TEMPERATURE FAN CONTROL
EEI28 FINAL PROJECT

Ryan Bordadora, Karam Shanti

ABOUT THE PROJECT

BreezeControl is an innovative project aimed at developing a smart fan that can be conveniently controlled through a dedicated mobile application. This cutting-edge fan combines state-of-the-art technology with user-friendly features to create a comfortable and personalized airflow experience. The project will focus on designing and prototyping a smart fan, developing an intuitive mobile application, and integrating the fan with various smart home ecosystems.

WHAT IT CAN DO

MANUAL PERSONALIZATION

Allows users to create personalized settings, such as preferred speed levels, catering to their specific comfort needs.

AUTOMATIC CLIMATE

Incorporates a smart sensor to detect room temperature. The fan will adjust its speed and airflow based on these inputs, reducing unnecessary energy consumption.

MONITORING

Provide real-time statistics, allowing users to track and manage their fan's energy usage compared to the current temperature..

THE PROCESS

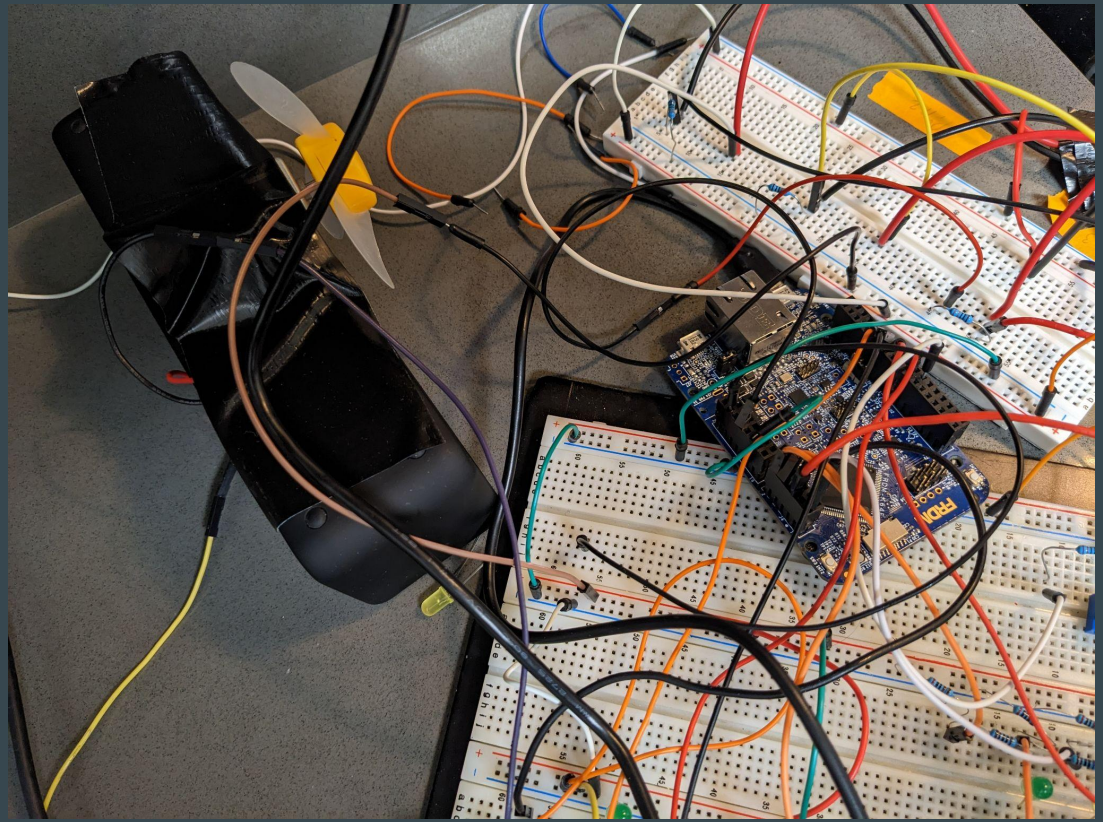
How the system converts information into data and action.

