TEMPERATURE FAN CONTROL

EE128 Final Project

PART 1. Project Description

PART 2. System Design

PART 3. Implementation Details

PART 4. Testing/Evaluation

PART 5. Discussions

PART 6. Roles and Responsibilities of Group Members

PART 7. Brief Conclusion

Ryan Andrew Bordadora
862226054

Karam Shanti
862229038

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

Introduction & Project Summary

BreezeControl is an innovative project aimed at developing a smart fan that can be conveniently controlled through a dedicated mobile application. This fan aims to create a comfortable and personalized airflow experience. The project will focus on designing and prototyping a smart fan for the eventual integration of the fan with various smart home ecosystems.

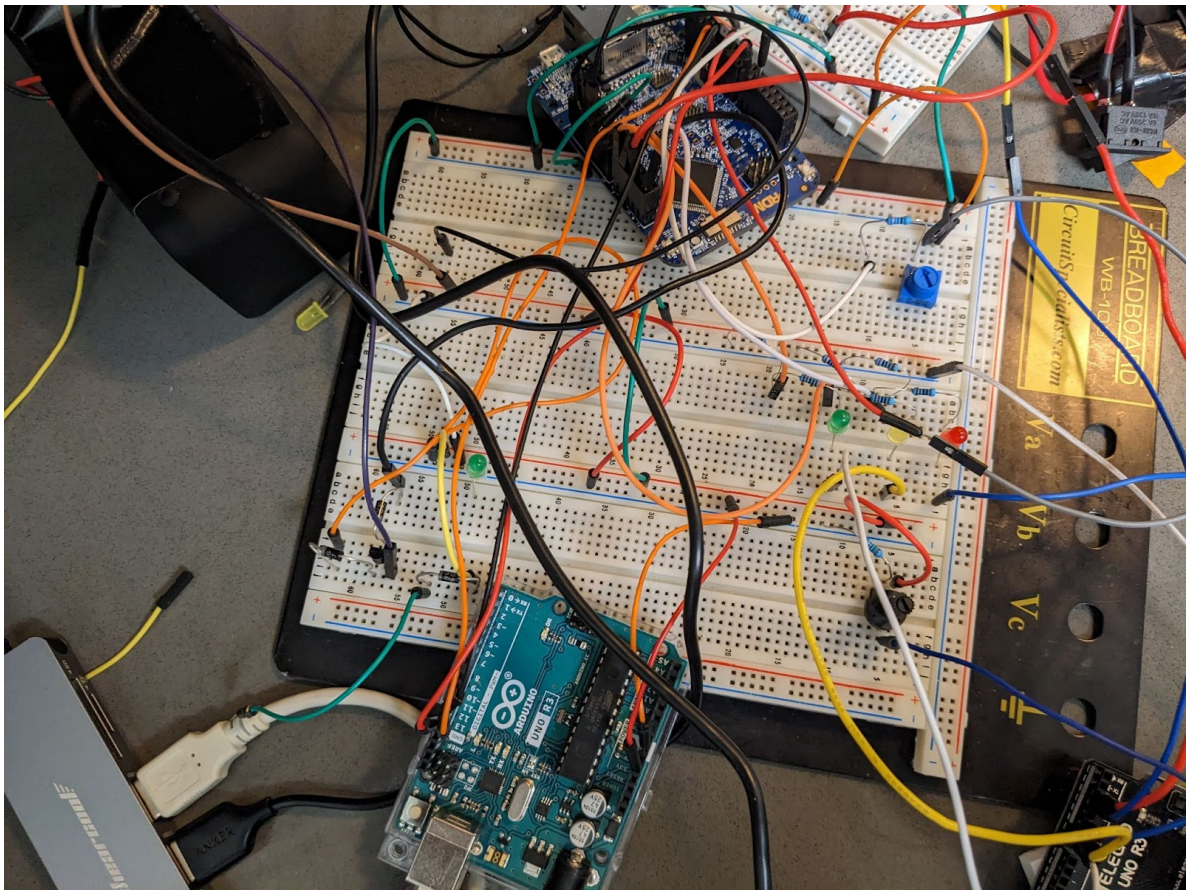
Complexities

- GPIO Pins
- ADC for controlling the set temperature
- PWM for a DC Motor
- Serial Communication for an LCD Display

