty ratio  = %.1f\r\n", High/period_DC);
                    bPrint = 0;
                }
            }
```

Figure 2.9 Define State_2

```
            else if(flag==3) {
                pwm1.pulsewidth_ms(period_DC);
                if(bPrint) {
                    pc.printf("Distance             = %.2f[cm]\r\n", distance);
                    pc.printf("SENSOR PWM duty ratio = %.1f\r\n", SensorDutyRatio);
                    pc.printf("MOTOR PWM duty ratio  = %.1f\r\n", VHigh/period_DC);
                    bPrint = 0;
                }
            }
        }
    }
}
```

Figure 2.10 Define State_3

Each state, S_0, S_1, S_2, S_3, S_4 was defined through 'if, else' conditional processing.

```
//Define Functions
void ledBLINK() {
    if(flag==0)
        led = 0;
    else
        led = !led;
}
```

Figure 2.11 Define Blink Function

```
void buttonPush() {
    flag++;

    if(flag>=4)
    flag=0;
}
```

Figure 2.12 Define button Push Function

```
void printStatus() {
    bPrint=1;
}
```

Figure 2.13 Print Status

```
void rising(){
    begin = tim.read_us();
}
```

Figure 2.14 Get Begin Time

```
void falling(){
    end = tim.read_us();
}
```

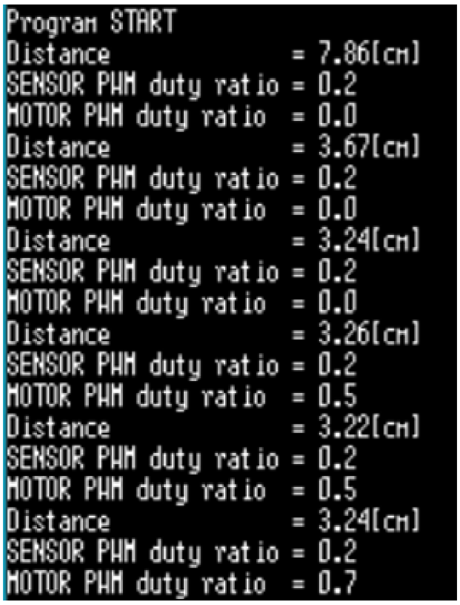
Figure 2.15 Get End Time

As can be seen in 'Figure 2.12', it is designed to move on to the next state only when 'Button Push' is performed through 'Flag' treatment. In other words, it can be seen that S_5, a state that maintains and remembers the previous state, is implemented.

III Experiment

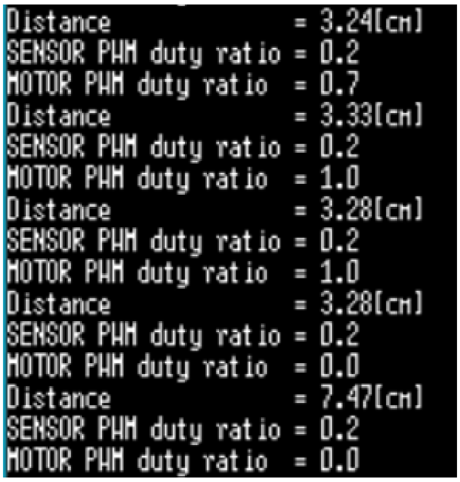
1. Tera-Term

Within the prescribed distance range, it was confirmed that the speed was controlled as the 'duty ratio' of the motor changed every time a button was pressed. In addition, it was also confirmed through the below screen that the motor was 'Pause' when the prescribed distance was exceeded.



```
Program START
Distance              = 7.86[cm]
SENSOR PWM duty ratio = 0.2
MOTOR PWM duty ratio  = 0.0
Distance              = 3.67[cm]
SENSOR PWM duty ratio = 0.2
MOTOR PWM duty ratio  = 0.0
Distance              = 3.24[cm]
SENSOR PWM duty ratio = 0.2
MOTOR PWM duty ratio  = 0.0
Distance              = 3.26[cm]
SENSOR PWM duty ratio = 0.2
MOTOR PWM duty ratio  = 0.5
Distance              = 3.22[cm]
SENSOR PWM duty ratio = 0.2
MOTOR PWM duty ratio  = 0.5
Distance              = 3.24[cm]
SENSOR PWM duty ratio = 0.2
MOTOR PWM duty ratio  = 0.7
```

Figure 3.1 Result from Tera-Term (1)



```
Distance              = 3.24[cm]
SENSOR PWM duty ratio = 0.2
MOTOR PWM duty ratio  = 0.7
Distance              = 3.33[cm]
SENSOR PWM duty ratio = 0.2
MOTOR PWM duty ratio  = 1.0
Distance              = 3.28[cm]
SENSOR PWM duty ratio = 0.2
MOTOR PWM duty ratio  = 1.0
Distance              = 3.28[cm]
SENSOR PWM duty ratio = 0.2
MOTOR PWM duty ratio  = 0.0
Distance              = 7.47[cm]
SENSOR PWM duty ratio = 0.2
MOTOR PWM duty ratio  = 0.0
```

Figure 3.2 Result from Tera-Term (2)

IV Discussion & Conclusion

1. Discussion

A Compiling error appeared when a code that prints 'distance' and 'Motor Duty Ratio' on the 'Tera-term screen' and a code that blinks 'LED' were implemented in a repetition statement, while loop. This problem was judged to be an error in the process of exchanging data in real time, and this problem was solved by using 'Structure' called 'Tick'. By generating 'Interrupt' every given time in the 'main', the variable 'bprint' was 'set' to 1, and printing was periodically performed through 'if statement'.

Second, there were many phrases used repeatedly in the code written this time. In the next lab, I think a more efficient code will be created if the declaration and definition of the function and the use of 'Structure' are used.

Third, there were cases in which the code was not physically implemented even though there was no theoretical error and no compiling error occurred. In fact, the motor that did not work properly in this lab was solved by replacing experimental equipment. For this reason, it was thought that sometimes the possibility of equipment damage should be left open and experimented.

2. Conclusion

In this lab, I learned about Mbed coding and fundamental structure of Arm code.

by controlling angular velocity of Motor and by looking at how the code I written is implemented in real space, I was able to look more intuitively at the code I wrote.

V References

• Yifeng Zhu (2018). Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C Third Edition, E-Man Press LLC

• Young-Keun Kim (2021). https://ykkim.gitbook.io/ec/