Spring 2024

Children's Hospital Animatronic

Erin Keller
eek39@uakron.edu

Wesley Cunningham
wec16@uakron.edu

Dylan Mueller
djm269@uakron.edu

Carl Richter
cdr62@uakron.edu

Follow this and additional works at: https://ideaexchange.uakron.edu/honors_research_projects

Part of the Robotics Commons

Please take a moment to share how this work helps you through this survey. Your feedback will be important as we plan further development of our repository.

Recommended Citation
Keller, Erin; Cunningham, Wesley; Mueller, Dylan; and Richter, Carl, "Children's Hospital Animatronic" (2024). Williams Honors College, Honors Research Projects. 1862.
https://ideaexchange.uakron.edu/honors_research_projects/1862

This Dissertation/Thesis is brought to you for free and open access by The Dr. Gary B. and Pamela S. Williams Honors College at IdeaExchange@UAkron, the institutional repository of The University of Akron in Akron, Ohio, USA. It has been accepted for inclusion in Williams Honors College, Honors Research Projects by an authorized administrator of IdeaExchange@UAkron. For more information, please contact mjon@uakron.edu, uapress@uakron.edu.
Children’s Hospital Animatronic

Senior Project Final Report

Design Team Number 2

Wesley Cunningham

Erin Keller

Dylan Mueller

Carl Richter

Dr. Hamid Bahrami

3 May 2024
Individual Contribution of Erin Keller

This project consists of several subsystems: the 3D printed body, audio system, power supply, servo motors, microcontroller, and phone application. My main focus working on this project was on the audio system and power supply. I have spent the past three semesters designing and implementing these subsystems as well as assisting with the microcontroller subsystem and the overall integration of the system. The largest challenge that the team had to overcome was creating a reliable Bluetooth connection between the microcontroller and the phone application. I developed code that allowed the selection from the app to be read into the microcontroller, and ultimately this work contributed to the final Bluetooth code, which was simple and efficient. In addition, I developed and carried out a set of tests to determine which conditions led to an unstable connection between the app and microcontroller. These tests allowed the team to understand the issues and how to troubleshoot them quickly. The details of my role developing the audio system and power supply are well detailed in the remainder of the report. Overall, this project was a wonderful exercise in teamwork, research, and testing.
Table of Contents EK

1. Problem Statement
   1.1. Need
   1.2. Objective
   1.3. Background
   1.4. Marketing Requirements

2. Engineering Analysis
   2.1. Circuits
   2.2. Electronics
   2.3. Signal Processing
   2.4. Communications
   2.5. Electromechanics
   2.6. Computer Networks
   2.7. Embedded Systems
   2.8. Controls

3. Engineering Requirements Specification

4. Engineering Standards Specification
   4.1. Safety
   4.2. Communication
   4.3. Data Formats
   4.4. Design Methods
   4.5. Programming Languages
   4.6. Connector Standards
5. Accepted Technical Design
   5.1. Hardware Design
   5.2. Software Design
6. Mechanical Sketch
7. Team Information
8. Parts Lists
9. Project Schedule: Gantt Chart
10. Conclusions and Recommendations
11. References
12. Appendices
List of Figures

1. Figure 1: Audio File Conversion Flowchart
2. Figure 2: Audio Circuit
3. Figure 3: PWM generated signal from the PCA9685
4. Figure 4: Level 0 Block Diagram
5. Figure 5: Level 1 Block Diagram
6. Figure 6: Phone App Interface Functional Decomposition
7. Figure 7: Development Board Functional Decomposition
8. Figure 8: Animatronic Body Functional Decomposition
9. Figure 9: Audio System Functional Decomposition
10. Figure 10: Power Source Functional Decomposition
11. Figure 11: Level 0 Software Diagram
12. Figure 12: Phone App Interface
13. Figure 13: Bluetooth Schematic
14. Figure 14: Bluetooth Communication
15. Figure 15: Mechanical Sketch
16. Figure 16: PWM generated from PCA9685’s datasheet
17. Figure 17: Servo motor circuit.
18. Figure 18: Steps to move a servo
19. Figure 19: Description of which channels control which body parts as well as the servos default position
List of Tables

1. Table 1: Level 0 Block Diagram Table
2. Table 2: Parts List
3. Table 3: Material Budget List
4. Table 4: Beginning of Fall Semester Gantt Chart Schedule
5. Table 5: Mid Fall Semester Gantt Chart Schedule
6. Table 6: End of Fall Semester Gantt Chart Schedule
7. Table 7: Spring Semester Gantt Chart Hardware Schedule
8. Table 8: Spring Semester Gantt Chart Software Schedule
Abstract

According to G. Beraldo, et al.’s presentation “A Preliminary Investigation of using humanoid social robots as non-pharmacological techniques with children,” hospitalized children experience high levels of stress and loneliness, and their mood can be improved by interacting with robots. The objective of this project is to design and prototype an animatronic and phone application that will provide comfort and entertainment to hospitalized children. A mobile application will allow users to select from a list of reactions for the animatronic to exhibit. The application will communicate with a series of motors via a microcontroller. The motors will move the animatronics arms, ears, head, and facial features to display the user-selected reactions. An audio system will produce noises that convey these emotions as well. The entire system will be powered by rechargeable batteries. The proposed animatronic will provide hospitalized children with an immersive experience that brings them joy and sympathizes with them.

Key Features of Product

- Interactive Phone Application
- Moving ears, eyes, mouth, and arms
- Produces audio
- Displays a variety of expressions and reactions
1. Problem Statement

1.1. Need DM

Many people who have to stay in hospitals for prolonged periods often have a stressful or traumatic experience depending on their situation. This would be even worse if a child had to stay in a hospital for a prolonged period. Estimates of emotional problems resulting from hospital experience have been reported to vary from 10% to 30% for severe psychological distress to as much as 90% for slight emotional upset in hospitalized children [Yap, 1988]. Children have little to no life experience and may be confused as to why they suddenly have been ripped out of their daily routines and are forced to stay in a hospital. Pediatric hospitals must go beyond a narrow focus of the medical aspect of treating children’s illnesses and improve the implications of the hospitalization’s distress on children [Rokach, 2016]. A device is needed to interact with children to reduce the stressful and anxiety-filled environment that hospitals have.