PARTS + COMPLEXITIES

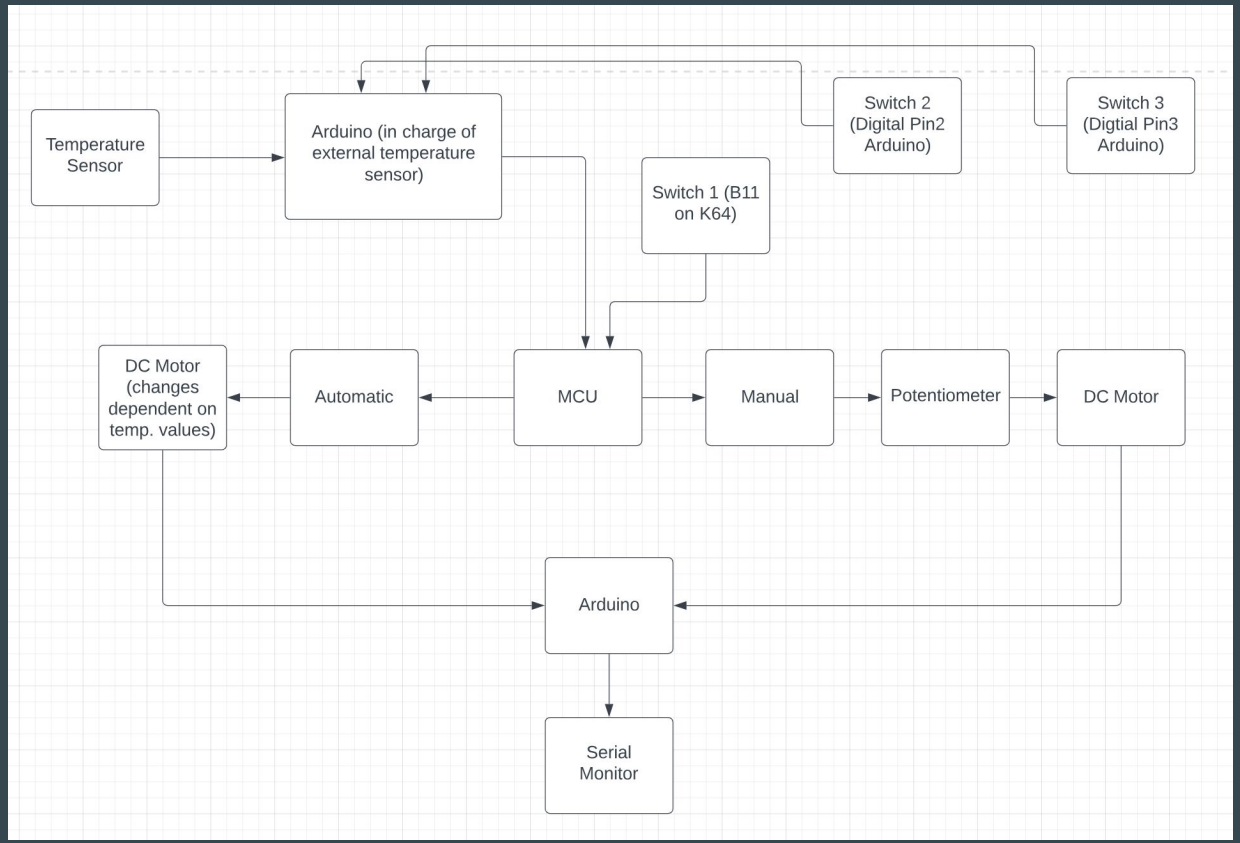
- K64F MICROCONTROLLER
- ARDUINO UNO (2X)
- 3 SWITCHES
- DC MOTOR
- BJT
- 3 LEDS (REPRESENT TEMP VALUES)