Extra Parts

- DC Motor
- LM35 Temp Sensor
- NPN BJT
- Other Basic Circuit Materials

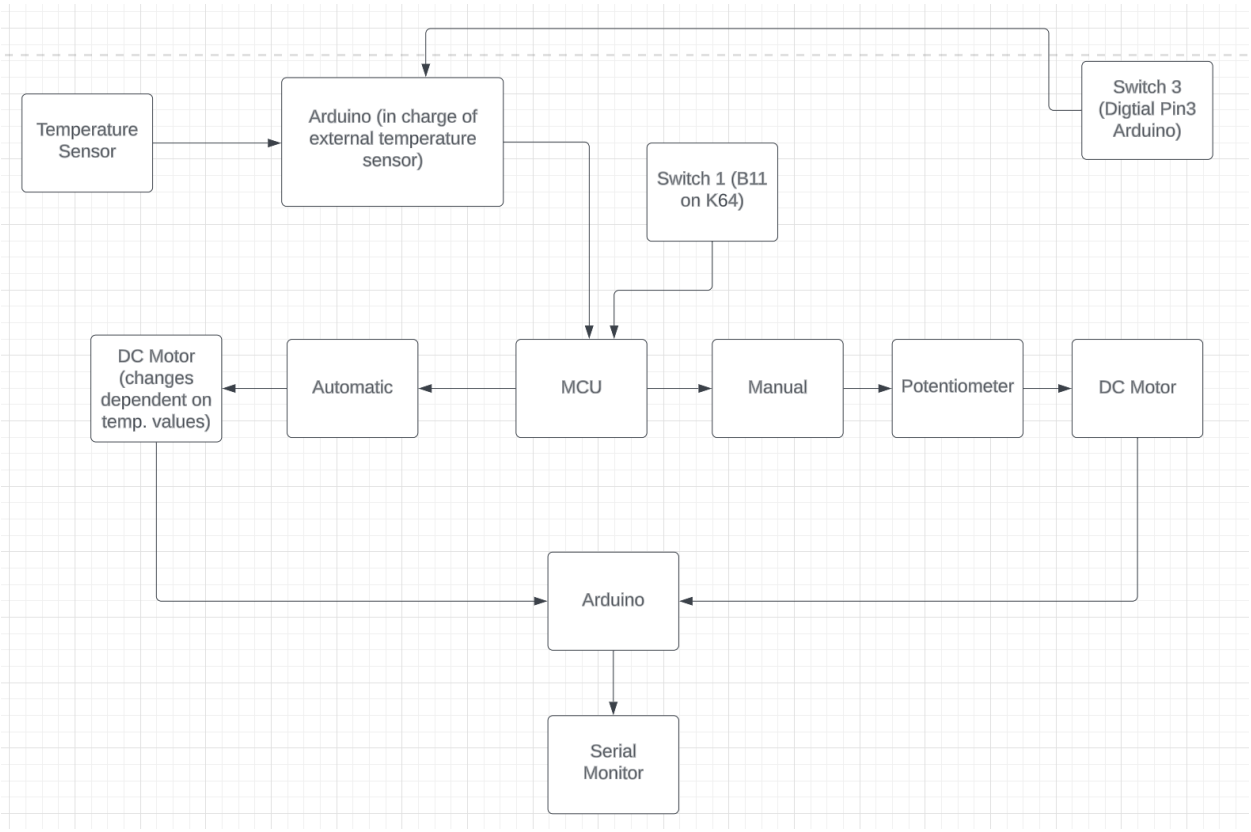
Part 2. System Design



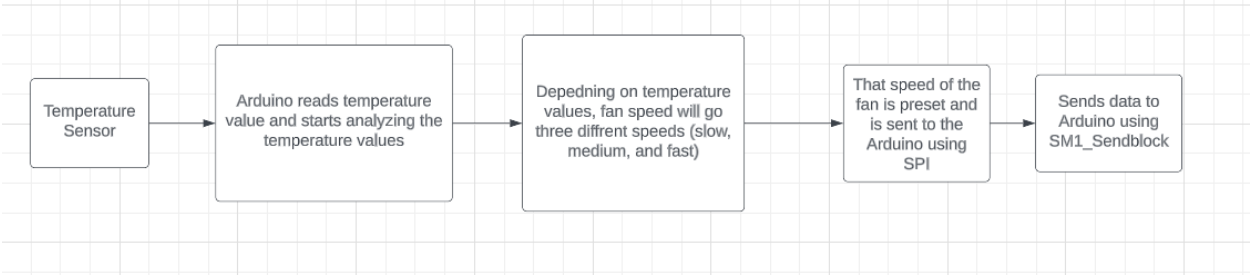
Current Implementation

- **Manual Personalization**
    - Allows users to personally set speed levels in both automatic and manual mode through potentiometers, catering to their specific comfort needs.
- **Automatic Climate**
    - Incorporates a 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 through Serial Monitors, allowing users to track and manage their fan's usage compared to the current temperature.