1.2. Objective WC & CR

The objective of this project is to design an animatronic for children’s hospitals that would help bring joy to children who are in pain. The animatronic will be a cute, friendly looking creature. The system would display multiple reactions such as happiness, sadness, or excitement in order to provide a child with comfort and emotional support. A phone application would be available to control and monitor the animatronic. The device would also be mobile, so that it is able to visit multiple children daily.
1.3. Background WC, EK, DM, CR

At any age, hospitalization and health problems can be scary and isolating. In particular, hospitalized children experience high levels of stress and loneliness [3]. It is necessary to support hospitalized children by providing them with entertainment and company, such as by visiting them with friendly animatronics. In 2019, a study published by the Institute of Electrical and Electronics Engineers (IEEE) found that after visiting hospitalized children with 2 humanoid social robots, the children experienced a sharp increase in positive emotions. In addition, more than 85% of the children reported that they would like to play with the robots again [3]. This study suggests that robotic characters are able to lift the spirits of hospitalized children.

In addition to sick children, robotic toys have also been proven to be useful therapeutic tools for children with autism spectrum disorder (ASD). Children with ASD typically struggle with social isolation and show a lack of interest in the world around them. However, they are often attracted to toys with lights and sounds, and a study published by IEEE in 2013 found that children with ASD quickly became interested in learning to operate and engage with robotic toys [4]. This suggests that animatronics could be used to help autistic children engage with their environment, thus improving their social skills and learning abilities.

An animatronic that appears to be a cute, friendly creature, such as an animal or character from pop culture would be a familiar and engaging device for children to interact with. A visit from an animatronic creature could be an
effective way for hospitals to engage with and comfort their young patients. This animatronic’s expressions and movements will be wirelessly controlled by the user via a smartphone app to allow the creature to interact with children with a wide range of needs. The user would be able to select pre-set movement patterns for the animatronic to perform in order to interact with the patient. The animatronic creature will utilize multiple motors to control movement of its eyes, ears, and mouth to mimic facial expressions associated with different emotions, such as happiness, excitement, or sadness. The animatronic creature will provide hospitalized children with entertainment and comfort by mimicking common emotions.

In order to develop the best possible animatronic, it is necessary to analyze similar technologies. The animatronic creature will have a range of facial expressions and movements, which requires careful control over the eyes and mouth of the character. William Willert’s patent, US6793553B2, proposes the use of a single-motor device that allows a doll’s mouth and eyes to open and close simultaneously. This system can only produce up and down movements [5]. Similarly, the creature animatronic will utilize motors to produce facial expressions. The creature animatronic will be more complex than the doll in this patent since its eyes will be more expressive, including both side to 4 side and up and down motions, and the eyes and mouth will move independently. In addition to the eyes and mouth, the creature animatronic’s ears and arms will also have motor-controlled movements. The motors will be controlled wirelessly via an electric motor drive. It is necessary to control the movement of many of the
creature’s features to ensure that the system appears life-like. By providing a more life-like experience, it will encourage children of all ages to engage with the robot.

Another important aspect of the creature animatronic is that it will be compatible with a phone application that allows user control of the animatronic’s expressions and movements. Deborah Simmon’s patent, US20140011423A1, proposes a doll that users can move and converse with via a transmitter cell phone which resembles a smart phone. This phone, which has a computer chip and transceiver, enables the doll to react verbally and physically based on signals received via the computer chip and transceiver in the doll itself [6]. The principle of wireless communication is integral to the animatronic system. Remote user control will allow the animatronic to seem more life-like and will allow it to serve a wider range of patients since relatable emotions may be selected for interactions with each child. In addition, allowing user control of the system is important when dealing with children who may be easily overwhelmed. This will ensure that if the child does get overwhelmed or upset, the animatronic’s motion can quickly be stopped so the child is able to calm down.

In a hospital setting, it is necessary that the creature animatronic will be easy to repeatedly sanitize. This suggests that the animatronic should be waterproof. Underwater robots are becoming more common, and in particular, silicone rubber based waterproof gloves can be used to waterproof robotic hands. These gloves provide the robotic hands with both grip while wet and protection from water damage [7]. The creature animatronic will have a similar rubber
coating to prevent water from damaging the internal components during sanitization processes. Similar to the robotic hand, it is necessary that the coating is flexible enough to allow for movement of the animatronic’s features. In addition, utilizing a material that is not slippery will help prevent slips and falls that could cause unnecessary damage.

The creature animatronic will also need to be easily controlled by anyone. User-friendly interfaces for the operation of robots are rapidly evolving. One such system utilizes an 8051 microcontroller interfaced with an Android App that communicates with the robot via a Bluetooth module [8]. This system is intended to provide children with an approachable way to learn about robotics. By utilizing a simple system with an intuitive interface, it is possible to provide anyone with the opportunity to operate animatronics. A simple interface ensures that people who do not speak English, who have little education, or who have little experience with technology can quickly become proficient users. It is important to ensure a child from any background can benefit from this system, otherwise it could make the child feel more upset. Similarly, it is important to have an animatronic that interacts through methods other than words, so that children who do not speak English or cannot hear will still enjoy the animatronic’s visit.

Many common animatronics, such as Halloween decorations, demonstrate pre-set movements that are not user controlled [9]. This may lead some children to perceive the animatronic as unpredictable and scary. On the other hand, they could perceive it as boring since it reacts randomly instead of reacting to specific situations. The creature animatronic will also have set movements and
expressions, however, its movements and 6 expressions will be remotely controlled by the user [9]. This element will allow the animatronic to be more suitable to children with a range of personalities, needs, and likes. In addition, while many common animatronics are not waterproof, the creature animatronic will be made of waterproof material that will ensure sanitization is easy, safe, and effective, thus allowing the character to visit multiple patients each day.

Current animatronics are often prone to water damage and are not user-friendly [9]. By providing a simple, intuitive interface to allow user control of animatronics, it is possible to bring the joy of animatronics to anyone. In addition, by developing a waterproof body for the animatronic, it is possible to allow for repeated sanitization, which will allow the animatronic to be used in hospitals to comfort multiple patients each day. Animatronics, especially older ones, have a reputation for being creepy due to their lack of life-like expressions. By including multiple motors to control the animatronic’s expressions, as well as designing it to represent a friendly and cute character, this reputation can be challenged. Overall, an animatronic representing a friendly creature will be a great addition to children’s hospitals, and can be realized with motors, smartphone applications, and a flexible waterproof coating.

1.4. Marketing Requirements EK

1. The system should be able to withstand repeated sanitization procedures.

2. The system should be rechargeable with a reasonably long battery life.
3. The system should display multiple greetings/reactions to comfort hospitalized children.