- GPIO
- SPI COMMUNICATION
- TIMER FUNCTION / PWM
- ADC

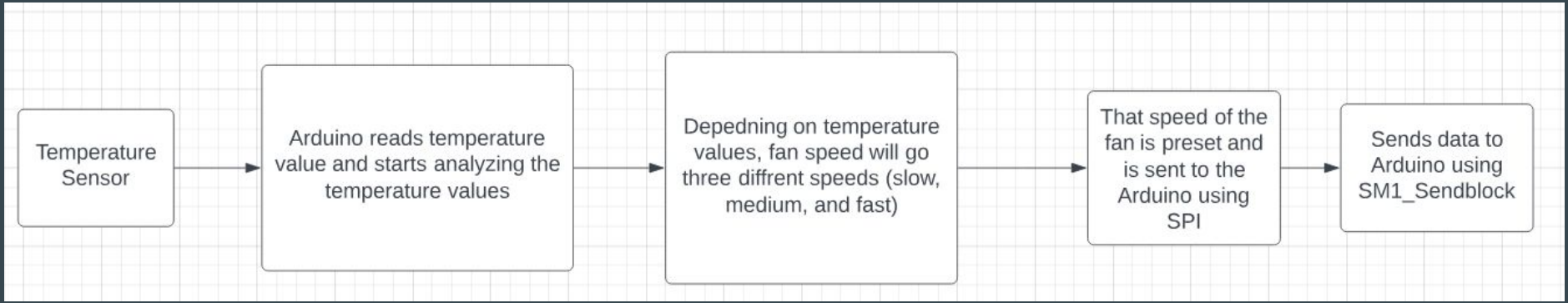
THERMOFAN...



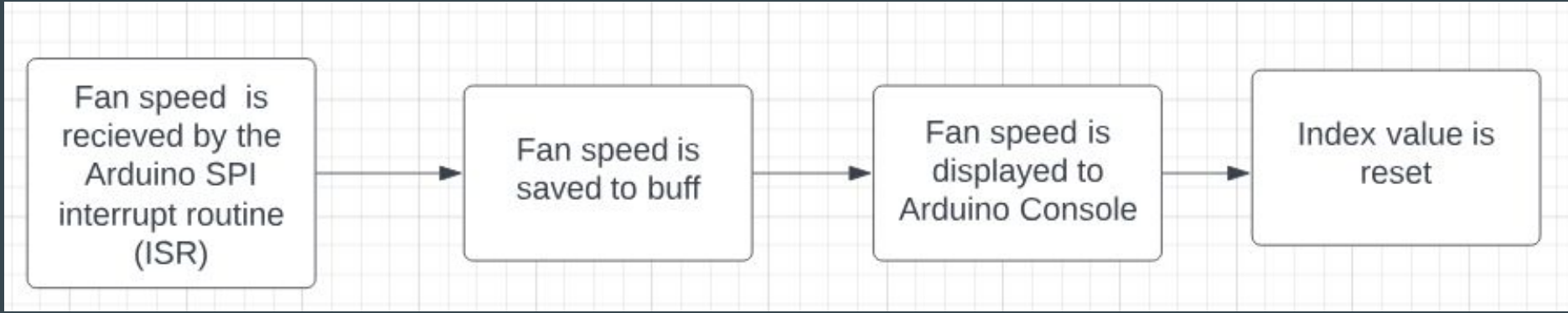
SO HOW DOES IT WORK...



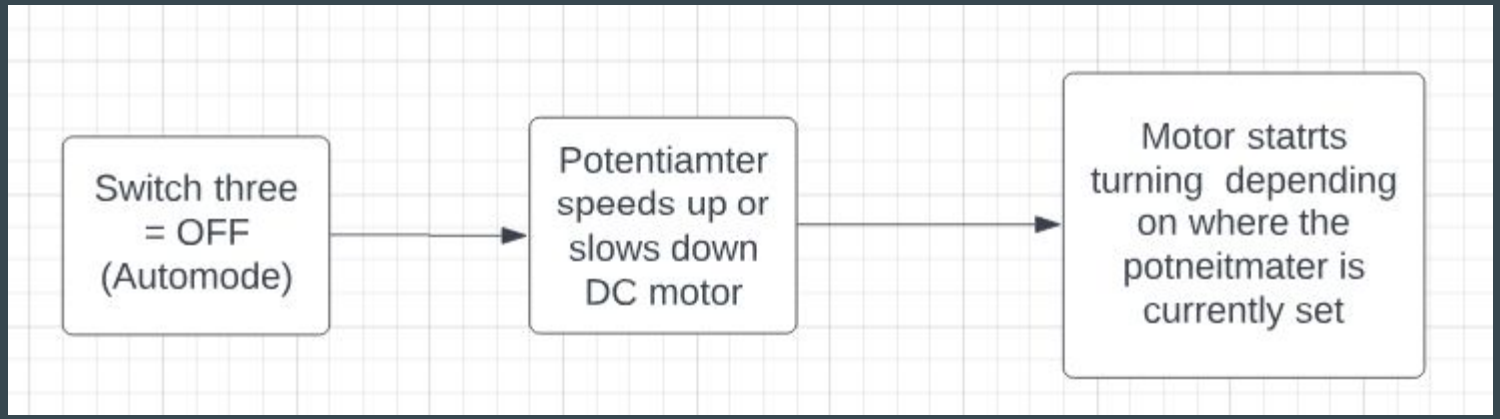
TEMPERATURE SENSOR -> MCU...



