Good Morning Blinds

Final Report - December 5th 2020

Prepared for:
ECE 5731
Fall 2020
Dr. Iyad Faisal Ghazi Mansour

Prepared by:
Lisa Branchick
Aaron Garofalo
Brian Neumeyer

Project Description:

Morning Blinds: Intelligent Window Blind Control. The Good Morning Blinds are “smart blinds” that adjust the light of the living space via a servo motor angle based on various inputs to a microcontroller. The project scope was to demonstrate an understanding of embedded controls and concepts using the PIC32 microcontroller, actuators, sensors, and I/O devices. Using I/O the PIC32 determines the servo position needed to adjust window blinds and convey information to the user via LCD. This can be accomplished either through manual or automatic controls, depending on a user input. The concepts covered in this project include bluetooth, timing, interrupts, analog-to-digital conversion, UART, and pulse-width modulation, among others learned throughout the course.

Program Stateflow:

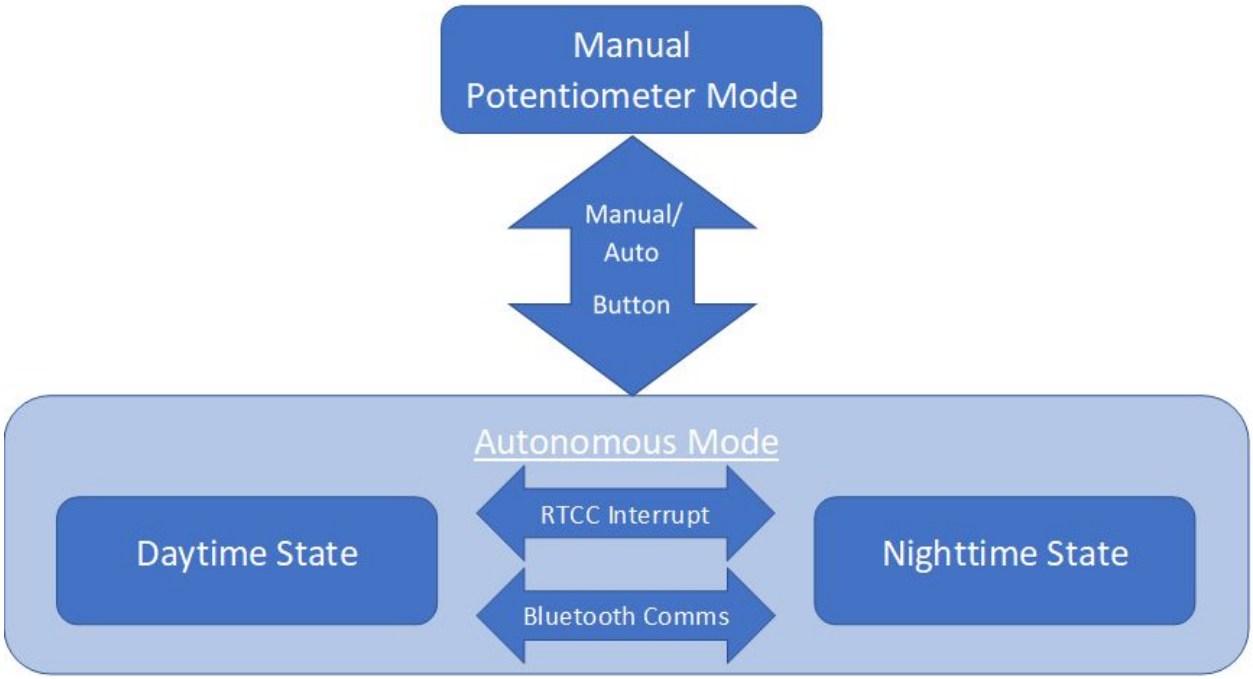


Figure 2: Stateflow diagram

The stateflow for the Good Morning Blinds is fairly simple. By default, the system runs autonomously switching between day and night mode with input from an interrupt timer or bluetooth device. The day and night modes will adjust the servo position to change the angle of the blinds accordingly. There is also a button that allows the user to switch between autonomous mode and manual mode. In manual mode the user can choose a custom angle for the blinds to be set, rather than full open or full close. In manual mode, the angle is controlled through the use of a potentiometer.