4. The system should be easily controlled by a wireless device and user-friendly app.

5. The system should be safe and portable.

2. Engineering Analysis

   2.1. Circuits

   2.1.1. Power Source EK

   A power source is necessary to provide power to the microcontroller, Bluetooth receiver, audio circuit and speaker, and the servo motors that control the facial expressions and gestures of the animatronic. According to the datasheets in Appendices A and B, the MG90D and MG995 servos will require 6.0V. However, the MG995 servos are supplied with 7.0V to support the large number of servos operating simultaneously within the animatronic. During operation, the combined eighteen servo motors can draw up to 1A for about half of a minute. At rest, the servos draw around .05A altogether. Assuming the animatronic operates for about 2 minutes every 10 minutes, the servo motors can draw around 1.5A every 10 minutes. This means the servo motors require around 9Ah. The PCA9685 requires about 30mAh. To allow for a battery life of about 2 hours, the power supply should be able to support 21Ah.
This would require 3 parallel connected 7V, 7Ah batteries. For this project, it was decided to utilize power supplies instead of batteries to ensure smooth operation for a few hours at a time.

The microcontroller and Bluetooth receiver will require a maximum of 50mA, the audio circuit and speaker will require a maximum of 2000mA during operation. Two rechargeable AA batteries connected in series can provide 3.0V with a life of 2500mAh, which will allow these components to last for longer than the servo motors. In addition, using AA batteries will make the device user-friendly as the average person is familiar with installing, charging, and removing AA batteries.

2.2. Electronics

2.2.1. **Audio System** EK

The audio system must produce audible noise between 5 and 85dB. The microcontroller produces a PWM waveform based on data from an audio file that has been processed according to the flowchart in Figure 1.
This PWM signal is then passed through a BJT amplifier circuit, using the BC547B transistor, and that outputs the amplified signal to an 8 ohm speaker. There is a 5V supply for the speaker. The full circuit is illustrated in Figure 2.
Figure 2: Audio circuit.

The code for the audio circuit is shown below. Note that there is no audio data included in this code.

```c
int count = 0;

int selection = 1;

void initAudio( void)
{
    // init TMR3 to provide the timebase
    T2CON = 0x8000; // enable TMR3, prescale 1:1, internal clock

    PR2 = 400-1; // set the period for the given bitrate

    // init PWM - set the initial duty cycles
    OC1R = OC1RS = 0;
}
```
// activate the PWM module

OC1CON = 0x0006; // E for TMR3, 6 for TMR2

_T2IF = 0; // clear interrupt flag

_T2IE = 1; // enable TMR3 interrupt

} // initAudio

void _ISR_T2Interrupt( void)
{

if(selection == 1) // excited
{

if(count%5==0)
{

OC1RS = excited[count/5];

}

count++;

if(count>=6135*5) // if count = length of signal
{

}
count = 1;

// clear interrupt flag and exit

_T2IF = 0;

// selection++;

}

// clear interrupt flag and exit

}

if (selection == 2) //force

{

if (count % 5 == 0)

{

OC1RS = force[count / 5];

}

count++;

if (count >= 1836 * 5) //if count = length of signal

{

count = 1;

// clear interrupt flag and exit
_T2IF = 0;

// selection++;

}

// clear interrupt flag and exit

}

if(selection == 3) //happy
{

if(count%5==0)
{

OC1RS = happy[count/5];

}

count++;

if(count>=7226*5) //if count = length of signal
{

count=1;

// clear interrupt flag and exit

_T2IF = 0;

}
/ clear interrupt flag and exit

} 

if(selection == 4) //sad 

{ 

    if(count%5==0) 

    { 

        OC1RS = sad[count/5]; 

    } 

    count++; 

    if(count>=7837*5) //if count = length of signal 

    { 

        count=1; 

        // clear interrupt flag and exit 

        _T2IF = 0; 

        // selection++; 

    } 

    // clear interrupt flag and exit 

}
_T2IF = 0;

} // T2 Interrupt

int main(void)
{

    SYSTEM_Initialize();

    initAudio();

    while (1)
    {

    } // while

    return 1;

} // main

2.3. Signal Processing

2.3.1. Microcontroller EK, CR

This project requires the generation of up to 19 separate PWM signals nearly simultaneously. Unfortunately, the PIC24
microcontroller can only generate 5 PWM signals. A PWM chip that can communicate with the PIC24 via I²C is used to overcome this limitation. Specifically, the PCA9685 chip is used, which can send up to 16 PWM signals simultaneously. Two separate chips are used in this project: one for the smaller MG90D servo motors and one for the larger MG995 servo motors. In addition to the two PWM chips, the output compare on the PIC24 is used to produce the PWM signal for the audio circuit.

2.3.2. **PCA9685 generating Pulse Width Modulation (PWM) Signals**

![Figure 3: PWM generated signal from the PCA9685](image-url)
Figure 16: PWM generated from PCA9685’s datasheet

The PCA9685 can only provide 30mAh, which is not enough power or enough PWM channels for all 18 servo motors. So, two separate chips are used in this project: one for the smaller MG90D servo motors and one for the larger MG995 servo motors. Both the MG90D and the MG995 operate off a PWM signal with a period of 20ms. By default, the PCA9685 generates a PWM signal period of 5ms. The device has a pre-scaler register that can be used to increase the period of the PWM signals. To use the pre-scaler register you must first put the device into sleep mode. The oscillator that the device uses to generate signals is disabled while in sleep mode. The device after being put into sleep mode and setting the pre-scaler is put back into wake mode so PWM signals can be generated. The device has four registers per PWM channel for a total of 64 registers. After the device is set back into wake mode the four registers that need to be set for a given channel to generate a PWM signal are OFF_H, OFF_L, ON_H, and ON_L. An example PWM from the device’s datasheet is shown in figure
16. The four registers and what part of the signal their values represent are shown in this figure. Once these four registers are set with appropriate values the PWM will be generated for the channel the four registers belong to. How to calculate the values for these four registers is shown in the PCA9685’s datasheet. How to set up the device is explained how to do, in detail, in section 115.1.2 Control servo motors. The circuit needed to supply the servo motors with the appropriate PWM signal from the microcontroller is shown in Figure 18.

Figure 17: Servo motor circuit.

2.4. Communications

2.4.1. Bluetooth Receiver EK
A Bluetooth receiver module will allow communication between the phone application and the microcontroller. Bluetooth is an ideal communication form for this project because it is relatively low power and can communicate up to distances of 300 feet, which is more than adequate for this project. The receiver must be capable of receiving the user’s selected behavior, which will be represented by a pre-determined symbol or letter. This should take less than 0.5 seconds to ensure smooth, life-like operation. In addition, the receiver should be less than 1 cubic inch and weigh less than 8 grams to support the goal of having a lightweight and portable system.