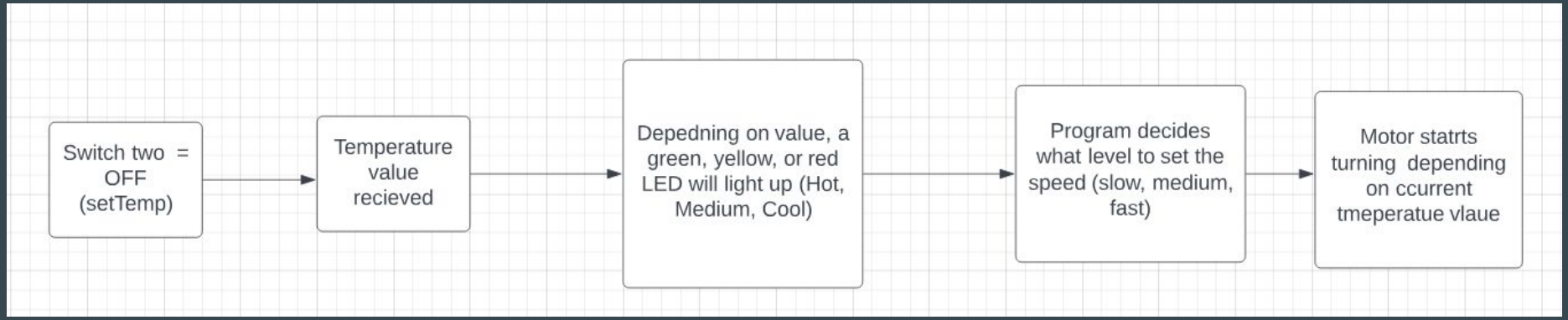
MCU -> ARDUINO...


Fan speed is recieved by the Arduino SPI interrupt routine (ISR)
Fan speed is saved to buff
Fan speed is displayed to Arduino Console
Index value is reset

DC MOTOR (MANUAL MODE)



DC MOTOR (AUTOMATIC MODE)