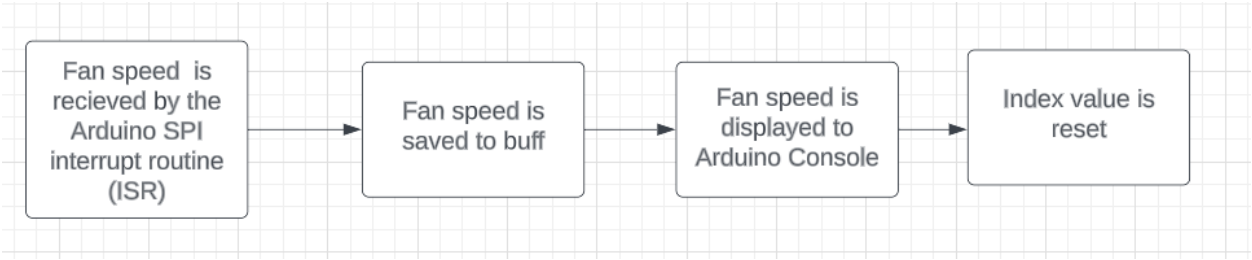
Block Diagrams and FlowCharts



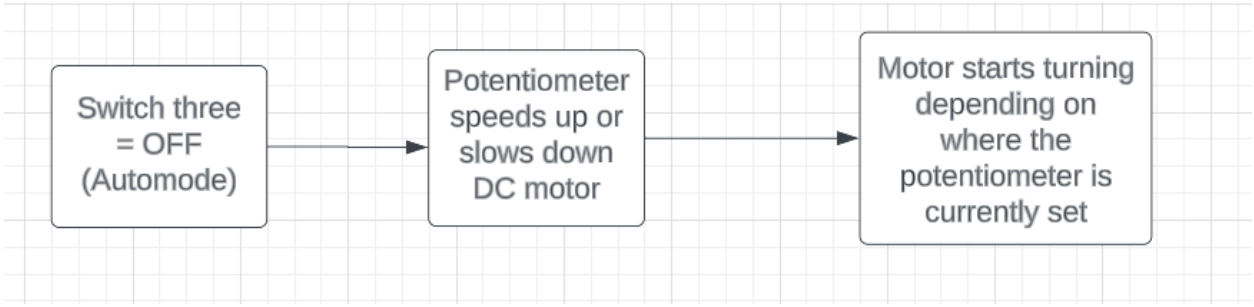
(High level description)



(Temperature -> MCU)



(MCU -> Arduino)