Materials Used:

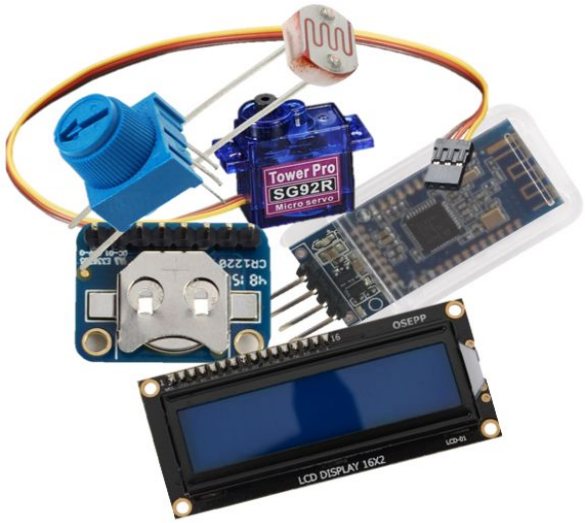


Figure 3: Components similar to those used in the project

Button: controls switching between automatic and manual operation modes. In automatic mode, the servo can either open or close the blinds, but it does not allow for setting between fully-open and fully-closed. In manual mode, the user can turn a potentiometer which is then mapped to the servo to allow for fine control of the blinds' angle. The state of the manual/auto mode is also shown on the LCD screen, so the user can easily understand the expected functionality of the system in the current state.

Potentiometer (Manual): if the user selects the button for manual mode, the user may use the potentiometer to finely adjust the angle of the window blinds as opposed to the automatic mode. This was achieved by using the ADC ports (B ports) and timer3 for ADC sampling. The input data from the conversion is then mapped to the servo angle output. The angle of the blinds is also read and output to the LCD screen.

Potentiometer (Battery Simulation): a second potentiometer was implemented to simulate the battery level of the battery sub-system if rechargeable batteries were included as the system's power supply. The information for the battery level is given via the LCD screen. As above, the use of the potentiometer required the use of ADC (B port) and timer3 for sampling.

LCD 1602A: the purpose of the LCD screen is to convey system information to the user. This information includes the ambient light level, battery level, the potentiometer to servo angle, and mode with respect to time (day or night). The LCD uses D ports for enable and read/write and E ports to manipulate display. The LCD has dimensions of 16x2 (columns x rows).

Timer45 was used to change the contents of the display every 5 seconds. Through multiple rounds of user testing, this was determined to be the most comfortable rate of change for a positive user experience.

Photoresistor: the photoresistor is a simple variable resistor that changes the voltage in response to ambient lighting conditions. The photoresistor uses an ADC port (B port) to provide input to the PIC32 microcontroller. As in the above components, timer3 was used for ADC sampling. Once the conversion is finished on the PIC32, the values (0-1023) are then converted using a simple equation that results in a percentage (lightPerc) to be displayed on the LCD. Further development would include PID control and a user-set desired light level to be maintained by the system.

RTCC Module: the real-time clock and calendar was to be implemented so that the servo could adjust the blinds based on the time of day. Initially, the on-board internal RTCC was to be used. This would require an oscillator crystal to run the clock. Another option was to use an external module that requires I2C ports to be used. However, due to multiple challenges, it was ultimately decided to implement a timer to set the day/night modes.

