Fan Speed Control System Project Report

By Micheal Dominguez and Kevin Wu

Table of Contents

1. Project description and goal:

Our project consists of two main elements: a light switch and a fan. When the light switch is turned on, the fan will automatically turn on, symbolizing that someone has entered the room. Once the light and fan are on, a switch can be used to adjust the speed of the fan depending on the individual's needs. Different combinations of inputs from the switch changes the speed from lowest, low, high, and highest. When the light is turned off, the fan will turn off. Our goal is to create a functioning light switch activated fan that changes speed on the users input.

2. Components:

- **Brief Overview**
  - **Smart Ceiling Fan Control**
    - OFF/ON setting based on brightness.
    - Fan speed based on switches.
  - **User Input**
    - Different switch combination to use different fan mode
      -
- **Components:**
  - L298N Dual Bridge Motor Driver
  - 2-phase, 12V DC bipolar stepper motor (fan)
  - K64F FRDM Board
  - Potentiometer (light switch)
  - DIP Switch Module

3. Hardware Schematics:

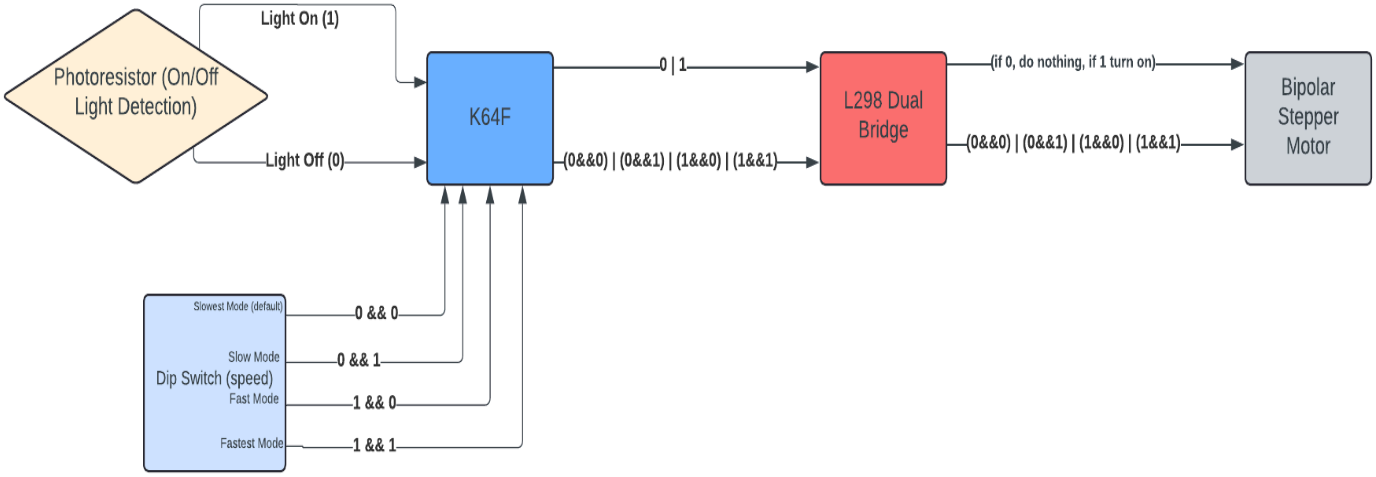

Figure 3.1: Hardware Flowchart

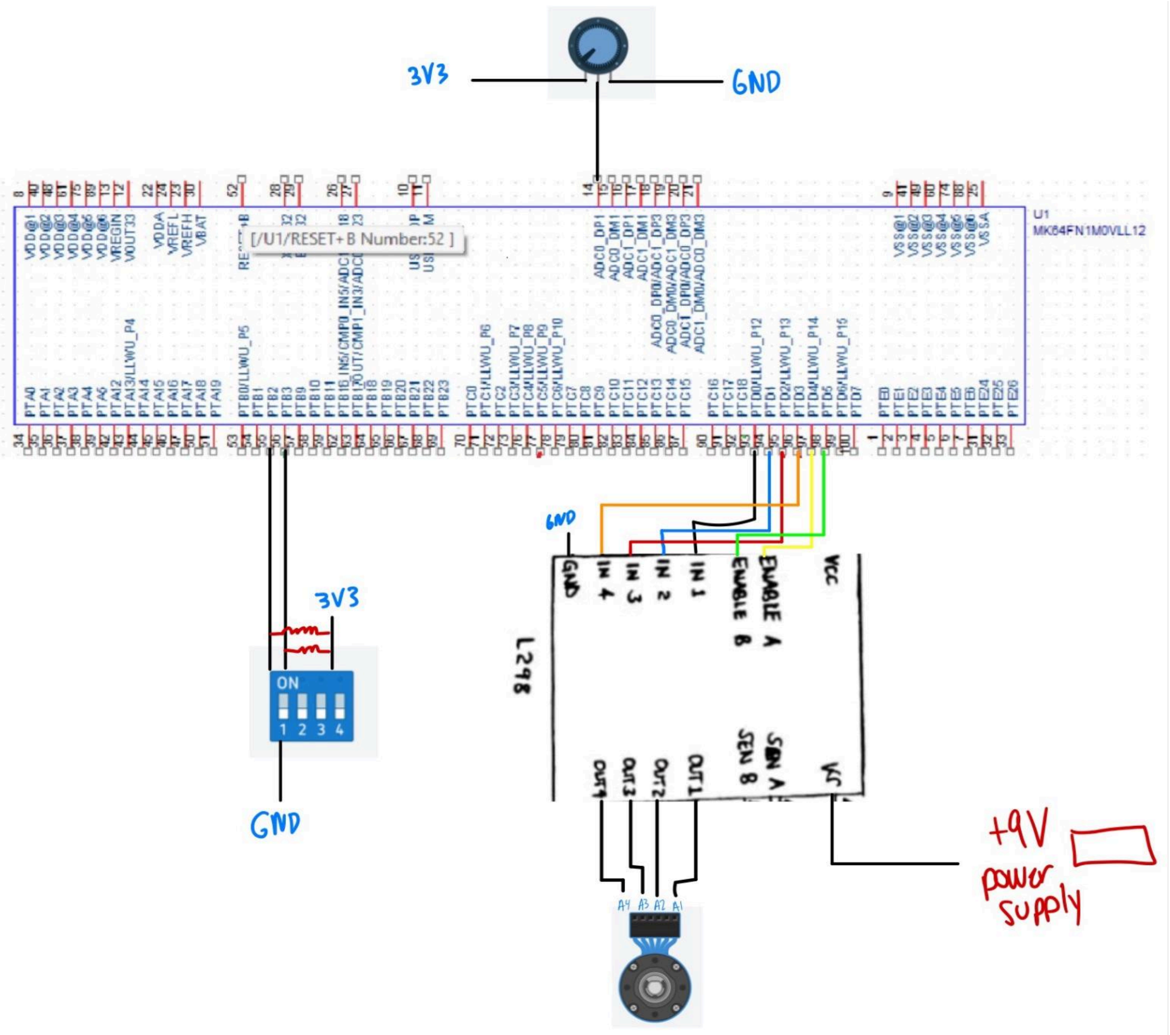

Figure 3.2: Hardware Schematic of FRDM K64F dev. board, potentiometer, switch, stepper motor, L298N motor driver, and 9V power supply

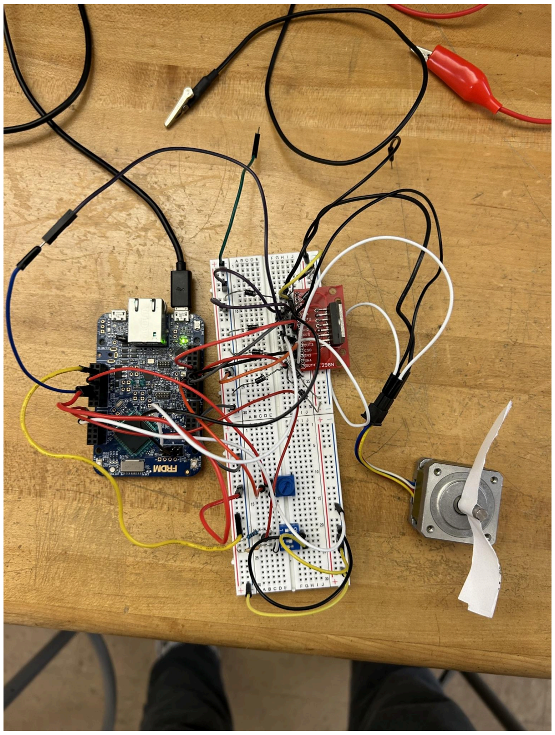

Figure 3.3: Photo of circuit

4. Testing/Evaluation Description:

Our environment within the lab consisted of a voltage supply source that connected to the breadboard and supplied the L298N motor driver with 9+ volts of power. This voltage source was required to power the 12V DC stepper motor. This power supply was required, as the fan would not have been able to turn on without the sufficient amount of voltage.

Link to demo: https://youtu.be/CHEWgq_nAGw

5. Technical Challenges and Improvements:

One of the biggest problems we encountered during this project was the analog to digital converter on the K64F. The first and most complex iteration of our project consisted of the DHT11 Temperature and Humidity module. Despite our best efforts to implement it directly onto the K64F, the lack of a working library to test the module ultimately led us to abandon the DHT. We then moved to a photoresistor. Again, through the use of 3 different photoresistors, we could not get it to read correctly. It would spit out values opposite of the expected. We moved to our last resort which was the potentiometer. We were able to get this one to work after connecting it directly to the ADC0 pin on the K64F. On the software side, we had no significant struggles. Our struggles were normal code building struggles, a matter of stepping through the program multiple times to make sure we did not make a simple error. The biggest improvement we could make to the project would be to go back and try to incorporate the DHT module using the K64F. We could also incorporate other elements such as a door lock or light bulb to make the project more realistic.