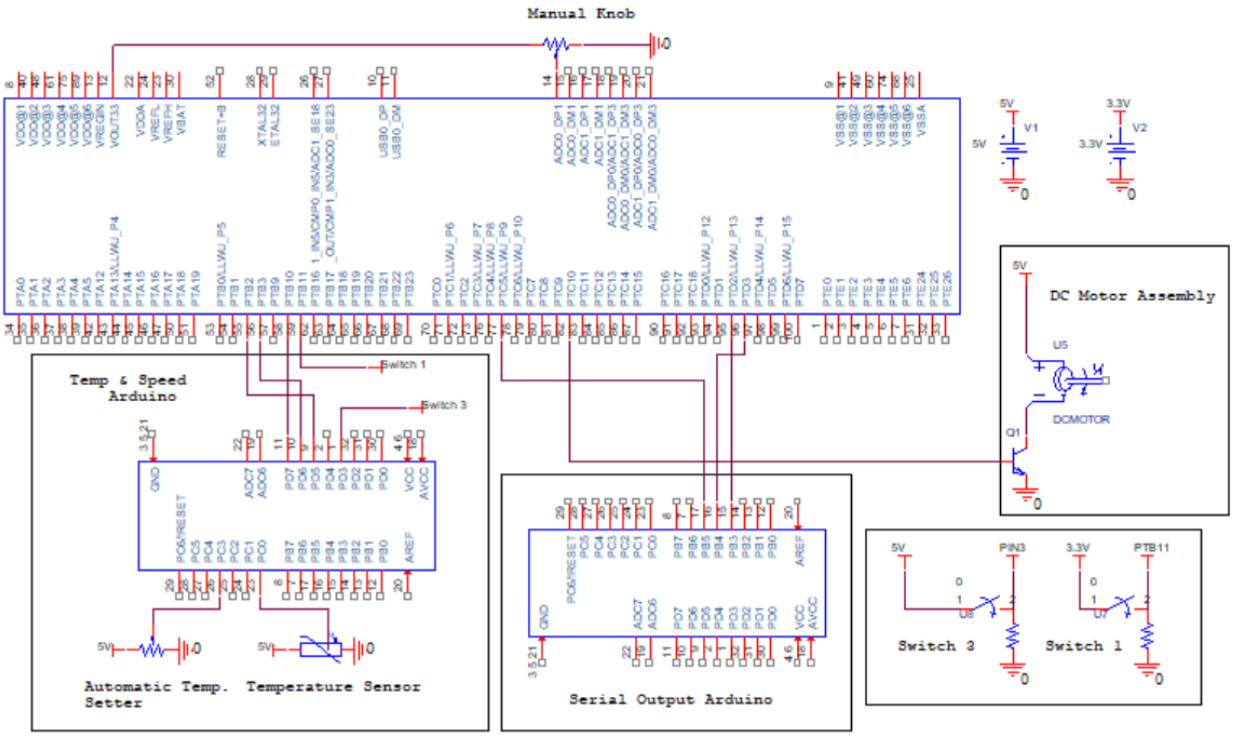
(DC Motor Manual Mode)

Explanation

Our system starts once the system has power and depends on which switches are on or off. If switch 3 is on then the Arduino connected to the temperature sensor sends a signal to the K64F to tell the DC motor how fast it should spin (slow, medium, or fast, depending on how high the temperature value is). This would be considered the system's automatic mode. To disable automatic mode you could turn off switch 3 and that enables you to use the potentiometer to change the speed of the DC motor and switch 1 will turn off the whole system. The serial monitor will display the current ADC values of the DC motor in either setting (automatic or manual).

Part 3. Implementation Details

Schematics



Code

Arduino (Temp Sensor)

```
int b_buf = 0x00;
void setup()
{
    pinMode(7, OUTPUT);
    pinMode(6, OUTPUT);
    pinMode(5, OUTPUT);
    pinMode(3, INPUT);
    Serial.begin(9600);//Set Baud Rate to 9600 bps
}
 void loop()
{ if (digitalRead(3)) { //Checks if Automatic is enabled
    unsigned int val;
    unsigned int dat;
    val=analogRead(A0);//LM35 (Sensor) on Analog 0
    dat=(500 * val) /1023; // Convert to Celsius
    Serial.print("Temp:"); //Display the temperature on Serial monitor
    Serial.print(dat);
    Serial.println("C");
    b_buf = analogRead(A3); // Read and Maps Potentiometer for User's Set Temp
    b_buf = map(b_buf, 0, 1023, 10, 40);
    Serial.print("SetTemp:"); //Display the Set temperature on Serial monitor
    Serial.print(b_buf);
    Serial.println("C");
    // Compares the User's Set Temp to the Actual Temp
    //Decides what speed to set based on the difference in set vs actual
    // Set Speed of K64 by outputting to a different pin
    if (dat > b_buf + 10) {digitalWrite(7, HIGH); digitalWrite(6, LOW); digitalWrite(5, LOW);}
    else if (dat > b_buf + 6) {digitalWrite(6, HIGH); digitalWrite(7, LOW); digitalWrite(5, LOW);}
    else if (dat > b_buf + 3) {digitalWrite(5, HIGH); digitalWrite(6, LOW); digitalWrite(7, LOW);}
    else {digitalWrite(7, LOW); digitalWrite(6, LOW); digitalWrite(5, LOW);}
    delay(500);
    }
}
```