VIDEO DEMO

https://youtu.be/3p-t7ZpaLaU

CODE (ARDUINO)...

```
/* Arduino UNO with SSD1306 128x64 I2C OLED display and 10K Ohm pot on A0
 * Tony Goodhew  4 June 2020
 */
#include <SPI.h>  // Not needed if device is I2C
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
#define OLED_RESET     4 // Reset pin # (or -1 if sharing Arduino reset pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

int sensorPin = A0;   // select the input pin for the potentiometer
int ledPin = 13;      // select the pin for the LED
int sensorValue = 0;  // variable to store the value coming from the sensor
int counter = 0;      // Seconds counter
unsigned long int currentTime,nextTime;



// NEWWWW CODEEEEEEEEEEE
char buff [255];
volatile byte indx;
volatile boolean process;
int pot = A0;

void setup() {
  Serial.begin(9600);
  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3D for 128x64
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  // declare the ledPin as an OUTPUT:
  pinMode(ledPin, OUTPUT);
  }
  display.clearDisplay();
  display.setTextSize(1);             // Normal 1:1 pixel scale
  display.setTextColor(SSD1306_WHITE);        // Draw white text
  display.setCursor(0,0);             // Start at top-left corner
  display.println("Smart Fan");
  display.setCursor(80,34);
  display.println("Turn Pot");
  display.setTextSize(2);             // Draw 2X-scale text
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(80,5);
  //counter = 0;
  //align3(counter);
  display.drawRect(0, 10, 70, 35, SSD1306_WHITE);  // RAW pot value box
  //nextTime = millis() + 1000;  // 1 second in the future
  //display.println(counter);
  display.display();
```

```
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(80,5);
  //counter = 0;
  //align3(counter);
  display.drawRect(0, 10, 70, 35, SSD1306_WHITE);  // RAW pot value box
  //nextTime = millis() + 1000;  // 1 second in the future
  //display.println(counter);
  display.display();




// NEWWWW CODEEEEEEEEEEE
  pinMode(MISO, OUTPUT); // have to send on master in so it set as output
  SPCR |= _BV(SPE); // turn on SPI in slave mode
  indx = 0; // buffer empty
  process = false;
  SPI.attachInterrupt(); // turn on interrupt
  pinMode(A0, INPUT);
}




// Functions for right alignment of integers
int align2(int q){  // Space in 100s and 10s
  if (q < 100) {display.print(" ");}
    if (q < 10) {display.print(" ");}
}
int align3(int q){  // Space in 1000s
  if (q < 1000) {display.print(" ");}
  align2(q);
}




// NEWWWW CODEEEEEEEEEEE
ISR (SPI_STC_vect) // SPI interrupt routine
{
  byte c = SPDR; // read integer value from SPI Data Register
  if (indx < sizeof(buff)) {
    buff[indx++] = c;

    if (c == '\n') { // save value in the next index in the array buff
      buff[indx - 1] = 0;
      process = true;
    }


    }
  }
```

```
// ++++++++++++Main Loop ++++++++++++++
// 1 second timer
void loop() {

 // MOST OF THE CODE IN THIS FUNCTION WAS FOR WHEN WE WERE USING A LCD, LINE 196 AND BELOW IS WHERE THE OUTPUTTING OF THE SERIAL DATA STARTS
 /*
  currentTime = millis();
  if (currentTime >= nextTime) {  // Then update one   second timer
    display.setTextSize(2);             // Draw 2X-scale text
    display.setTextColor(SSD1306_BLACK);
    display.setCursor(80,5);
    align3(counter);          // Right align value
    display.println(counter); // Overwrite current value in black - rub out
    counter = counter + 1;
    if (counter >= 10000){counter = 0;} // Reset befor overflows space
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(80,5);
    align3(counter);
    display.println(counter);  // Write new value in white
    nextTime = nextTime + 1000;
  }

  */
 // Read pot and display values
 sensorValue = analogRead(A0); // read the value from the sensor
//  int perCent = float(buff[31] * 100.0/1018);
 int x = analogRead(pot);
 int value = map(x, 0, 1023, 0, 120);
 int perCent;
// Serial.println("HAHAHAHHA");
// Serial.println(value);

 if (buff[31] == '0') {
  perCent = 0;
 }

 else if (buff[31] == '1') {
  perCent = 10;
}

else if (buff[31] == '2') {
  perCent = 20;
}

else if (buff[31] == '3') {
  perCent = 30;
}

else if (buff[31] == '4') {
  perCent = 40;
```

CODE (ARDUINO) CONTINUED...

```
  perCent = 40;
}

else if (buff[31] == '5') {
  perCent = 50;
}

else if (buff[31] == '6') {
  perCent = 60;
}

else if (buff[31] == '7') {
  perCent = 70;
}

else if (buff[31] == '8') {
  perCent = 80;
}

else if (buff[31] == '9') {
  perCent = 99;
}

  display.drawLine(0,47,0,62, SSD1306_WHITE);
  display.fillRect(1, 50, perCent, 10, SSD1306_WHITE);
  display.setTextSize(1);              // Normal 1:1 pixel scale
  display.setTextColor(SSD1306_WHITE);  // Draw white text
  display.setCursor(102,50);
  align2(perCent);
  display.print(perCent);
  display.println("%");
  display.fillRect(1, 11, 68, 33, SSD1306_BLACK); // Clear box
  display.setCursor(10,20);
  display.setTextSize(2);
  align3(sensorValue);
  delay(200);
  display.setTextSize(1);
  display.display();
  delay(100);
  display.fillRect(1, 50, perCent, 10, SSD1306_BLACK);
  display.setTextColor(SSD1306_BLACK);
  display.setCursor(102,50);
  align2(perCent);
  display.print(perCent);
  display.println("%");

  if (process) {
    process = false;
    Serial.println (buff);
    delay(10);
    indx= 0;
  }
}
```

