University of California, Riverside

Smart Fan System

EE 128 Section 21, Fall 2025

Name: Muneer Al Jufout

Student ID: 862410258

Lab Partner: Sameer Anjum

Lab Partner ID: 862422332

TA: Johnson Zhang

Intro/Abstract

This project contains a Smart Fan system built on the K64F using a level-edge interrupt-driven design. The fan automatically rotates based on motion and temperature by reading a PIR sensor, ML-style motion prediction, temperature processing, and motor control. When motion is detected with enough confidence built in the code, the stepper motor turns for a minimum runtime, and its speed changes based on temperature thresholds. Safety features such as smoke detection are included.

Experiment System Specification

The Smart Fan system operates on the K64F, utilizing an Interrupt timer to trigger all sensor readings and fan behavior. The PIR motion sensor is connected to Port B, the smoke sensor to Port C, and the temperature is read through the internal ADC. Each time the interrupt cycles, it updates the built-in motion buffer, calculates confidence, reads the temperature, checks for smoke detection, and adjusts the fan speed or activation accordingly. The stepper motor is driven through pins on Port A and B.

Flowchart

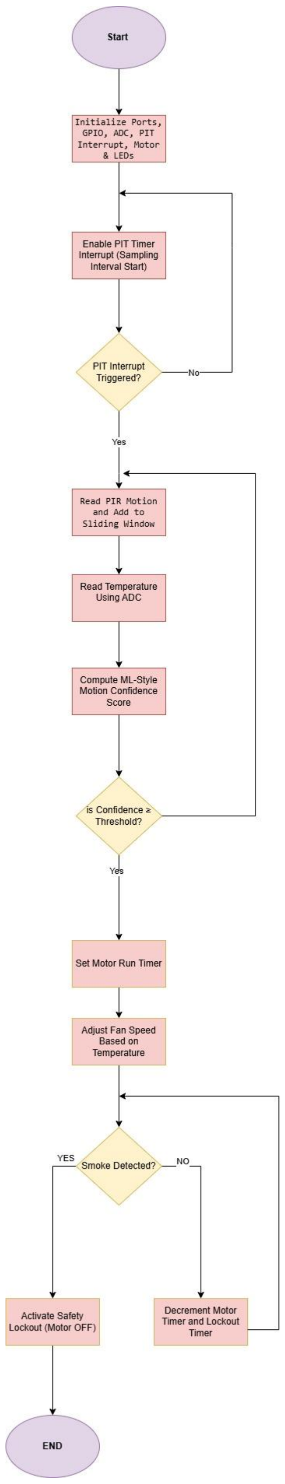
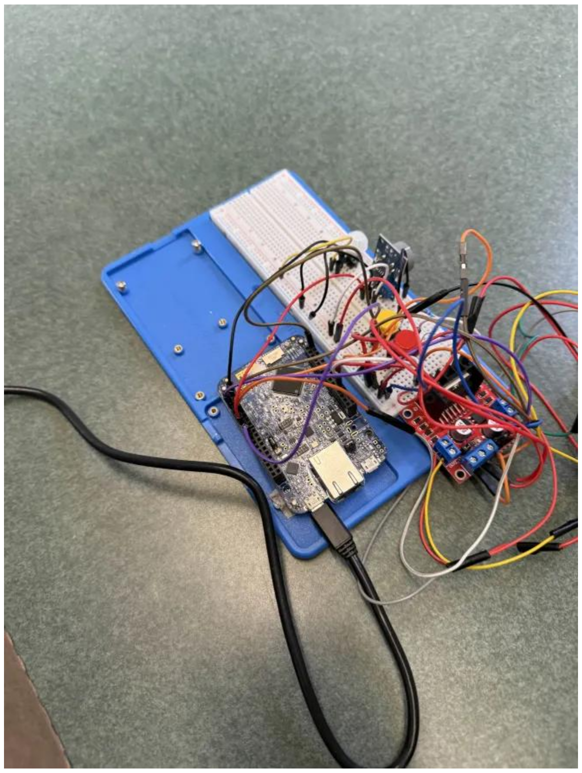