Arduino (Serial Monitor)

```
#include <SPI.h>  // Not needed if device is I2C
#include <Wire.h>

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 senso
int counter = 0;      // Seconds counter
unsigned long int currentTime,nextTime;


void setup() {


  Serial.begin(9600);
// NEWWWW CODEEEEEEEEEE
  pinMode(MISO, OUTPUT); // Master Input - Salve Output
  SPCR |= _BV(SPE); //SPI Slave Mode On
  indx = 0; // buffer empty
  process = false;
  SPI.attachInterrupt(); // Interrupt Function
  pinMode(A0, INPUT);
}


// NEWWWW CODEEEEEEEEEE
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() {

  if (process) { // If process is valid
    process = false;
    Serial.println (buff); // print everything in the buff to the serial monitor
    delay(10); // added a little delay to be able to actually read the serial monitor
    indx= 0; // resets index to zero
  }
}
```

K64F Code

```
**     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];
```

```c
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;
}


// 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
}
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
    }
}
```

```c
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*/
```

```c
    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;
    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(;;) {

    // 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; // Converts data value
                    Percent = dataR * 100 / 33;
                    Percent = (Percent * 70 / 100) + 49;
                    if (Percent > 99) { // Ensures percent value doesn't overload
                        Percent = 99;
                    }
                    fanSpeed = Percent;
                        High = Percent * 1000 / 100;
                        Low = 1000 - High;
                        Hs = High / 1.5;
                        Ls = Low/1.5;
                    }

            outputCompare(Hs, Ls, 60, toggle); // Calls outputCompare function to start motor
            GPIOC_PDOR= 0x04; // Turns on pin 2
            //outputCompare(Hs, Ls, 10);
            printf("Fan Speed \t: %4d\n", fanSpeed); //Prints the string
```

```
    /* 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.
 **
 ** ###################################################################
 */
```

Part 4. Testing/Evaluation

Parts Required For Testing

- Heat Source
- Cooling Source

Introduction:
This section will discuss the testing environment and the calibration methodology used for a temperature-controlled fan. The temperature-controlled fan aimed to maintain a stable temperature by adjusting its speed based on the surrounding environmental conditions. The testing environment simulated a kitchen setting, utilizing various appliances to alter the temperature around the fan's sensor.

Testing Environment

The testing environment aimed to mimic real-world scenarios, specifically focusing on kitchen conditions where temperature fluctuations are common due to cooking activities and thus easy to reproduce. The following appliances were utilized in the kitchen setting:

Stove: The stove were used to generate heat within the kitchen environment. By varying the cooking temperature and duration, the temperature around the sensor could be controlled to observe the fan's response.

Refrigerator: The refrigerator played a crucial role in altering the ambient temperature. By opening and closing the refrigerator and freezer door, the release or retention of cool air affected the overall temperature within the kitchen space.

Calibration Methodology:
To ensure the accurate and reliable operation of the temperature-controlled fan, a calibration process was conducted. The calibration methodology involved the following steps:

Sensor Placement: The temperature sensor was strategically positioned within the testing environment to capture the representative temperature of the surroundings.

Data Collection: The temperature sensor was connected to a data acquisition system (the Arduino), which recorded temperature readings at regular intervals. The data acquisition system was configured to log temperature values accurately and consistently. The separation and addition of the extra board were necessary to ensure accuracy and avoid noise.

Baseline Measurements: Before initiating any experiments, baseline measurements were recorded with no appliances running in the kitchen. These readings served as a reference point for subsequent experiments.

Appliance Activation: Each appliance in the kitchen was individually activated to simulate their real-world usage. The temperature readings were continuously monitored during the activation of each machine to determine the impact on the surrounding temperature.

Calibration Adjustment: Based on the results, calibration adjustments were made to the temperature-controlled fan system. These adjustments ensured that the fan's speed modulation accurately corresponded to changes in the surrounding temperature, providing efficient temperature regulation, as well as ensuring the fan never turned off when it was not supposed to.

Testing Scenarios

Testing Environment and Calibration Methodology:
Please refer to the previous section for detailed information on the testing environment and calibration methodology employed for the temperature-controlled fan.

Testing Different Speeds by Changing the Requested Set Temperature:
Objective: To assess the temperature-controlled fan's response and speed adjustments based on changes in the requested set temperature.