2.5. Electromechanics

2.5.1. 3D Printed Body Parts and Facial Features EK, CR

The animatronic will be about 18 inches tall with a diameter that is around 12 inches. The animatronic’s skeleton will be composed of a 3D printed head, neck, arms, and a body. The animatronic will also have 3D printed eyes, eyelids, ears, and lips. To account for any inevitable printing failures, a total of four 1kg spools will be ordered. Each individual part must be less than 2kg to ensure that they are light enough for the servo motors to operate.

2.5.2. Servo Motors WC
The animatronic’s body parts and facial features will be moved by a series of servo motors. For lighter components MG90D servos will be used. The MG995 servos have slightly faster movements and greater torque so they can handle heavier components. There will be seven MG90D servos in the animatronic and eleven MG995 servos. Four MG90D servos will be required for the eye movements; there will be one for each set of eyelids and two for the eyes’ left and right movements. Using a set of servos to control both eyes’ left and right movements will satisfy the engineering requirement to make the eyes appear to be moving simultaneously. Only one of the animatronic’s arms will need to move for the range of behaviors it displays. An MG995 servo will be used to move the entire 3D-printed limb instead of the less powerful MG90D servo since this is a slightly heavier component. Three MG995 servos will control the neck of the animatronic. The neck’s left and right tilt will be controlled by two of the servos, while the twisting of the animatronics neck will be controlled by the remaining MG995 servo. Only one MG995 servo will be required to open and close the mouth. The ears will require two servos each. One MG995 servo will control the up and down movements of the ear while an MG90D servo will draw the ear back and forward.

2.6. Computer Networks

2.6.1. Bluetooth WC, DM
The phone application will communicate with the microcontroller via Bluetooth. The Bluetooth communication standard is the 2.4 GHz spectrum band. The reason why this communication standard is being used is because it is a relatively low-power communication that can exist over relatively large distances. While distance is not a concern for the current use-case of this animatronic, since it will constantly be in the same room as the phone that is connected, it may be useful for future applications of the product.

2.7. Embedded Systems

2.7.1. Microcontroller CR

The microcontroller must be capable of connecting to a Bluetooth receiver which operates on the 2.4GHz spectrum band. The microcontroller must be able to produce 19 PWM signals. The microcontroller must be capable of storing digital audio data to generate PWM signals with. The PIC24 microchip should be able to support these functions while being small, lightweight, and relatively low power consumption, thus making it a reasonable choice for this project.

3. Engineering Requirements Specification

3.1. Phone Application WC, DM
3.1.1. The phone application must allow users to select one of four reactions for the animatronic. This selection must be transmitted to the microcontroller via Bluetooth.

3.1.2. The phone application must connect to the microcontroller via Bluetooth in less than 10 seconds to create an immersive experience.

3.2. **Body Parts and Facial Features** WC, DM, CR, EK

3.2.1. The servo motors must be powerful enough to manipulate the 3D printed body parts and facial features of the animatronic to perform preset movements based on the signal from the microcontroller.

3.2.2. The servo motors must move the body parts in a way that appears to be simultaneous, realistic, and cohesive when they receive the PWM signal from the microcontroller.

3.2.3. The limbs and features of the animatronic must perform 4 separate behaviors determined by the PWM signal from the microcontroller.

3.2.4. The ears and arms must be able to move independently of one another.

3.2.5. The mouth must open 1ms before audio is played so that the animatronic appears to be speaking when audio is played.

3.2.6. The neck must move circularly at a speed of at least 5 seconds per 180 degrees, with a maximum range of motion of 180 degrees. The neck will also tilt the head on four axes arranged in a square.

3.3. **Microcontroller** EK, CR, DM
3.3.1. The microcontroller must interface with a Bluetooth receiver that receives the user’s selected reaction from the phone application.

3.3.2. The microcontroller must be capable of storing the two-bit selection from the Bluetooth receiver.

3.3.3. The microcontroller must interpret the user’s input from the Bluetooth receiver and determine what the appropriate responses are for the body parts and audio circuit.

3.3.4. The microcontroller must be capable of sending PWM signals to all 18 servo motors within 3 seconds of each other to move the body parts.

3.3.5. The microcontroller must be able to store raw digital voice files and transmit the file corresponding to the user's selected emotion to the audio circuit via an analog electrical signal.

3.4. Audio System EK

3.4.1. The audio system will produce audible noise with a maximum level of 85 decibels.

3.4.2. The audio system must produce a noise corresponding to the user’s selection from the phone application based on an analog electrical signal from the microcontroller.

3.5. Power Source EK

3.5.1. The power source must provide the audio circuit, microcontroller, Bluetooth receiver, and 18 servo motors with enough power to sustain operation for 2 hours.

3.5.2. The power source must be capable of fully recharging overnight.
4. Engineering Standards Specification

4.1. Safety EK

4.1.1. Throughout the project, the rules and guidelines for the University of Akron senior design laboratories will be followed.

4.2. Communication EK

4.2.1. Bluetooth will be used to facilitate wireless communication between the phone app and the microcontroller.

4.2.2. Pulse width modulation will be used for communication between the microcontroller and servo motors.

4.3. Data Formats EK

4.3.1. Audio data will be stored as wave files.

4.4. Design Methods DM

4.4.1. The project was initiated with a top-down approach, aiming to design the baby Yoda animatronic. The initial focus was to conceptualize the design, although there were uncertainties regarding the specific design procedures. Consequently, the project was deconstructed into nine primary subsystems: Phone app, Microcontroller, Servo Motors, 3D Printed Body, Power, and Audio. The underlying concept involved the phone app transmitting a Bluetooth signal to the microcontroller, which would then relay pre-defined instructions to the various body and audio components. Upon identifying the requisite subsystems for implementation, each was further broken down into its individual components.

4.5. Programming Languages DM, WC, CR
4.5.1. The phone application will be programmed via the React Native Framework.

4.5.2. The microcontroller will be programmed via C programming language.

4.6. Connector Standards EK

4.6.1. USB-C connectors will be primarily used in this project.

5. Accepted Technical Design

5.1. Hardware Design

The Level 0 Block Diagram in Figure 1 describes the project from a basic standpoint. The main subsystem for the project is the microcontroller, which requires two inputs: power, and communication from a phone application. Then, the microcontroller will output eye, neck/head, and ear movement as well as audio.
Table 1 describes the design in more detail, elaborating on the designers, inputs, and outputs, as well as providing a description for the basic operation of the project.