Start
Initialize Ports, GPIO, ADC, PIT Interrupt, Motor & LEDs
Enable PIT Timer Interrupt (Sampling Interval Start)
PIT Interrupt Triggered?
No
Yes
Read PIR Motion and Add to Sliding Window
Read Temperature Using ADC
Compute ML-Style Motion Confidence Score
is Confidence ≥ Threshold?
Yes
Set Motor Run Timer
Adjust Fan Speed Based on Temperature
Smoke Detected?
YES
NO
Activate Safety Lockout (Motor OFF)
Decrement Motor Timer and Lockout Timer
END

Fig. 1: The Program Flowchart

Hardware Design


a)

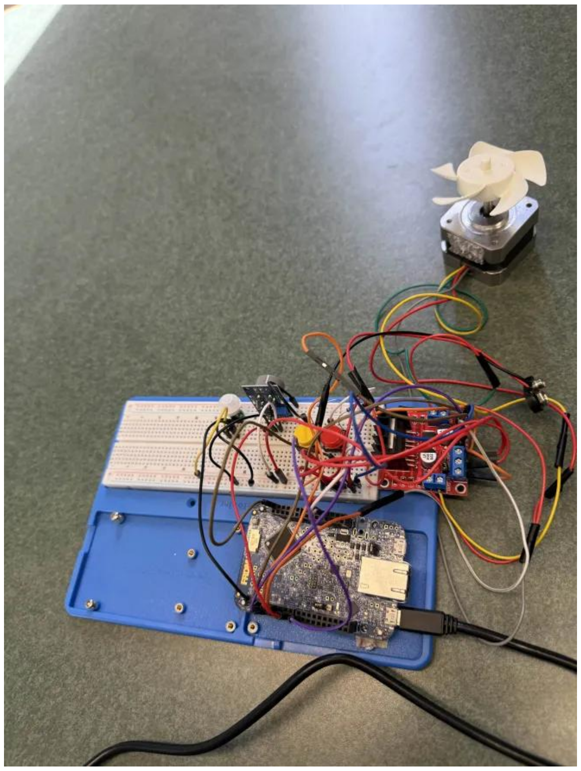

b)

Fig. 2: Photos of the Setup

Schematic Diagram

This schematic included:

- MK64FN1M0VLL12 microcontroller (FRDM-K64F)
- Ground (GND)
- PIR Motion Sensor
- MQ-2 Smoke Detector
- Stepper Motor