Procedure:

1. Set the initial requested set temperature to a comfortable level.
2. Observe the fan's initial speed and airflow.
3. Increase the requested set temperature and note the fan's response in terms of speed adjustment and airflow.
4. Decrease the requested set temperature and observe the fan's speed adjustment and airflow.
5. Repeat steps 3 and 4 for multiple temperature increments or decrements.
6. Observe the fan's initial speed and airflow.

Testing the System's Response to an Increase of Heat through the Stove:
Objective: To evaluate the temperature-controlled fan's ability to respond to a sudden increase in heat generated by the stove.

Procedure:

1. Ensure the temperature-controlled fan is operating at a stable speed and maintaining a desired temperature range.
2. Activate the stove and increase the heat output to a predefined level.
3. Observe the fan's response in terms of speed adjustment and airflow.

Manually Using the Knob to Change the Fan Speed:
Objective: To test the manual control capability of the temperature-controlled fan's speed adjustment.

Procedure:
1. Set the temperature-controlled fan to manual mode, where the speed is regulated based on the surrounding temperature.
2. Manually adjust the fan speed using the control knob or interface.
3. Increase the fan speed to maximum and decrease it back to the lowest setting
4. Observe the fan's response to the manual speed adjustment and changes in airflow.

Conclusion:
The three test cases, including testing different speeds by changing the requested set temperature, testing the system's response to an increase of heat through the stove, and manually using the knob to change the fan speed, provide valuable insights into the performance and functionality of the temperature-controlled fan. These tests evaluate the fan's responsiveness, adaptability, and ability to maintain a stable temperature in varying conditions, ensuring its effectiveness in real-world applications, such as kitchens.

Part 5. Discussion

Limitations & Advances Introduction

This section will discuss the limitations we had in our project and what we could do about those limitations moving forward with this project.

The Board

The K64F microcontroller itself is adequate for this job, but it falls short in comparison to other microcontrollers available. For example, the Raspberry Pi has the capability to host data on a web server, while the ESP32 boasts a built-in wifi module that enables web access, data transmission, and Bluetooth usage. These alternative microcontrollers offer faster response times, efficiency in data communication, and a lack of excessive interference. Additionally, they eliminate the need for extensive pin configuration prior to usage, making them far more user-friendly and thus simplifying the overall process of working with a microcontroller. The K64F microcontroller, although useful and efficient in many scenarios, proves to have too many complexities that serve little use in a project like this. Therefore, when it comes to implementing a product of this nature, there are better-suited options available that can fulfill the requirements more seamlessly.

Sending and Receiving Data

When trying to send ADC values to the Arduino using the K64F, we would sometimes get random symbols and other times the system would just stop. After editing and troubleshooting, we were finally able to read some actual values but the errors would still sometimes happen. After some testing, we realized that it was likely caused by noise, which could be caused by a variety of reasons.

Power Issues

A frustrating aspect of the K64F is the fact that it could only receive and deliver data at 3.3V. In contrast, most devices use a standard 5 volts, from a normal sensor, to an Arduino, and LCDs. Much of this system had to be designed around this inconvenience, having to make sure that we were not putting too much voltage back into the K64F. This proved to be quite tedious since our project involved a temperature sensor connected to an Arduino that transmitted temperature data to the K64F for controlling a motor based on the current temperature, most of which ran on a 5V input.

Sensors

Another limitation was that the temperature sensor on the K64F was not even working properly. We put the K64F over the stove to try to raise its temperature and it barely went up a degree. Since we were trying to make a “smart” fan, a fan that couldn’t detect temperature is kind of useless. This is why we switched to an Arduino Temperature Sensor.

Challenges & Drawbacks Introduction

This section will discuss the challenges and drawbacks we had in our project

Karam

Challenges

There were multiple challenges that were faced when figuring out how to display the ADC values from the DC motor and the temperature values from the K64F to the LCD display. The first actual problem when we initially wanted to use the LCD was that we couldn't get both the ADC value and the temperature values to display on the screen simultaneously despite explicitly instructing them to. Occasionally, these values would even transform into random symbols. After swapping LCD displays and changing the code, we were able to get it working consistently.

However, when we were integrating our components, a new problem emerged. My responsibility focused on the Serial connection, ensuring proper data reading and transmission. Meanwhile, another team member concentrated on the core system that was controlling the motor. Unfortunately, during the combination of these two aspects, the motor spun correctly but the LCD refused to print anything. Additionally, we discovered that the serial monitor was not functioning properly. This setback prompted us to backtrack and troubleshoot the serial monitor. After testing, noise proved to be a problem once again. Ultimately, the LCD screen idea was ditched and efforts were made to isolate the Serial connection to ensure it was obtaining and outputting the proper data.