Timer 1 from the PIC32 was used to monitor and control the day/night state of the system. Since Timer 1 is a 16-bit timer, the maximum duration before rolling over is approximately 200 ms. Because timers 2-5 are already being used for other functions, and since 200 ms was not enough time, a loop was created to delay the functionality of the day/night state conversion. Rather than delay time while inside the interrupt (which is not good programming practice), the interrupt would instead increase an arbitrary count the first 50 times the interrupt is called. On the 51st interrupt, the day/night flag is negated (and the count reset to 0), thereby changing the state of the system every 10 seconds.

Bluetooth Module: the DSD TECH HM-10 Bluetooth 4.0 BLE iBeacon UART Module was selected to allow the user to control the Good Morning Blinds from a distance using their Android smartphone. The bluetooth module has a developer-friendly setup that allows for easy application development. The bluetooth module uses port D to communicate with the PIC32 microcontroller.

Servo Motor: The SG90 servo is used to control the angle of the blinds through a PWM signal. To control the PWM signal, timer 3 was selected. In autonomous mode, the servo is set to either 0° or 90°. However, during manual mode, a user is able to finely control the angle of the blinds with a potentiometer. In this mode, the blinds can be placed at any angle between 0-90°.

System Block Diagram:

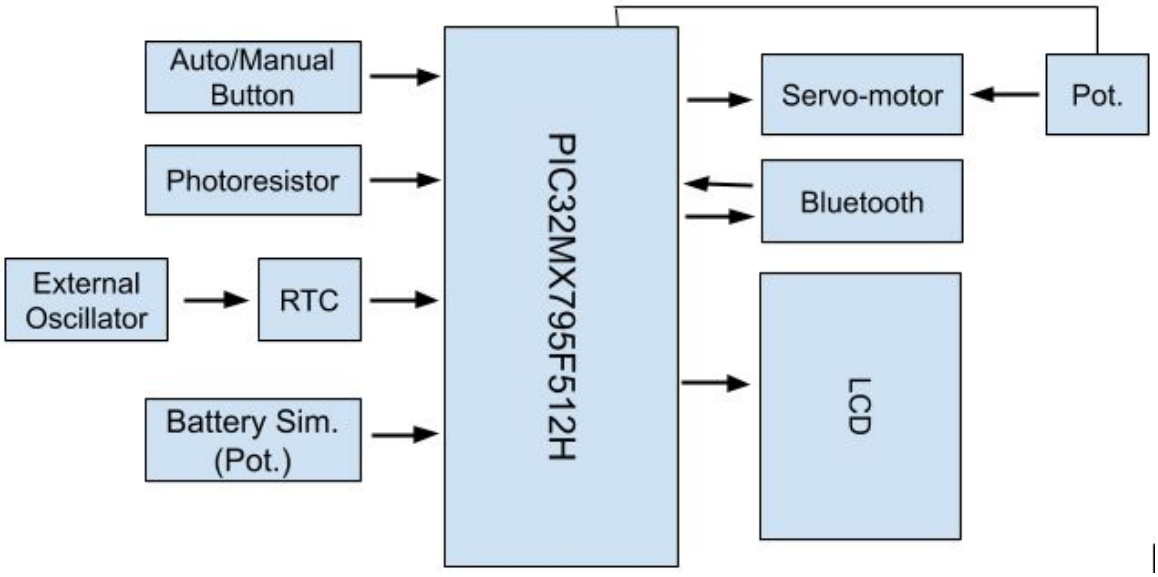
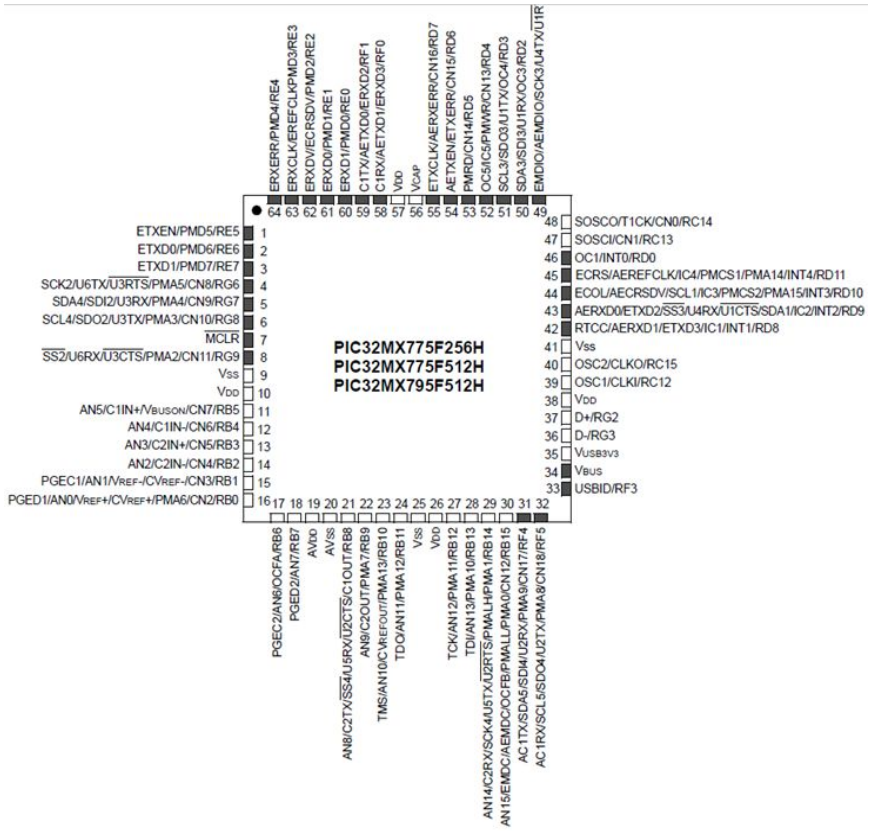