CODE (K64F)...

```
**     Filename    : main.c
/*!
 ** @file main.c
 ** @version 01.01
 ** @brief
 **         Main module.
 **         This module contains user's application code.
 */
/*!
 **  @addtogroup main_module main module documentation
 **  @{
 */
/* MODULE main */


/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.h"
#include "Pins1.h"
#include "FX1.h"
#include "GI2C1.h"
#include "WAIT1.h"
#include "CI2C1.h"
#include "CsIO1.h"
#include "IO1.h"
#include "MCUC1.h"
#include "SM1.h"
/* Including shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"
#include "PDD_Includes.h"
#include "Init_Config.h"
#include "MK64F12.h"
// #include "fsl_device_registers.h"

/* User includes (#include below this line is not maintained by Processor Expert) */

/*lint -save  -e970 Disable MISRA rule (6.3) checking. */

unsigned char write[512];

unsigned short ADC_read16b(void) {
    ADC0_SC1A = 0x00; // Write to SC1A to start conversion from ADC_0
    while(ADC0_SC2 & ADC_SC2_ADACT_MASK); // Conversion in progress
    while(!(ADC0_SC1A & ADC_SC1_COCO_MASK)); //until conversion complete
    return ADC0_RA;
}
```

```
unsigned short ADC_read16b(void) {
    ADC0_SC1A = 0x00; // Write to SC1A to start conversion from ADC_0
    while(ADC0_SC2 & ADC_SC2_ADACT_MASK); // Conversion in progress
    while(!(ADC0_SC1A & ADC_SC1_COCO_MASK)); //until conversion complete
    return ADC0_RA;
}


unsigned short ADC2_read16b(void) {
    ADC1_SC1A = 0x00; // Write to SC1A to start conversion from ADC1_DP0
    while(ADC1_SC2 & ADC_SC2_ADACT_MASK); // Conversion in progress
    while(!(ADC1_SC1A & ADC_SC1_COCO_MASK)); // Wait until conversion complete
    return ADC1_RA;
}



uint8_t FX1_GetTemperature(int8_t *temperature)
{
  int16_t value1;
  int16_t value2;
  int8_t temp;

  if (GI2C1_ReadByteAddress8(FX1_I2C_ADDR, FX1_DIE_TEMP, (uint8_t*)&temp) != ERR_OK) {
    return ERR_FAILED;
  }
   // *temperature = (int8_t)(temp+FX1_DIE_TEMP_OFFSET);
   *temperature = 19;

   if (GPIOD_PDIR & 0x01) {
    //value1 = ADC_read16b();
    //value2 = value1 * 33 / 0xFFFF;
    *temperature++;
   }

    // *temperature = 10;
  return ERR_OK;
}

// NEW CODEEEE


void timer_init() { //Starts timer for delay function
    SIM_SCGC3 |= SIM_SCGC3_FTM3_MASK; // FTM3 clock enable
    FTM3_MODE = 0x5; // Enable FTM3
    FTM3_MOD = 0xFFFF;
    FTM3_SC = 0x0E; // System clock / 64
```