Drawbacks

A large drawback I experienced while working on this project was the K64F microcontroller. It takes a while to program anything on it, it's really bulky and for this project, it lacked a purpose aside from it being a requirement in our project. It's just requires a lot of extra steps we had to take without a benefit (or a benefit we can see). Moreover, if we were to sell this project, the K64F Microcontroller costs about $50 alone, making it far beyond the budget and scope of this project. It would be far better to just find a cheap IC Chip that could do everything the K64F can do for about $10 and make a PCB with that chip for the rest of the Smart Fan and sell that.

Ryan

Challenges

The central challenge of this project mostly revolves around noise and voltage levels. Before noise is explained, the voltage requirements played a significant role in the project's direction. If you noticed the components used in this project, most of the components operated at a rating of 5V. This proved to be a consistent problem throughout the assembly process. The motor assembly ended up needing a transistor, which was used to create a low-side switch with an npn BJT, which proved to be problematic. The sensor, which had to be used since the plan with the K64F fell through, also had to be used with 5V. This all culminated in a problem of noise.

For the sensor, we had planned originally to use the second ADC on the K64F to directly input the speed, but that proved to be a waste of time. After stepping down the voltage, the readings proved to

be too small for the ADC to accurately capture. Moreover, the noise being generated made the readings totally unreliable. After numerous iterations, the decision was made to isolate the sensor entirely and provide it with its own Arduino and code. This approach greatly improved its performance and reliability as well as conveniently giving us a method of interacting with the GPIO pins. However, this is not the best method and is convoluted at best. The BJT also proved difficult to deal with. It provided no protection a MOSFET or an SSR would provide, but it was the best on hand. Due to this limitation, the motor feedback to the K64 was not mitigated. Consequently, while the K64F was safeguarded against overvoltage, it was also exposed to random noise, significantly impacting the Serial connection and its readings.

Improvements and the Future

- Implement an app or web server to display data
- Use a cheaper more flexible microcontroller like the ESP8266 to replace the K64F and both Arduinos
- Use a external temperature sensor that is able to read temperature data properly and accurately without noise interference for easier integration
- Design a PCB (Printed Circuit Board) and isolate specific sections of the board from interacting with one another to minimize noise
- Gain a good external power supply and a large, more powerful motor (Could be AC instead)
- Better switching mechanism: External power supply would allow for a motor driver. Also could use a MOSFET and SSR to decrease noise and increase consistency and speed.

The Final Product Aims to meet these goals:

- **Personalized Settings: Replace the ADC potentiometers**
    - Create personalized fan settings tailored to their preferences, ensuring a comfortable environment at all times.
    - Control the fan's speed from anywhere using a smartphone or table, offering a sleek interface for effortless control.
- Energy Monitoring: Improved Sensors
    - 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 automate it alongside other smart devices in their homes.
- Remote Access: Replace Serial Monitors
    - 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.
- Intelligent Modes
    - Features intelligent modes to enhance comfort and convenience, essentially allowing for programmability. 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.

Part 6. Group Responsibility

Karam

[What did you do on the project? What did you implement?]

1. Assembly
    a. Code for the ADC and Serial for the K64 and Arduino
    b. Wired the ADC and Serial
    c. Wired the two switches
2. Lab Report
    a. Wrote System Design's Explanation and Conclusion
    b. Drafted Discussion
    c. Created All the Box diagrams and Flow Charts

Ryan

[What did you do on the project? What did you implement?]

3. Assembly
    a. Code for the PWM/Timer for the Motor
    b. Code for the Arduino Temp Sensor
    c. Wired the Low-Side Switch for the Motor
    d. Wired the connection between the Temperature Arduino and the K64F (GPIO Input)
4. Lab Report
    a. Wrote the introduction, Implementation, and Testing
    b. Revised and Rewrote the Discussion
    c. Created the Schematic
    d. Created and Designed the Presentation
    e. Video Taped and Edited the Video

Conclusion

[Summary of what happened. Reinstate the objective of this project. Significance of the project. Reflection of methodology. Lessons learned. Recommendations for future research. Final concluding statement.]