Figure 4: System block diagram

In the block diagram, the PIC32 microcontroller is the center of the system. There are a number of inputs and outputs that branch off of the controller. The inputs to the controller include the auto/manual button, the photoresistor, the RTC with external oscillator, the potentiometer for the battery level simulation, and the bluetooth module. Meanwhile, there are two outputs, one managing the servo motor and second output to the LCD. The servo motor has two inputs: the PIC32 microcontroller that manages the day/night mode and auto/manual modes and the potentiometer, that can finely adjust the blinds when in manual mode; however, the potentiometer input to the servo is received and managed by the PIC. The LCD then provides information about the system to the user, such as the day/night state, ambient light level, position of the servo, and simulated battery level. The following section explains more about the pinout for the PIC32 microcontroller.

PIC32 Pinout:



Pin	Function
B12	Manual control potentiometer
B13	LCD RS pin
B14	Photoresistor
B15	Battery Sim. Potentiometer
C13	SOCSO for RTCC
C14	SOSCI for RTCC
D0	Servo PWM
D2	UART RX
D3	UART TX
D4	LCD E Pin
D5	LCD R/W Pin
E0-E7	LCD Pins

Figure 5: PIC32 pinout and pin descriptions

When designing the system, the function of the pins given in the PIC32 microcontroller reference manual must be considered. There are several pins available for digital I/O on ports B-G. However, port B is the only port that can support ADC, and other pins share functionalities - only one of which can be selected for a given program. Thus, all analog sensors (photoresistor and potentiometers) were equipped to a pin on port B, with the exception of the LCD RS pin. The LCD pinout was very specific. How to pin the LCD to the PIC32 microcontroller was given in both the *Embedded Computing and Mechatronics with the PIC32 Microcontroller* textbook as well as the LCD library used for the course. For the digital display on the LCD, pins E0-E7 were used. Then pin D4 was used to enable the LCD and pin D5 was used for read/write function. Meanwhile, pins D2 and D3 are used for UART communication with the bluetooth module. The pin D2 will receive messages from the module while pin D3 will transmit messages to the module. Lastly, the RTCC module was wired to pins C13 and C14, which are the secondary oscillator in and out pins, respectively (SOCSO and SOSCI).