Table 1: Level 0 Block Diagram Table WC

<table>
<thead>
<tr>
<th>Module</th>
<th>Animatronic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designer</td>
<td>Wesley Cunningham, Erin Keller, Dylan Mueller, Carl Richter</td>
</tr>
<tr>
<td>Inputs</td>
<td>Microcontroller, Battery</td>
</tr>
<tr>
<td>Outputs</td>
<td>Eye Movement, Head Movement, Ear Movement, Audio</td>
</tr>
<tr>
<td>Description</td>
<td>The animatronic will receive power from a battery and some form of communication from a phone application. Then parse the data and move the proper body parts of the animatronic.</td>
</tr>
</tbody>
</table>

Figure 4 and Table 1 provide a general outline for the project, but these ideas are described in much more detail in Figures 5-10.

Figure 5: Level 1 Block Diagram WC, EK, DM
Figure 6: Phone App Interface Functional Decomposition WC, EK, DM

<table>
<thead>
<tr>
<th>Module</th>
<th>Front End Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designer</td>
<td>Wesley Cunningham, Dylan Mueller</td>
</tr>
<tr>
<td>Inputs</td>
<td>User Input</td>
</tr>
<tr>
<td>Outputs</td>
<td>Stored Data</td>
</tr>
<tr>
<td>Description</td>
<td>The user will select a prompt presented by the phone application. Once the selection is made, the front end code will store the information.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module</th>
<th>Back End Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designer</td>
<td>Wesley Cunningham, Dylan Mueller</td>
</tr>
<tr>
<td>Inputs</td>
<td>Stored Data</td>
</tr>
<tr>
<td>Outputs</td>
<td>Bluetooth Signal</td>
</tr>
<tr>
<td>Description</td>
<td>The back end code will access the stored data to determine what the microcontroller needs to do. This data will be sent as a bluetooth signal to the bluetooth receiver.</td>
</tr>
</tbody>
</table>

Figure 7: Development Board Functional Decomposition WC, EK, DM

<table>
<thead>
<tr>
<th>Module</th>
<th>Bluetooth Receiver</th>
<th>Module</th>
<th>Microcontroller</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designer</td>
<td>Carl Richter</td>
<td>Designer</td>
<td>Carl Richter</td>
</tr>
<tr>
<td>Inputs</td>
<td>Bluetooth Signal, 5V DC Power</td>
<td>Inputs</td>
<td>Digital Signal, 5-12V DC Power</td>
</tr>
<tr>
<td>Outputs</td>
<td>Digital Signal</td>
<td>Outputs</td>
<td>PWM Signal and Raw Digital Voice Signal</td>
</tr>
<tr>
<td>Description</td>
<td>Receives a bluetooth signal from the phone application. Sends a digital signal to the microcontroller describing the user's input.</td>
<td>Description</td>
<td>Receives a digital signal from the bluetooth receiver. Uses Pulse Width Modulation to send signals corresponding to the user-selected expression/gesture to the servo motors and audio amplifier circuit.</td>
</tr>
</tbody>
</table>
Figure 8: Animatronic Body Functional Decomposition WC, EK, DM

Figure 9: Audio System Functional Decomposition WC, EK, DM
The Level 1 Block Diagram in Figure 5 provides an in-depth explanation of the animatronic’s operation and design. The figure flows from the phone application to the development board to the animatronic body and audio system, all of which is connected to a power source. The phone application subsystem, described at length in Figure 6, is composed of both front-end and back-end code. This system takes in user input and produces a Bluetooth signal which is sent to the Bluetooth receiver. The Bluetooth receiver, which is powered by a source, sends data to the microcontroller. The microcontroller, which is also powered by a source, makes decisions based on the data from the receiver and sends PWM signals to each servo motor as well as the audio circuit. The Bluetooth receiver and microcontroller make up the development board subsystem, which is
described further in Figure 7. The animatronic body is composed of 3D printed body parts and facial features that are positioned by a series of 18 servo motors. The servo motors move the body parts to their programmed positions based on the PWM signal received from the microcontroller. This subsystem is described in Figure 8. The audio system, described further in Figure 9, is fed by a PWM signal from the microcontroller and is composed of a simple amplifier circuit and a speaker, which will produce audible noise.

Ultimately, these subsystems will create an animatronic. Users will have the opportunity to select from four reactions, and the subsystems described in Figures 5-10 will work together to create the corresponding response demonstrated by the animatronic’s movements. Note that the tables in Figures 6-10 describe the inputs, outputs, designers, and operation of each subsystem in depth.

5.2. Software Design WC & DM
Figure 11: Level 0 Software Diagram DM, WC

DM

Starting on the left side of Figure 11 the initial input to the system will be a user input to a phone app interface. The code will display four buttons and each button will be tied to an integer. Once the user makes their selection the code will send the integer over Bluetooth to the microcontroller. The microcontroller will then take the received integer value and make the correct decision based on what the user selected. The microcontroller will then send the correct PWM signals to the servos and the audio circuit.

5.2.1. Phone application interface WC

The phone application is fully programmed in REACT Native. When a user opens the phone application, they are greeted with 4 buttons each containing a unique emoji that relates to the emotion the given button transmits to the animatronic. If the user tries to
press one of the buttons before the app is connected to the
animatronic, they will be prompted with a popup telling them to
connect to Grogu first.

In order to the connect to the animatronic the user will need
to press the “Connect to Grogu” button. This will then open up a
menu that will only show the Grogu animatronic. The user can
then select on the listed Bluetooth device and a modal will display
saying that the user connected or that an error occurred when the
user tried to connect to the device. Now that the animatronic is
connected the user can close out of the Bluetooth Screen by
pressing the close button at the bottom of the app.

Once returned to the main menu the user can now press a
button to send an emotion to the microcontroller in the
animatronic. The app uses the react-native-ble-manager package in
order to communicate with the animatronic. Each time a button is
pressed the following series of events occurs. Once the button is
pressed a function called handlePress is called, this function is
passed one parameter which is the expressionChoice variable. The
happy expression button has an expressionChoice of 1, the sad
expression has an expressionChoice of 2, the excited expression
has an expressionChoice of 3, and lastly the force expression has
an expressionChoice of 4.
Once inside the handlePress function there is a series of if statements that decide which button was pressed based on the expressionChoice parameter that was passed to the handlePress function. Inside each if statement there is a “message” variable that will tell the user that the selected expression is being sent to the microcontroller. Inside the if statements there is also a variable called choice that will assign a special character to the choice variable depending on what button was pressed. The Bluetooth module that was used for this project would send a long string of characters anytime a device is connected. In order to make sure that the choice variable would properly be received each expression was assigned a special ASCII character that did not appear in the connection string. When sending the happy expression the “choice” variable is assigned the value of “!”, when sending the sad expression the “choice” variable is assigned the value of “?”, when sending the excited expression the “choice” variable is assigned the value of “#”, and finally when sending the force expression the “choice” variable is assigned the value of “+”.