This project took us on a journey filled with twists and turns, much like navigating through a complex maze. Initially, we thought that this project would be pretty simple and straightforward. We were just going to use the temperature on the K64F, write some code to change the speed on the DC Motor that depends on the temperature value and output that value along with the ADC value of a LCD screen that's connected to an Arduino. However, things started going wrong once we discovered that the K64F temperature sensor was not working. So we had to rethink our approach. After some time we decided on the external temperature sensor that sort of worked. I say this because the temperature sensor was very sensitive and had to be placed in the freezer for some time and then heated up a little bit afterwards to make it have a neutral temperature and only then could we properly use it. We also had other issues such as the motor causing so much noise that the serial monitor couldn't even read the ADC values coming from the DC motor. But after some time and a lot of testing trials, we were able to successfully debug and demo our system.

Our project's main goal was to be able to build a prototype smart fan that was able to automatically detect the current temperature of a room and run a DC motor in three settings (slow, medium, fast) depending on how warm the room was, which allowed the people coming into the room to have a nice cool room to walk into, and eventually we could also allow users to be able to integrate it with other various smart home ecosystems. Our project's significance has to do with creating a new way to live with technology. A device that enables you to personalize a smart fan that fits your needs, a way to help you monitor how much energy your fan is using up, and a way to regulate the current temperature of a room and keep it at a certain temperature.
In terms of our methodology, our system started off with just looking at what devices we had at our disposal and what the current temperature sensors on the market looked like, what they were able to do, and how they were able to do it.

After doing some research on that, we came up with our initial design which consisted of a K64F connected to a DC motor and SPI connected to an Arduino which was connected to a LCD screen. However, we ran into a big issue. Our temperature sensor on the K64F was not working properly. We tried figuring out ways to fix this issue using the K64F data sheet and the NXP discussion form but we couldn't figure out how to fix the issue. So we had to rethink and figure out where we were going to go from here. We thought of a couple of ideas we could do and eventually decided on one, using an external temperature sensor. Despite this, this plan came a whole other complexity we had to incorporate. Because we were going to be using an external temperature sensor, the only real way we would be able to read and send the signal of the temperature sensor without an overload of noise was through another Arduino. So at this moment our current setup consisted of 2x Arduinos, 1x external temperature sensor, 1x LCD,

and 1x DC motor. So we tried the temperature sensor out and we were able to get proper readings (after calibration) but the new issue we encountered was with our serial monitor. It was unable to read actual ADC values coming from the motor. It was just giving us random symbols or sometimes even nothing so we did some research on some of the fixes for this issue and started to play around with the baud rate, change the code a bit, and step down the voltage from the DC motor which eventually got us the values we wanted.

Now, during our demoing session, we felt that we needed to find a way to test everything in our system quickly and efficiently to allow the viewers to see exactly what we have done and so we can make sure everything is working. So we decided to use switches to show when our system was in Automatic mode or manual mode, we also had LEDs set up to show how hot or cold the current temperature values were (only in Automatic mode) and we also added a potentiometer to be able to manually change the speed of the DC motor. And after calibrating the temperature sensor about a dozen times, recording everything properly, and testing every feature out making sure we do all this in one take, we were finally able to get a good video of our project working perfectly.

Some lessons we learned from this project are: always test what you're going to use before using it (temperature sensor on K64F), working together rather than working on individual parts separately and trying to put them together later (Serial Monitor + Motor problem), as well as starting a bit early. Realistically this project would have probably been done in about a week and a half but debugging everything and dealing with unexpected errors in our devices really pushed us back. Also talking with your TA is a big help. In terms of future research, we will probably be looking more into more alternatives for the Arduino and the K64F specifically. As well as finding something with a Wifi-module that would enable us to send data to the web or put it in an app. We would also want to use components that are cheap and reliable in the chance that we do want to make this project into a sellable product. Some things we would also have to look into are IC chips that would fit our needs, figuring out how to make a PCB that successfully isolates noise from different parts of the system as well as a way to make our project look more aesthetically pleasing.

In terms of our concluding statements, we would say that this project was annoying, stressful, and was probably one of the most annoying things to debug, but even with dealing with all that it was still a fun project to work on. It was also one of the few times we had a project we had to do so this was a great experience for us, and maybe one day we will get a chance to work on this project again but this time make it into an actual product rather than just a class assignment.