Bill of Materials:

Item	Supplier	Price Per Unit	Qty.	Total Per Item
PIC32	Microchip	$49.00	1	$49.00
Elegoo Uno Project Kit	Amazon	$29.99	1	$29.99
Photoresistor	Elegoo Kit	Included	1	Included
LCD Display	Elegoo Kit	Included	1	Included
Servo	Elegoo Kit	Included	1	Included
Potentiometer	Elegoo Kit	Included	2	Included
Buttons	Elegoo Kit	Included	1	Included
RTC Module	Microcenter	$4.99	1	$4.99
Bluetooth	Amazon	$9.99	1	$9.99
			Total:	**$93.97**

Figure 6: Bill of Materials

In total, this project would cost under $100 to purchase all the required materials. As the group members are focussing in mechatronics engineering, many of the components were already in possession and did not need to be purchased separately. While some components could be purchased individually at lower individual prices, the value of the project kits allows for several projects to be performed, and for additional components, which may be used for the debugging process. For example, LEDs are commonly used to ensure pins are working properly, or to ensure that programs are performing as expected, so it is recommended to purchase a similar project kit which contains the necessary components.

Project Highlights

Throughout the project, group members were determined to demonstrate the knowledge and skills acquired throughout this course, as well as display other skills gained from various interests. The team was eager to learn more about mechatronics and embedded systems.

Course Topics Utilized:

Interrupts: Interrupts were utilized as a means of managing timers and priorities within the code. Interrupts are used within the file for ADC, bluetooth, button switch, LCD, and the RTC timer substitute.

Digital I/O: Most of the inputs and outputs utilized in this project were digital I/Os with the exception of the photoresistor and potentiometers. The digital I/O pins are fairly simple in that they give a zero if at 0V and a 1 if the voltage is at or near 3.3V. Ports B-G can all be used for digital I/O, while port B can also be used for ADC. The team primarily used buffered I/O throughout the project.

Timers - 16 & 32 bit: Timers were another essential mechanism in controlling the algorithm flow. In some cases the timers were used in addition to an interrupt to allot the proper sample or wait time. For less time consuming processes, like the ADC, the 16-bit timer was used. For processes that require more time, such as the LCD turnover, the 32-bit timer45 was implemented. Following in class examples made this a manageable task.

PWM (Output Compare): Pulse-width modulation was used to control the servo. PWM uses an interrupt with timers 2 or 3 (only timers that can be used for output compare) to generate a pulse that drives the servo's duty cycle. The duty cycle determines the servo position.

ADC: The ADC ports and code were utilized any time there was a sensor that had variable resistance. In the case of the Good Morning Blinds, there were three inputs that required the ADC. These devices were the photoresistor and the two potentiometers. The ADC requires an interrupt that allows the data to be sampled and then converted. From there the bits are mapped to a value that can be utilized in the rest of the code.

UART → Terminal (w/library): UART is the Universal Asynchronous Receiver/Transmitter which allows multiple devices to communicate with each other. In the case of this project the NU32 library allows communication with the board to the developer via PuTTY. This was useful for debugging purposes as the developer could see how far the code had run before encountering an issue. This terminal can also be used for sending a command to the board, but in this case the function was not necessary.

UART → Bluetooth: The bluetooth module chosen for this project communicates through the PIC's UART registers. Since the terminal communication uses UART3, it was decided to use UART1. This way, there was still the option to use PuTTY for debugging. The Android application used for Bluetooth sends a message to the bluetooth module, which is then communicated to the PIC through UART. The message sent between the application and the bluetooth module is read by the PIC by triggering an interrupt. Based on the received message, the interrupt is used to control the Good Morning Blinds system. In this case, the interrupt checks whether the received input correlates to a valid input. Once the input is validated, the day/night state is inverted, and an onboard LED is inverted to confirm the function has been executed.