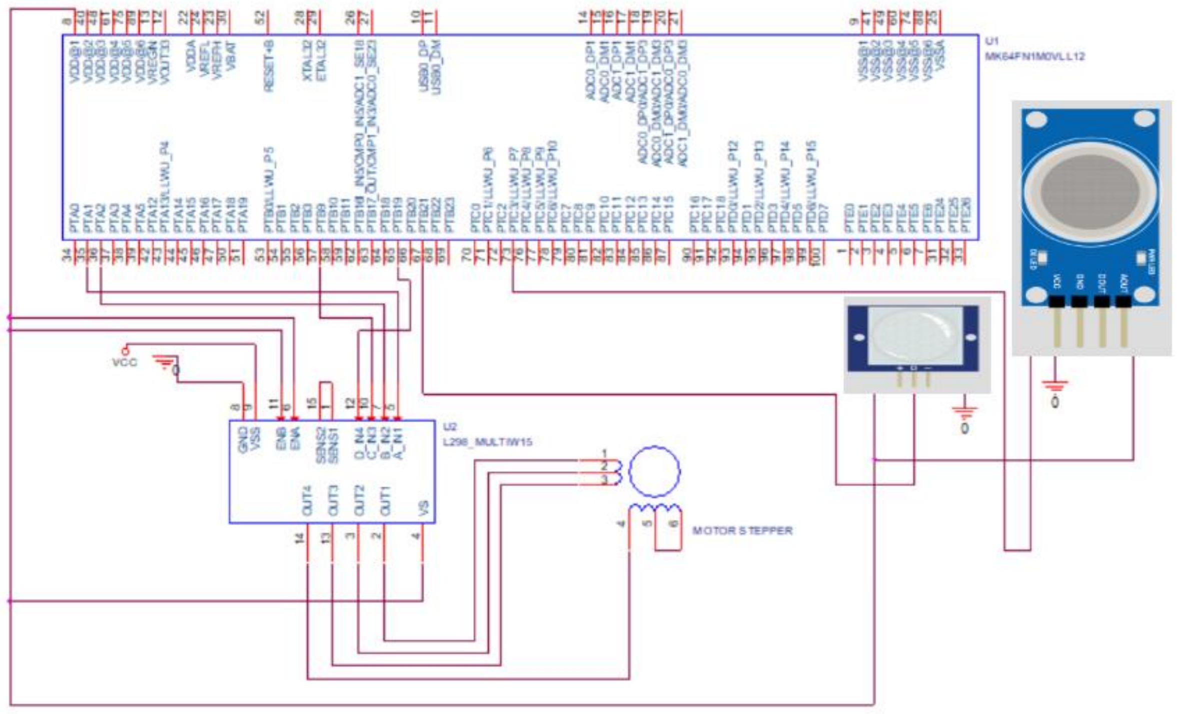


Fig. 3: Schematic Diagram

High-Level Description

During each interrupt, the system reads the PIR motion sensor, checks the smoke input, and samples temperature using the ADC. The ML logic begins to run after, essentially waiting and an interrupt trigger when motion is detected. Safety logic immediately disables the fan if smoke is detected. The main loop remains idle, while all sensing and decision-making occur automatically inside the interrupt.

Program Listing (Main Sections)