6. Brief Conclusion:

Overall, the project was partly successful. We did not get the initial result that were looking for, and the beginning turned out to be more difficult than we anticipated. However, despite the reiterations we made, we had fun putting the project together and having the freedom to figure things out on our own. We furthered our knowledge of the microcontrollers and embedded systems. We were able to work hands on with the different components of the K64F such as its ADC pin, SPI setup, and other components such as the L298N Motor Driver.

Functional Code:

Main.c

```
#include "fsl_device_registers.h"
static int i = 0;
uint32_t light = 0;
uint32_t mode = 0;
void speed(int mode){
	if ((mode & 0x8) != 0 && (mode & 0x04) != 0){ // 1,1
		GPIOD_PDOR = 0x36;
		for(i = 0; i < 10000; i++);
		GPIOD_PDOR = 0x35;
		for(i = 0; i < 10000; i++);
		GPIOD_PDOR = 0x39;
		for(i = 0; i < 10000; i++);
		GPIOD_PDOR = 0x3A;
		for(i = 0; i < 10000; i++);
	}
	else if ((mode & 0x8) != 0 && (mode & 0x04) == 0){ // 1, 0
		GPIOD_PDOR = 0x36;
		for(i = 0; i < 20000; i++);
		GPIOD_PDOR = 0x35;
		for(i = 0; i < 20000; i++);
		GPIOD_PDOR = 0x39;
		for(i = 0; i < 20000; i++);
		GPIOD_PDOR = 0x3A;
		for(i = 0; i < 20000; i++);
	}
	else if ((mode & 0x8) == 0 && (mode & 0x04) != 0){ // 0, 1
		GPIOD_PDOR = 0x36;
		for(i = 0; i < 30000; i++);
		GPIOD_PDOR = 0x35;
		for(i = 0; i < 30000; i++);
		GPIOD_PDOR = 0x39;
		for(i = 0; i < 30000; i++);
		GPIOD_PDOR = 0x3A;
		for(i = 0; i < 30000; i++);
	}
	else if ((mode & 0x8) == 0 && (mode & 0x04) == 0){ // 0, 0
		GPIOD_PDOR = 0x36;
		for(i = 0; i < 40000; i++);
		GPIOD_PDOR = 0x35;
		for(i = 0; i < 40000; i++);
		GPIOD_PDOR = 0x39;
		for(i = 0; i < 40000; i++);
		GPIOD_PDOR = 0x3A;
		for(i = 0; i < 40000; i++);
	}
}

unsigned short adc_read16b(){
	//ADC0_SC3 = 0x07;
	ADC0_SC1A = 0x00;
	while(ADC0_SC2 & ADC_SC2_ADACT_MASK);
	while(!(ADC0_SC1A & ADC_SC1_COCO_MASK));
	return ADC0_RA; //31539, 32534
}

void PORTA_IRQHandler(void)
```

```
{
       mode = GPIOB_PDIR; // read port b(switch)
       light = (adc_read16b()* 33) / 0xFFFF; //read ADC value and convert to decimal
       if(light > 16){
               speed(mode);
       }
       else{
               //do nothing
       }
       //Clear ISFR
       PORTA_ISFR = (1 << 1);

}


int main(void)
{
       SIM_SCGC5 |= SIM_SCGC5_PORTA_MASK; /*Enable Port A Clock Gate Control*/
    SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK; /*Enable Port B Clock Gate Control*/
    SIM_SCGC5 |= SIM_SCGC5_PORTD_MASK; /*Enable Port D Clock Gate Control*/
    SIM_SCGC6 |= SIM_SCGC6_ADC0_MASK;  /*Enable ADC*/

    //port b
    PORTB_GPCLR = 0x000C0100; /*Configure Port B Pin 2-3 for GPIO*/
    GPIOB_PDDR = 0x00000000; /*Configure Port B Pin 2-3 for Input*/


    //Port D
    PORTD_GPCLR = 0x00FF0100; /*Configure Port D Pins 0-7 for GPIO*/
       GPIOD_PDDR = 0x000000FF; /*Configure Port D Pins 0-7 for Output*/
       GPIOD_PDOR = 0x01; /*Initialize Port D such that only 1 bit is ON*/

    PORTA_PCR4 = 0x0100;  //port A

    //Configure PA1 to trigger interrupts on falling edge input.
    //PORTC_PCR1 = 0xA0100;
    //Configure ADC for 16 bits, and to use bus clock.
    ADC0_CFG1 = 0x0C;
    //Disable the ADC module;
    ADC0_SC1A = 0x1F;
    //Set PB[3:2] and PA[1] for input;
    GPIOA_PDDR |= (0 << 1);  // PC[1] input
    GPIOB_PDDR |= ((0 << 3) | (0 << 2) | (1 <<2) | (1 << 10)); // PB[3:2] input


    PORTA_ISFR = (1 << 1);
    //NVIC_EnableIRQ(PORTA_IRQn);

    for(;;){
       GPIOB_PTOR |= (1 << 10);
       PORTA_IRQHandler();
    }
    return 0;
}
```