RTCC w/external oscillator: The RTCC was one of the more challenging systems the team had tried to implement. There were two approaches to take, one using the internal microcontroller module and the other was to implement an external module. Using the internal RTCC would require an oscillator to be attached to the board. Additionally, there was little to no

information as to how to enable and set the clock for the purposes of this project. With the second option of using the external module, team members would have to learn how to code for the I2C bus. Both methods were challenging and in the end the team instead opted for an additional timer to manage the autonomous day and night mode. Though a timer was opted for in the end, this does not mean that members did not learn a great deal about RTCC and its various methods of implementation.

LCD: The LCD acts as a user interface for Good Morning Blinds system. Though the system may seem daunting, the Microchip reference manual provides plenty of information on the pinout and coding of the LCD, including UART. The LCD displays four pieces of information using an interrupt with a 32-bit timer to cycle through each display.

Additionally, the team utilized Github for collaboration. Github is a popular and powerful tool for developers that allows for the sharing of ideas and open source code. It is a tool often used in professional development. The team also designed a modular code structure that enabled parallel workflow. This allowed multiple developers to work on the same project without having to worry about overwrite or merge issues. The extensive use of supplementary .h & .c files also allowed for easy organization and readability of the code. All of this is very important when programs get to be very large. In the end this project resulted in 650+ lines of code. The following directory holds the complete code and demonstrates the teams collaboration: github.com/66hades/ECE5731_Final_Project.

Challenges:

The largest challenge of this project was the Real Time Clock and Calendar (RTCC). The PIC32 board used for the course has an onboard RTCC module; however, after attempting to run example programs from the course textbook, it was discovered that the onboard RTCC does not include an internal oscillator. With this information, an external 32,768 Hz oscillator was wired to the board. Unfortunately, even with the external oscillator, the textbook examples did not perform as expected. To confirm the operation of the external crystal oscillator, a message was output to PuTTY every time a signal was received. Additionally, an I2C RTCC with a 32.768 kHz output pin was used to further confirm the operation. Once it was determined that the external oscillator was functioning properly, the software was inspected.

The RTCC textbook example required multiple holding functions, which would not allow continuation of the program until the previous line had completed. Examples of these functions included turning the clock off or on, and turning on the PIC's secondary clock. To determine whether these functions were working, PuTTY was used once again to output a message once the holding functions had completed.

Initially, all holding functions were working properly, except for the clock turning on. After reading through online forums and confirming the proper PIC registers were set appropriately, the clock was able to move past the holding function. Unfortunately, even once the clock was

turned on, the alarm function did not trigger as expected. The alarm function acts as an interrupt and can be programmed to trigger at a given time, for a given number of repeats (or indefinitely), and for a given time interval between repeats. For demonstration purposes, the alarm was programmed to trigger after 2 seconds of the main program running, every second, indefinitely. However, once the clock was turned on, the interrupt triggered immediately (not after 2 seconds), and did not repeat. After much time spent debugging, consulting the textbook, datasheet, reference manual, online forums, contacting the PIC manufacturer, and working with the TA and professor, the RTCC alarm function was not able to perform as expected. For this reason, and for demonstration purposes, a timer interrupt was implemented in place of the RTCC.

Subsystem Integration was another facet of troubleshooting that took a large amount of time. As individual teammates worked on independent subsystems, the same resources were used by different modules. Examples of non-sharable resources include timers, pins, interrupt priorities, and function names. As each module was added to the full system, resources were reallocated in order for the whole system to function properly. Many resource allocation interferences were avoided by claiming resources in the main file. This central reference for the team greatly cut down confusion and time spent coordinating who was using which files.

Along with resource allocation another task was ensuring each module and the files associated with it could communicate and be seen by other modules as required. In C programming, the structure utilizes a primary “main” function that runs first. The main function resides in one file, but additional files with additional supplementary code can be added. Supplementary files require a “header” file in order to be compiled with the others. The team created a file structure that divided hardware into relevant files with their own header files. While this allowed for flexibility and easy integration, the process of using several files in one program, when functions from one file communicate to several others was a learning experience. Eventually all subsystems worked together.