FRDM-K64F Code

```
#include "fsl_device_registers.h"

void delay_loops(int n){
    for(volatile int i=0; i<n; i++);
}
```

```
void init_led(void)
{
   SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK | SIM_SCGC5_PORTE_MASK;

   PORTB_PCR22 = PORT_PCR_MUX(1);
   GPIOB_PDDR |= (1 << 22);
   GPIOB_PSOR = (1 << 22);

   PORTE_PCR26 = PORT_PCR_MUX(1);
   GPIOE_PDDR |= (1 << 26);
   GPIOE_PSOR = (1 << 26);

   PORTB_PCR21 = PORT_PCR_MUX(1);
   GPIOB_PDDR |= (1 << 21);
   GPIOB_PSOR = (1 << 21);
}

void led_red_on(void)    { GPIOB_PCOR = (1 << 22); }
void led_red_off(void)   { GPIOB_PSOR = (1 << 22); }
void led_green_on(void)  { GPIOE_PCOR = (1 << 26); }
void led_green_off(void) { GPIOE_PSOR = (1 << 26); }
void led_blue_on(void)   { GPIOB_PCOR = (1 << 21); }
void led_blue_off(void)  { GPIOB_PSOR = (1 << 21); }

void all_leds_off(void)
{
   led_red_off();
   led_green_off();
   led_blue_off();
}

// Initialize stepper motor control pins
void init_motor(void){
   SIM_SCGC5 |= SIM_SCGC5_PORTA_MASK | SIM_SCGC5_PORTB_MASK;
   PORTA_PCR1  = PORT_PCR_MUX(1);
   PORTA_PCR2  = PORT_PCR_MUX(1);
   PORTB_PCR9  = PORT_PCR_MUX(1);
   PORTB_PCR19 = PORT_PCR_MUX(1);
   GPIOA_PDDR |= (1 << 1) | (1 << 2);
   GPIOB_PDDR |= (1 << 9) | (1 << 19);
}

void motor_stop(void)
{
   GPIOA_PCOR = (1 << 1) | (1 << 2);
   GPIOB_PCOR = (1 << 9) | (1 << 19);
}

// Different cases for stepper motor coils (0-3)
void motor_step(int step_position)
{
    switch(step_position)
    {
        case 0:
            GPIOA_PSOR = (1 << 1);
            GPIOA_PCOR = (1 << 2);
            GPIOB_PSOR = (1 << 9);
            GPIOB_PCOR = (1 << 19);
            break;
        case 1:
            GPIOA_PCOR = (1 << 1);
            GPIOA_PSOR = (1 << 2);
            GPIOB_PSOR = (1 << 9);
            GPIOB_PCOR = (1 << 19);
            break;
        case 2:
            GPIOA_PCOR = (1 << 1);
            GPIOA_PSOR = (1 << 2);
            GPIOB_PCOR = (1 << 9);
            GPIOB_PSOR = (1 << 19);
            break;
        case 3:
            GPIOA_PSOR = (1 << 1);
            GPIOA_PCOR = (1 << 2);
            GPIOB_PCOR = (1 << 9);
            GPIOB_PSOR = (1 << 19);
            break;
    }
}

#define WINDOW_SIZE 20
#define THRESHOLD 0.45f
#define MIN_RUNTIME 200

typedef struct {
    float data[WINDOW_SIZE];
    int index;
    int count;
} SensorWindow;

SensorWindow sens_buf = {{0}, 0, 0};
volatile int motor_timer = 0;
volatile int pred_count = 0;
volatile unsigned short temp_val = 0;
volatile int speed_mode = 0;
volatile int lockout_timer = 0;

// Add reading to circular buffer
void add_sensor_reading(float value)
{
    sens_buf.data[sens_buf.index] = value;
    sens_buf.index = (sens_buf.index + 1) % WINDOW_SIZE;
    if (sens_buf.count < WINDOW_SIZE)
    {
        sens_buf.count++;
    }
}

// Calculate motion confidence using mean, variance, and transitions
float predict_motion(void)
{
    if (sens_buf.count < WINDOW_SIZE)
    {
        return 0.0f;
    }

    // Weighted moving average
    float wsum = 0.0f;
    float wtotal = 0.0f;
    for (int i = 0; i < WINDOW_SIZE; i++)
    {
        int idx = (sens_buf.index - WINDOW_SIZE + i + WINDOW_SIZE) % WINDOW_SIZE;
        float w = (float)(i + 1) / WINDOW_SIZE;
        wsum += sens_buf.data[idx] * w;
        wtotal += w;
    }
    float wma = wsum / wtotal;

    // Variance
    float avg = 0.0f;
    for (int i = 0; i < WINDOW_SIZE; i++)
    {
        avg += sens_buf.data[i];
    }
    avg /= WINDOW_SIZE;

    float var = 0.0f;
    for (int i = 0; i < WINDOW_SIZE; i++)
    {
        float d = sens_buf.data[i] - avg;
        var += d * d;
    }
    var /= WINDOW_SIZE;

    // Transition count
    int trans = 0;
    for (int i = 1; i < WINDOW_SIZE; i++)
    {
        int prev = (sens_buf.index - WINDOW_SIZE + i - 1 + WINDOW_SIZE) % WINDOW_SIZE;
        int curr = (sens_buf.index - WINDOW_SIZE + i + WINDOW_SIZE) % WINDOW_SIZE;
        if (sens_buf.data[prev] != sens_buf.data[curr])
        {
            trans++;
        }
    }
    float trans_score = (float)trans / (WINDOW_SIZE - 1);

    // Weighted combination
    float conf = (wma * 0.5f) + (var * 0.3f) + (trans_score * 0.2f);

    return conf;
}

void init_adc(void)
{
    SIM_SCGC6 |= SIM_SCGC6_ADC0_MASK;
    ADC0_CFG1 = 0x0C;
    ADC0_SC1A = 0x1F;
}

unsigned short ADC_read16b(void)
{
    ADC0_SC1A = 0x1A;
    while(!(ADC0_SC1A & ADC_SC1_COCO_MASK));