CODE (K64F) CONTINUED...

```c
void timer_init() { //Starts timer for delay function
    SIM_SCGC3 |= SIM_SCGC3_FTM3_MASK; // FTM3 clock enable
    FTM3_MODE = 0x5; // Enable FTM3
    FTM3_MOD = 0xFFFF;
    FTM3_SC = 0x0E; // System clock / 64
}
void delayby1ms(int k) { //Delays time by K milliseconds
    FTM3_C6SC = 0x1C; // Output-compare; Set output
    FTM3_C6V = FTM3_CNT + 333; // 1ms
    for (int i = 0; i < k; i++) {
        while(!(FTM3_C6SC & 0x80));
        FTM3_C6SC &= ~(1 << 7);
        FTM3_C6V = FTM3_CNT + 333; // 1ms
    }
}

void outputCompare(unsigned long H, unsigned long L, int k, int tog) {
    if (tog != 0) {
        for (unsigned int i = 0; i < k; i++) {
            FTM3_C6SC = 0x1C; // Output-compare; Set output
            FTM3_C6V = FTM3_CNT + L;//466; // 300 us LOW
            while(!(FTM3_C6SC & 0x80));
            FTM3_C6SC &= ~(1 << 7);
            FTM3_C6SC = 0x18; // Output-compare; Clear output
            FTM3_C6V = FTM3_CNT + H; //200; // 700 us HIGH
            while(!(FTM3_C6SC & 0x80));
            FTM3_C6SC &= ~(1 << 7);
        }
    }
}
```

```c
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{

    SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK; /*Enable Port B Clock Gate Control*/
    SIM_SCGC5 |= SIM_SCGC5_PORTD_MASK;

    // New Codeeee
    SIM_SCGC5 |= SIM_SCGC5_PORTC_MASK; /*Enable Port C Clock Gate Control*/
    SIM_SCGC3 |= SIM_SCGC3_FTM3_MASK; //   FTM3 clock enable
    PORTC_PCR10 = 0x300; // Port C Pin 10 as FTM3_CH6 (ALT3)
    FTM3_MODE = 0x5; // Enable FTM3
    FTM3_MOD = 0xFFFF;
    FTM3_SC = 0x0D; // System clock / 32


    PORTB_GPCLR = 0x0C0C0100; /*Port B, Pins 2-3, 10-11 configured as Alternative 1 (GPIO)*/
    PORTC_GPCLR = 0x01BF0100; /*Port C, Pins 0-5. 7-8, configured as Alternative 1 (GPIO)*/
    PORTD_GPCLR = 0x00FF0100; /*Port D, Pins 0-7, configured as Alternative 1 (GPIO)*/

    GPIOB_PDDR = 0x00000000; /*Sets all port B pins to Input*/
    GPIOC_PDDR = 0x000001BF; /*Sets Port C, Pins 0-5, 7-8, as Output*/
    GPIOD_PDDR = 0x000000FF; /*Sets Port D, Pins 0-7, as Output*/




    PORTB_GPCLR = 0x00040100; /*Port B, Pin 2 is configured as Alternative 1 (GPIO)*/
    //GPIOB_PDDR = 0x00000004; /*Sets port B pin 2 to Input*/
    PORTD_GPCLR = 0x00000000; // Initialize Port D0

    GPIOD_PDDR = 0x00010100;

    SIM_SCGC5 |= SIM_SCGC5_PORTA_MASK; /*Enable Port B Clock Gate Control*/
    SIM_SCGC6 |= SIM_SCGC6_ADC0_MASK;
    ADC0_CFG1 = 0x0C;
    ADC0_SC1A = 0x1F;


//  SIM_SCGC6 |= 0x00;  /* Enable ADC1 Clock Gate Control */
//  ADC1_CFG2 = 0x0C;    /* Configure ADC1, refer to the documentation for the specific values */
//  ADC1_SC1B = 0x00;    /* Start conversion on ADC1 channel, refer to the documentation for the specific values */


    PORTA_PCR1 = 0xA0100;
```

CODE (K64F) CONTINUED...

```
    PORTA_PCR1 = 0xA0100;
    GPIOA_PDDR |= (0 << 1);

    /* Write your local variable definition here */
//b12 ->
    //1 2 4 8 16 32
    // 1100 0000 1100
    // 0000 0000 0100
    // 0100 0000 0000
    // 0000 0000 0100 0x0000004
    /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
    PE_low_level_init();
    /*** End of Processor Expert internal initialization.                    ***/
    /* Write your code here */
    uint32_t delay;
    uint8_t ret, who;
    int8_t temp;
    int16_t accX, accY, accZ;
    int16_t magX, magY, magZ;

    //int len;
    LDD_TDeviceData *SM1_DeviceData;
    SM1_DeviceData = SM1_Init(NULL);

    printf("Hello\n");

    int16_t data1;
    int16_t data2;

    int len;
    unsigned long i ,data, Percent, High, Low, realTemp, setTemp;
    unsigned long Hs = 600; unsigned long Ls = 66;
    int fanSpeed = 0;
    int countR = 0;
    int dataL = 0;
    int dataR = 0;
    int toggle = 1;



    FX1_Init();

    for(;;) {
/*
        data1 = ADC_read16b();
        data2 = data1 * 33 / 0xFFFF;
        printf("RAW Temperature value in decimal \t~ %4d\n",temp);
        len = sprintf(write, "RAW Temperature value in decimal \t~ %4d\n",temp);
        //GPIOB_PDOR = 0x4; // makes pin 2 HIGH
        SM1_SendBlock(SM1_DeviceData, &write, len);
        for(delay = 0; delay < 300000; delay++); //delay
        //GPIOB_PDOR = 0x0; // makes pin 2 LOW
*/


        // NEW CODEEE

        outputCompare(Hs, Ls, 91, toggle);
                data = ADC_read16b();
                int portBread = GPIOB_PDIR & 0xC0C;
                toggle = 1;
                if (portBread & 0x800) {Hs = 1000; Ls = 1; fanSpeed = 1; toggle = 0;} /* If pin 11 is powered */
                else if (portBread & 0x4) {Hs = 600; Ls = 66; fanSpeed = 2;} /* If pin 2 is powered */
                else if (portBread & 0x8) {Hs = 466; Ls = 200; fanSpeed = 3;} /* If pin 3 is powered */
                else if (portBread & 0x400) {Hs = 300; Ls = 366; fanSpeed = 4;} /* If pin 10 is powered */
                        else {
                        dataR = data * 33 / 0xFFFF;
                        Percent = dataR * 100 / 33;
                        Percent = (Percent * 70 / 100) + 49;
                        if (Percent > 99) {
                            Percent = 99;
                        }
                        fanSpeed = Percent;
                            High = Percent * 1000 / 100;
                            Low = 1000 - High;
                            Hs = High / 1.5;
                            Ls = Low/1.5;
                        }
```