While it took a little more effort initially to divide up sub systems, our team is confident that was the correct move as the organization and modular benefits far outweighed the difficulties in getting it to work. To elaborate, having different subsystems in separate files allowed the team to work on different components simultaneously without interfering with anyone else’s code. Once ready to combine code, GitHub’s version control was used to merge all changes and evaluate the results. Without separate files this would have been a time consuming process of editing and verifying functionality. Separate files and GitHub ensured that changes to one subsystem would only change that subsystem, and not alter anything else.

One last improvement to implement in this project would be to reduce the operations inside interrupts. Currently the interrupts execute several lines of code. Best practice is to spend minimal time inside interrupts as they change code flow and can disrupt timing in unanticipated ways. To make this improvement, the interrupts would only toggle a flag. During the main runtime a check would inspect the flag and execute functions depending on the flag’s state.

Future Development:

The main topics for future development of this project include RTCC, the battery/solar subsystem, and PID control for user-desired ambient light setting. Reasons for these changes would be to challenge the teams understanding of embedded systems and to improve the quality and functionality of the product.

RTCC: Throughout the project, a large amount of time was spent debugging the RTCC and attempting to use different versions of timekeeping devices, both internal and external to the PIC32. Unfortunately, after all the effort spent on this component, it was not possible to achieve the expected performance from the RTCC or its alarm function. For the future progress of this project, one of the main goals would be to get the RTCC and alarm working properly. Currently, the RTCC setup program seems to be working, and the clock is able to be turned on and its function verified. However, the main issue is that the alarm function is not producing the desired results. After starting the program, the alarm interrupt triggers immediately, but not when expected, and the repetition of the function is not working. Therefore, the main goal for the future would be to make the alarm functional through further debugging and setup.

Interrupt improvements:
Best practice is to spend as little time as possible inside interrupts. This allows the program to execute interrupts as they are triggered, without the need to create a backlog of interrupt events. The other possible issue is that a higher-priority interrupt gets triggered, and the lower-priority one does not get fully executed. Rather than performing functional code inside the interrupt, flags should instead be used. As an example, when the day/night interrupt is triggered, a flag should be inverted, allowing the interrupt to include very few lines of code. Outside of the interrupt, another function would be used to read the flag and perform the necessary actions depending on the state of the interrupt flag.

Battery/Solar:
For the purpose of demonstrating this project, the battery charge level was simulated with a potentiometer. 0-100% battery level was mapped to the ADC's 0-1023 reading. However, in a production version of this, a battery would be installed in place of the potentiometer, and the current battery voltage would be displayed on the LCD. Additionally, a solar charger would be integrated for an environmentally friendly

PID:
Ideally, PID control would have been implemented to maintain the light-level of the environment with regards to user preference. In future development the PID control would be included and then, in the autonomous mode, the input from the photoresistor would not only be displayed to the user, but also be used as a factor for the blind angle. With this method, there would be consistent lighting from the outside regardless of the time of day, with night time being the exception.

Conclusion:

Throughout the Good Morning Blinds project, the team managed to accomplish a great deal. The team took an original idea and produced a functional proof of concept with 650+ lines of code. This project also successfully demonstrated all the course objectives which clearly shows the students' understanding of the material. Students also learned and utilized additional tools that can be used throughout their professional development such as GitHub, PuTTY, and Visual Studios IDE. The use of GitHub allowed easy collaboration and version control management, while PuTTY served as an excellent debugger, and Visual Studio provided intuitive syntax color coding/highlighting as well as gave exposure to a tool that is widely used in industry. The end result is that the students have gained knowledge and experience that will make them more effective mechatronics engineers.

**Full references list can be found on GitHub repository README
https://github.com/66hades/ECE5731_Final_Project.git