After the “choice” variable is assigned the next operation in the handlePress function is to check whether the device is still connected, if the device is no longer connected then the program will throw an error and tell the user “Failed to send expression to Grogu:” then the message will list whatever the error was. If the
device is still connected the handlePress function will continue by calling the writeToCharacteristic function that is defined in the useBLE.tsx file.

The writeToCharacteristic function takes four parameters, the first being the deviceId of the connected device. This parameter is of type string. The second parameter is the serviceUUID which lets the function know if the device is transmitting or receiving data. The third parameter is the characteristicUUID which is unique to the device and establishes client and server roles. The final parameter is data, data contains the choice variable that was assigned prior to when the writeToCharacteristic function was called. Inside the writeToCharacteristic function a buffer is created that converts the sent data into a JSON format so that the microcontroller can parse what is being sent. Finally, the BleManager.write method is called which is part of the react-native-ble-manager package. The BleManager takes the same parameters as the writeToCharacteristic function. The rest of the processes happen behind the scenes thanks to the write function. Once the expression is successfully sent an alert is displayed to the user notifying them that the expression was successfully sent to the microcontroller.

Once a user is done using the Grogu animatronic they can press the disconnect button on the main screen of the application.
When this button is pressed it disconnects the phone from the RN4870 so that another user can connect to the animatronic from their own phone.

5.2.1.1. Code for the app interface:

```javascript
6. import React, { useState } from 'react';
7. import { View, Alert, Button } from 'react-native';
8. import CustomButton from './Buttons/customButton';
9. import BluetoothScreen from './BluetoothScreen';
10. import styles from './styles/styles';
11. import useReactBLE from './useReactBLE'; // Change import to useReactBLE
12.
13. const MyComponent = () => {
14.   const [modalVisible, setModalVisible] = useState(false);
15.   const [connectedDevice, setConnectedDevice] = useState<string | null>(null); // Update device state type
16.
17.   const openModal = () => {
18.     setModalVisible(true);
19.   };
20.
21.   const closeModal = () => {
22.     setModalVisible(false);
23.   };
24.
25.   const {
26.     scanForPeripherals,
27.     requestPermissions,
28.     connectToDevice,
29.     allDevices,
30.     disconnectFromDevice,
31.     writeToCharacteristic,
32.   } = useReactBLE(); // Change hook to useReactBLE
33.
34.   const setDevice = (device: any | null) => { // Update the function type
35.     if (device) {
36.       setConnectedDevice(device.id); // Assuming device.id is the identifier
37.     } else {
38.       setConnectedDevice(null);
39.     }
40.   };
```
41. const MICROCONTROLLER_SERVICE_UUID = '49535343-fe7d-4ae5-8fa9-9fafd205e455'; // Update service UUID
42. const MICROCONTROLLER_CHARACTERISTIC_RECEIVE_UUID = '49535343-1e4d-4bd9-ba61-23c647249616'; // Update characteristic UUID
43. const MICROCONTROLLER_CHARACTERISTIC_TRANSMIT_UUID = '49535343-1E4D-4BD9-BA61-23C647249616'; // Update characteristic UUID
44. const handlePress = (expressionChoice: number) => {
45.   let choice: string = "00";
46.   let message = '';
47. 
48.   if (expressionChoice === 1) {
49.     message = 'Sending Happy Expression to the microcontroller.';
50.     choice = "!";
51.   } else if (expressionChoice === 2) {
52.     message = 'Sending Sad Expression to the microcontroller.';
53.     choice = "?";
54.   } else if (expressionChoice === 3) {
55.     message = 'Sending Excited Expression to the microcontroller.';
56.     choice = "#";
57.   } else if (expressionChoice === 4) {
58.     message = 'Sending Force Expression to the microcontroller.';
59.     choice = "+";
60.   }
61.   Alert.alert(message);
62. 
63.   if (connectedDevice) {
64.     writeToCharacteristic(connectedDevice, MICROCONTROLLER_SERVICE_UUID,
65.       MICROCONTROLLER_CHARACTERISTIC_TRANSMIT_UUID, choice)
66.       .then(() => {
67.         Alert.alert('Expression sent to Grogu.');
68.       })
69.       .catch((error) => {
70.         console.log('Failed to send expression to Grogu: ', error);
71.       });
72.   } else {
73.     Alert.alert('Please connect to Grogu first.');
74.   }
75. }
76. };
77. }
78. return (<View style={styles.container}>
Code for the CustomButton Component:

```javascript
import React from 'react';
import { Text, TouchableOpacity } from 'react-native';
```
const CustomButton = ({ title, style, onPress }) => {
  return (<TouchableOpacity style={style} onPress={onPress} hitSlop={{ top: 120, bottom: 120, left: 60, right: 60 }}> <Text style={{ color: 'black', fontWeight: 'bold', fontSize: 35, fontFamily: 'Starjedi' }}>{title}</Text> </TouchableOpacity>);
}

export default CustomButton;

useBLE.tsx for Bluetooth functions:

import {useState} from 'react';
import {PermissionsAndroid, Platform} from 'react-native';
import BleManager from 'react-native-ble-manager';
import {PERMISSIONS, request} from 'react-native-permissions';
import DeviceInfo from 'react-native-device-info';

// Define your characteristic UUIDs
const MICROCONTROLLER_CHARACTERISTIC_RECEIVE_UUID = '49535343-8841-43F4-A8D4-ECBE34729BB3';
const MICROCONTROLLER_CHARACTERISTIC_TRANSMIT_UUID = '49535343-1E4D-4BD9-BA61-23C647249616';