CODE (K64F) CONTINUED...

```
                        Low = 1000 - High;
                        Hs = High / 1.5;
                        Ls = Low/1.5;
                    }

            outputCompare(Hs, Ls, 60, toggle);
            GPIOC_PDOR= 0x04;
            //outputCompare(Hs, Ls, 10);
            printf("Fan Speed \t: %4d\n", fanSpeed); //Prints the string
            //outputCompare(Hs, Ls, 10);
            len = sprintf(write, "Fan Speed \t: %4d\n",fanSpeed);  //Prints the string
            //outputCompare(Hs, Ls, 10);
            SM1_SendBlock(SM1_DeviceData, &write, len);//Sends the data through SM1
            //for(delay = 0; delay < 300000; delay++); //delay
            outputCompare(Hs, Ls, 500, toggle);


   }



   /* For example: for(;;) { } */

   /*** Don't write any code pass this line, or it will be deleted during code generation. ***/
  /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component. DON'T MODIFY THIS CODE!!! ***/
  #ifdef PEX_RTOS_START
    PEX_RTOS_START();                  /* Startup of the selected RTOS. Macro is defined by the RTOS component. */
  #endif
  /*** End of RTOS startup code.  ***/
  /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
  for(;;){}
  /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
} /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/

/* END main */
/*!
 ** @}
 */
/*
 ** ###################################################################
 **
 **     This file was created by Processor Expert 10.4 [05.11]
 **     for the Freescale Kinetis series of microcontrollers.
```

THE FUTURE

KEY FEATURES AND FUNCTIONALITY

APP-CONTROLLED FAN

Control the fan's speed from anywhere using a smartphone or table, offering a sleek interface for effortless control.

REMOTE ACCESS AND NOTIFICATIONS

Remote access to the fan, even when away from home. Users can turn the fan on or off, adjust settings, and receive notifications regarding room temperature, air quality, and filter replacement reminders.

PERSONALIZED SETTINGS

Allows users to create personalized fan settings tailored to their preferences, ensuring a comfortable environment at all times.

INTELLIGENT MODES

Features intelligent modes to enhance comfort and convenience. For example, a sleep mode that gradually decreases the speed and eventually turns off as the user falls asleep.

ENERGY EFFICIENCY AND MONITORING

Prioritizes energy efficiency by incorporating smart sensors, adjusting speed and airflow, to reduce unnecessary energy consumption for eco-consciousness.

SMART HOME INTEGRATION

Seamlessly integrates with popular smart home ecosystems. Users can \+ conveniently control the fan using voice commands or automate it alongside other smart devices in their homes.

A GUIDING PATH: IMPROVEMENT IMPLEMENTATION

ESP IC CHIP

Enables us to send data to a web server is a lot smaller and a lot more flexible that than K64F Microcontroller

BETTER TEMP SENSOR

Lets us calculate accurate temperature values consistently

MAKE OUR OWN PCB

To help mitigate the noise occurring in our system

WEB SERVER TO SEND DATA

Allows us to send data freely without worrying about wires that cause noisy values and random lag

THANK YOU

Does anyone have any questions?