Smart Fan Write-Up

Introduction:

Since it is the end of the semester, we have been tasked with creating a real-world mechanism simulation as a final project. There are so many different mechanisms to choose from, but this choice was not easy. The project is to be simulated on a small microcontroller called a Dragon 12 Light kit, which uses assembly and C code to program. These programming languages have their own flaws and errors, which made development tricky, but after hours of engineering, I have come up with a smart fan design. What inspired me to develop this mechanism was a fan I have at home with variable speed, time setting, and directional adjustment. If my fan was on a larger scale, it could be compared to a modern fan with smart settings.

Features:

The main features the smart fan contains is variable speed, timer based shut off, temperature-based activation and shut off, and directional adjustment. It can compare to a modern fan since the blade speed can be changed, there is a timer for how long the fan should run, and the direction can be swiveled. Nearly all of the larger non-stationary fans have these general features now.

Components:

There were many internal and external components utilized within the development of my smart fan. The components and their types shall be listed below.

Internal:

* TIMER module – Time based operations
* DIP switches 0, 1, and 7 – [0]: Stays up for H-bridge, [1]: Stays down for H-bridge, [7]: Toggle fan swivel
* Switch 5 – Turns the fan on and off
* LEDs – Motor control
* AD0 – Channels 4 (Phototransistor) , 5 (Temperature)
* AD1 – Channel 3 (External potentiometer input)
* LCD – Display text
* Motor0 – Motor control
* Servo54 – Servo control
* RTI – Switch inspection loop
* Keypad – Number based user input
* SCI0 – Display and receive characters from the connected computer
* PORT T Bits – 0 (External switch input), 1 (Night light output), 2 (Speaker output), 3(External switch input)

External:

* Servo – Simulate fan swivel
* Motor – Simulate fan blade motor
* 2 Buttons – Set / clear timer, set / clear temperature trigger
* Speaker – Make a noise on fan toggle
* Potentiometer – Set fan speed level
* LED – Night light
* 2 F to M wires – Motor connection
* 14 M to M wires – Other connections
* 2 1k Resistors – LED, Speaker
* 2 10k Resistors – Switches

Purchase Incentive:

There are numerous types of fans out in the market currently, but my model is at the top of the list. As stated previously, the speed of the blade can be tweaked manually. What makes this fan stand out is the precision of the speed. Most fans just give the option to press a button to toggle the speed from around 4 different presets. With the smart fan, the user may dial the potentiometer to any speed they prefer, from 0 to a max speed of 255, giving the most precision for that desired chill. Additionally, the timer is particularly advanced where the user can insert a time up to 358839 seconds, or 98:99:99 on the keypad to keep the fan on for. Similar to the speed, most fans have a few preset times to keep the fan on for generally up to 6 hours. For workers, 6 hours may not be enough; the operator will have to continuously have to turn their fan on again, or they risk forgetting to turn the fan off when they leave their residence. A custom time to fit nearly all individuals may be set on the smart fan - from seconds to minutes to hours. The final standout that the smart fan contains would be the temperature trigger point. In situations where an individual cannot afford AC or would just like the fan to be on when a certain temperature is reached, can be simply set using the keypad. If the desired temperature point is hit (up to 200 degree’s F), the fan will automatically turn on, and turn off again once the room temperature is 3 degrees below the set point.

Development Issues:

Throughout the development of my project, I had many issues in which I had to overcome. These issues were not only external, but there were some internal coding issues I had as well, with C programming. To start off, I wanted to use an external distance sensor which could stop the fan if an object was so close to the blade. It would have been a positive addition, but it just would not work with my board for some reason. I do not believe it is the sensor itself, it’s a problem possibly with the board since I was able to use the sensor with my Raspberry Pi. This was somewhat frustrating, but luckily it was one of the first components I tried. Another issue I had was with my motor. At first, I could not understand why the motor was not running, even though it was connecting and spinning slightly when I shut off the board. I later figured out that the LEDS must be constantly turned on to match the first dip switch if I wished for rotation alongside the motor0 method. Secondly, my board would randomly decide it was not getting enough power to run, so it would continuously reset. What was strange about this was that it would have enough power sometimes when it was ran, and not enough on others. My fix to this was connecting the board to a larger power block. Likewise, in the TIMER loop originally, I gathered the state of ad0, and wrote to the LCD. This caused the loop speed to decrease considerably, down to about 1000 clicks per MS, instead of 24000. Since we are using a timer and other time-reliant devices, I had to switch those methods to the main loop of the program, which fixed this issue. Lastly, gathering and writing data from / to an array in C code is not the easiest to do repeatedly. In one of my character array, if I tried to write characters, and overwrite those same characters, it would just increase the size of the array, which is numerously different from over-writing the previous array. To fix this issue, I allocated space to overwrite the previous array, and this fixed the issue. Luckily, all of my major issues were able to be overcome, and my project is working just as expected.

Impossible Improvements:

Although my project is working as expected, there are still some issues I wish that could be improved, but to my knowledge cannot. One of the main preferable improvements would be the separate thread execution. Although the threads should be completely separated, certain actions seem to interfere with other threads, bogging them down at times. For example, when an external switch is pressed, the fan stutters slightly. This doesn’t really affect the performance of my project, much, but would be nice if it could be fixed. Another issue is the LCD, at some points, strange characters are displayed to the LCD. I’m unsure of why this is, especially with pre-written strings and such, but I could not really find a fix to this. It was very uncommon, but again, I spent lots of time on the project so it would be nice if it ran without error. Those really are the main improvements I can think of, and really isn’t too bad.

Future Improvements:

Future improvements could include a range-based sensor which turns the blade off if an object is too close to prevent injury, and a more compressed overall fan model where the wires aren’t visible, and the buttons are well-oriented.

Conclusion:

After working on my smart fan, I feel much more confident in my understanding of C and assembly code. Throughout my work, I had to understand the correlation between them both to determine a positive outcome. C coding now comes to me as second nature (with slight hesitance towards array). Initially, I struggled with reading external inputs, but now I can proudly read these with ease through ports and my code. My final project proves the knowledge I have learned in this course because it contains numerous units which we have learned, all combined into one C code file and some physical wire connections as well. I am very glad this course was provided with a final project because I tend to be a more hands-on person. Tests and quizzes aren’t as straight forward and interrupt my learning. I am excited to use this knowledge learned in the future to benefit myself and others.