// Initialize the BleManager
BleManager.start({showAlert: false});

type VoidCallback = (result: boolean) => void;

interface BluetoothLowEnergyApi {
  requestPermissions(cb: VoidCallback): Promise<void>;
  scanForPeripherals(): void;
  connectToDevice: (deviceId: string) => Promise<void>;
  disconnectFromDevice: (deviceId: string) => void;
  connectedDevice: string | null;
  allDevices: string[];
  writeToCharacteristic: (deviceId: string, serviceUUID: string, characteristicUUID: string, data: string) => Promise<void>;
}
function useBLE(): BluetoothLowEnergyApi {
    const [allDevices, setAllDevices] = useState<string[]>([]);
    const [connectedDevice, setConnectedDevice] = useState<string | null>(null);

    const requestPermissions = async (cb: VoidCallback) => {
        if (Platform.OS === 'android') {
            const result = await request(PERMISSIONS_ANDROID.ACCESS_FINE_LOCATION);
            cb(result === 'granted');
        } else {
            cb(true);
        }
    };

    const scanForPeripherals = async (): Promise<void> => {
        try {
            const results = await BleManager.scan([], 1, true);
            console.log('Scanning for peripherals:', results);
            if (Array.isArray(results)) {
                results.forEach((result: any) => {
                    const {id, name}: {id: string, name: string} = result;
                    setAllDevices((prevDevices: string[]) => {
                        if (!prevDevices.includes(id)) {
                            return [...prevDevices, id];
                        }
                    });
                });
            }
        } catch (error) {
            console.log('Scan error:', error);
        }
    };

    const connectToDevice = async (deviceId: string) => {
        try {
            await BleManager.connect(deviceId);
            setConnectedDevice(deviceId);
            console.log('Connected to device:', deviceId);
        } catch (error) {
            console.log('Connect error:', error);
        }
    };
};
const writeToCharacteristic = async (deviceId: string, serviceUUID: string, characteristicUUID: string, data: string) => {
    const buffer = Buffer.from(data);
    const sentData = buffer.toJSONString().data;
    try {
        await BleManager.write(deviceId, serviceUUID, characteristicUUID, sentData);
        console.log('Data sent to characteristic:', sentData);
    } catch (error) {
        console.log('Write error:', error);
    }
};

const disconnectFromDevice = (deviceId: string) => {
    try {
        BleManager.disconnect(deviceId)
            .then(() => {
                console.log('Disconnected from device:', deviceId);
            })
            .catch(error => {
                console.log('Disconnect error:', error);
            });
    } catch (error) {
        console.log('Disconnect error:', error);
    }
};

return {
    scanForPeripherals,
    requestPermissions,
    connectToDevice,
    allDevices,
    connectedDevice,
    disconnectFromDevice,
    writeToCharacteristic,
};

export default useBLE;
import React, { useEffect, useState } from 'react';
import { Button, Modal, SafeAreaView, StyleSheet, Text, TouchableOpacity, View } from 'react-native';
import BleManager from 'react-native-ble-manager'; // Import BleManager
import { Buffer } from 'buffer'; // Import Buffer for string conversion

interface Device {
  id: string;
  name: string;
  // Add any other properties of the device object if needed
}

interface BluetoothScreenProps {
  isVisible: boolean;
  openModal: () => void;
  closeModal: () => void;
  setDevice: (device: Device | null) => void;
  disconnectDevice: () => void;
  connectedDevice: string | null;
  connectToDevice: (deviceId: string) => void; // Add connectToDevice prop
}

const BluetoothScreen: React.FC<BluetoothScreenProps> = ({ isVisible, openModal, closeModal, setDevice, disconnectDevice, connectedDevice, connectToDevice }) => {
  const [scannedDevices, setScannedDevices] = useState<Set<string>>(new Set());
  // Store scanned device IDs
  const [allDevices, setAllDevices] = useState<Device[]>([]); // Displayed devices

  useEffect(() => {
    scanForDevices();
  }, []);

  const scanForDevices = () => {
    BleManager.enableBluetooth().then(() => {
      console.log('Bluetooth is enabled, starting scan...');
      BleManager.scan([], 0, true).then(results => {
        console.log('Scanning...');
      }).catch(error => {
        console.log('Error scanning:', error);
      });
    });
  };

  // Return the component
};

const BluetoothScreenComponent: React.FC<BluetoothScreenProps> = ({ isVisible, openModal, closeModal, setDevice, disconnectDevice, connectedDevice, connectToDevice }) => {
  return <BluetoothScreen isVisible={isVisible} openModal={openModal} closeModal={closeModal} setDevice={setDevice} disconnectDevice={disconnectDevice} connectedDevice={connectedDevice} connectToDevice={connectToDevice} />
};

export default BluetoothScreenComponent;
```javascript
.catch(error => {
    console.log('Bluetooth is not enabled:', error);
});

BleManager.addListener('BleManagerDiscoverPeripheral', device => {
    // console.log('Discovered device:', device);
    if (device.name && device.name.startsWith('RN4870') &&
    !scannedDevices.has(device.id)) {
        setScannedDevices(prevDevices => new Set([...prevDevices, device.id]));
        // Store scanned device ID
        setAllDevices(prevDevices => [...prevDevices, device]); // Add device
        object to displayed devices
    }
});

return (  
    <Modal  
        animationType="slide"  
        transparent={true}  
        visible={isVisible}  
        onRequestClose={closeModal}  
    >  
        <SafeAreaView style={styles.container}>  
            <View style={styles.heartRateTitleWrapper}>  
                {connectedDevice ? (  
                    <>  
                        <Text style={styles.heartRateTitleText}>Grogu Activated</Text>  
                    </>  
                ) : (  
                    <Text style={styles.heartRateTitleText}>Please Connect to Grogu</Text>  
                )}  
            </View>  
            <Button title="Close" onPress={closeModal} />  
        </SafeAreaView>  
    </Modal>
```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#f2f2f2',
  },
  heartRateTitleWrapper: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  heartRateTitleText: {
    fontSize: 30,
    fontWeight: 'bold',
    textAlign: 'center',
    marginHorizontal: 20,
    color: 'black',
  },
  heartRateText: {
    fontSize: 25,
    marginTop: 15,
    textAlign: 'center',
    color: 'black',
  },
  ctaButton: {
    backgroundColor: 'purple',
    justifyContent: 'center',
    alignItems: 'center',
    height: 50,
    marginHorizontal: 20,
    marginVertical: 5,
    borderBottom: 8,
  },
  ctaButtonText: {
    fontSize: 18,
    fontWeight: 'bold',
    color: 'black',
  },
});
import React, { FC, useCallback } from 'react';
import { FlatList, ListRenderItemInfo, Modal, SafeAreaView, Text, StyleSheet, TouchableOpacity } from 'react-native';
import BleManager from 'react-native-ble-manager'; // Import BleManager
import { Buffer } from 'buffer'; // Import Buffer for string conversion

type DeviceModalListItemProps = {
  item: string; // Change type from ListRenderItemInfo<Device> to string
  connectToPeripheral: (deviceId: string) => void; // Adjust connectToPeripheral parameter
  closeModal: () => void;
};