    return ADC0_RA;
}

// Initialize PIR sensor (PTB18) and smoke detector (PTC3)
void init_inputs(void)
{
    SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK | SIM_SCGC5_PORTC_MASK;

    PORTB_PCR18 = PORT_PCR_MUX(1) | PORT_PCR_PE_MASK | PORT_PCR_PS_MASK;
    GPIOB_PDDR &= ~(1 << 18);

    PORTC_PCR3 = PORT_PCR_MUX(1);
    GPIOC_PDDR &= ~(1 << 3);
}

static inline int smoke_detected(void)
{
    int val = (GPIOC_PDIR >> 3) & 1;
    return !val;
}

void init_pit(void)
{
    SIM_SCGC6 |= SIM_SCGC6_PIT_MASK;
    PIT_MCR = 0x00;
    PIT_LDVAL0 = 600000;
    PIT_TCTRL0 = PIT_TCTRL_TIE_MASK;
    NVIC_EnableIRQ(PIT0_IRQn);
    PIT_TCTRL0 |= PIT_TCTRL_TEN_MASK;
}
```

```
// Set K64F LED color based on temperature
void set_temperature_led(unsigned short adc_value)
{
    all_leds_off();

    if (adc_value < 12800)
    {
        led_red_on();
    }
    else if (adc_value > 14000)
    {
        led_blue_on();
    }
    else
    {
        led_green_on();
    }
}

// Periodic interrupt: sample sensors, run motion prediction, control motor
void PIT0_IRQHandler(void)
{
    PIT_TFLG0 = PIT_TFLG_TIF_MASK;

    if (lockout_timer > 0)
    {
        lockout_timer--;
    }

    // Smoke detection triggers safety lockout
    if (smoke_detected())
    {
        lockout_timer = 1000;
        motor_timer = 0;
    }

    int motion = (GPIOB_PDIR >> 18) & 1;
    add_sensor_reading((float)motion);

    // Read temperature every 100 ticks
    static int temp_cnt = 0;
    temp_cnt++;
    if (temp_cnt >= 100)
    {
        temp_cnt = 0;
        temp_val = ADC_read16b();

        if (temp_val < 12800)
        {
            speed_mode = 1;
        }
        else
        {
            speed_mode = 0;
        }

        set_temperature_led(temp_val);
    }

    // Run motion prediction every 50 clock ticks
    if (lockout_timer == 0)
    {
        pred_count++;
        if (pred_count >= 50)
        {
            pred_count = 0;

            float conf = predict_motion();

            if (conf >= THRESHOLD)
            {
                motor_timer = MIN_RUNTIME;
            }
        }
    }
    else
    {
        pred_count = 0;
    }

    if (motor_timer > 0)
    {
        motor_timer--;
    }
}

int main(void)
{
    init_led();
    init_motor();
    init_inputs();
    init_adc();
    init_pit();
    motor_stop();

    __enable_irq();

    int step_pos = 0;
    int delay_slow = 200000;
    int delay_fast = 17000;
    int curr_delay;
    led_green_on();

    while(1)
    {
        if (speed_mode == 1)
        {
            curr_delay = delay_fast;
        }
        else
        {
            curr_delay = delay_slow;
        }

        // Stop motor during safety lockout
        if (lockout_timer > 0)
        {
            motor_stop();
            delay_loops(curr_delay);
            continue;
        }

        if (motor_timer > 0)
        {
            motor_step(step_pos);
            step_pos = (step_pos + 1) & 3;
        }
        else
        {
            motor_stop();
        }

        delay_loops(curr_delay);
    }
}
```

Technical Problems Encountered and Solutions

One major technical problem we encountered was when we could not make the K64F internal temperature work. We realized that a processor expert was not needed, which was causing most of our errors. Another problem was that we tried doing a different kind of ML model with Edge Impulse. We could not use it, however, because the KDS compiler would not let us compile it, so we opted for making an ML-like model.

Conclusions

In this project, the program successfully read motion, temperature, and smoke inputs with level edge interrupts and used ML-style algorithm to detect motion accurately. The fan responded correctly to motion and temperature changes, and the safety lockout behaved as expected when smoke was detected. Overall, the system operated well in real time, demonstrating the use of interrupts, sensor processing, and stepper motor control using a K64F.

Team Contribution Summary

We worked together on all parts of the project. I (Muneer) contributed to building the circuit, writing part of the report, and assisting with system testing. Sameer worked on the software implementation and another portion of the report. Both of us participated in debugging, verifying the system, and ensuring the Smart Fan operated correctly.