type DeviceModalProps = {
  devices: string[]; // Change type from Device[] to string[]
  visible: boolean;
  connectToPeripheral: (deviceId: string) => void; // Adjust connectToPeripheral parameter
  closeModal: () => void;
};

const DeviceModalListItem: FC<DeviceModalListItemProps> = props => {
  const { item, connectToPeripheral, closeModal } = props;

  const connectAndCloseModal = useCallback(() => {
    connectToPeripheral(item);
    closeModal();
  }, [closeModal, connectToPeripheral, item]);

  return (
    <TouchableOpacity onPress={connectAndCloseModal}
      style={{ modalStyle.ctaButton }}
      style={modalStyle.ctaButtonText}>{item}</Text>
    </TouchableOpacity>
  )
};
const DeviceModal: FC<DeviceModalProps> = props => {
    const { devices, visible, connectToPeripheral, closeModal } = props;

    const renderDeviceModalListItem = useCallback((
        item: ListRenderItemInfo<string>) => {
        return (<DeviceModalListItem
            item={item.item}
            connectToPeripheral={connectToPeripheral}
            closeModal={closeModal}
        />);
    }, [closeModal, connectToPeripheral],);

    return (<Modal
        style={modalStyle.modalContainer}
        animationType="slide"
        transparent={false}
        visible={visible}>
        <SafeAreaView style={modalStyle.modalTitleStyle}>
            <Text style={modalStyle.modalTitleText}>Tap on a device to connect</Text>
            <FlatList
                contentContainerStyle={modalStyle.modalFlatlistContinerStyle}
                data={devices}
                renderItem={renderDeviceModalListItem}
            />
        </SafeAreaView>
    </Modal>);
};

const modalStyle = StyleSheet.create({
    modalContainer: {
        flex: 1,
        backgroundColor: '#f2f2f2',
    },
    modalFlatlistContiner: {
        flex: 1,
        justifyContent: 'center',
    },
});
modalCellOutline: {
    borderWidth: 1,
    borderColor: 'black',
    alignItems: 'center',
    marginHorizontal: 20,
    paddingVertical: 15,
    borderRadius: 8,
},
modalTitle: {
    flex: 1,
    backgroundColor: '#f2f2f2',
    color: 'black'
},
modalTitleText: {
    marginTop: 40,
    fontSize: 30,
    fontWeight: 'bold',
    marginHorizontal: 20,
    textAlign: 'center',
    color: 'black'
},
ctaButton: {
    backgroundColor: 'purple',
    justifyContent: 'center',
    alignItems: 'center',
    height: 50,
    marginHorizontal: 20,
    marginVertical: 5,
    borderRadius: 8,
},
ctaButtonText: {
    fontSize: 18,
    fontWeight: 'bold',
    color: 'black',
},
});

export default DeviceModal;
115.1.1. Send integer over Bluetooth CR
Bluetooth communication is needed so that the user can communicate with the animatronic. To do this a phone with the app on it will need to communicate via Bluetooth from the microcontroller (PIC24), this must be done through Uart via the MIKROE RN4870 Bluetooth module. For testing purposes, what was received from the phone was also printed on the LCD screen that is on the development board (the LCD will not be in the final design of the animatronic). During testing any characters could be sent from the phone to the Bluetooth module but we will only need to send integers from 1-4. As proof the Bluetooth communication is possible the user typed ‘hello world’ on their phone and was then displayed onto the LCD (Figure 14).
# Definitions of CTS and RTS pins
#define CTS _RD6
#define RTS _RF12

// Function to initialize UART2 with a specified baud rate
void initU2(int BRG)
{
    // Set the baud rate generator value for UART2
    U2BRG = BRG;
    // Configure UART2 mode: 8-bit data, no parity, one stop bit, high baud rate
    U2MODE = 0x8008;
    U2STA = 0x0400; // Enable the transmission for UART2
    TRISFbits.TRISF12 = 0; // Set the TRIS register for RTS pin as input
    RTS = 1; // Set the RTS pin high
}
// Function to receive a character from UART2
char getU2(void)
{
    RTS = 0; // Set the RTS pin low

    // Wait until the UART2 receive buffer has data available
    while (!U2STAbits.URXDA);
    RTS = 1; // Set the RTS pin high

    return U2RXREG; // Return the received character
}

// main function
int main(void)
{
    SYSTEM_Initialize(); // Initialize the microcontroller system

    initU2(34); // Initialize UART2 with a baud rate of 115000
    TRISA = 0xff00;
    T1CON = 0x8030; // Configure Timer1

    TRISA = 0xFF00; // Set TRISA register for specified pins
    TRISFbits.TRISF2 = 0; // Set TRIS register for specified pin
    TRISDbits.TRISD7 = 1; // Set TRIS register for specified pin

    // Variable to store the received character from Bluetooth
    char receivedChar;

    while (1)
    {
        _RA0 = 1;

        // Receive a character from Bluetooth via UART2
        receivedChar = getU2();

        if (receivedChar == 1)
        {
            // Send Happy PWM signal
        }
    }
}
else if(receivedChar == 2) {
    //Send Sad PWM signal
}
else if(receivedChar == 3) {
    //Send Excitement PWM signal
}
else if(receivedChar == 4) {
    //Send Force PWM signal
}
_RA0 = 0;

return 1;

115.1.2. Control servo motors DM

Put PCA9685 into sleep mode

Set pre-scaler

Put PCA9685 into wake mode

Select which PWM channel

Send values for the four corresponding registers

Servo moves based off signal generated

Figure 18: Steps to move a servo
Figure 19: Description of which channels control which body parts as well as the servos default position

As described in section 2.3.2 and shown in figure 18 there are a certain number of steps to move a servo. The PCA9685 communicates with the microcontroller via I2C. A basic I2C communication frame begins with a start condition then the devices address in this case the device’s address is 0x80. The device address is followed by the data you would like to send to
the device. The I2C communication is stopped when a stop condition is called and the I2C frame is ended.

Step 1 to move a servo is to put the PCA9685 into sleep mode. This is done using the preScaler function in section 115.1.2.1 (The start of all functions are highlighted in yellow). The code starts with a start condition followed by the devices address 0x80. Next the microcontroller needs to know where on the device to send the next byte of data. The microcontroller is told to send the data to the mode 1 register denoted by the line that sends 0x00 to the PCA9685. The byte that is sent to the mode 1 register is 0x30. This will put the device into sleep mode as well as auto increment mode. Sleep mode has been discussed. However, auto increment allows the device to move to the next register without the microcontroller needing to tell the device what register to put the following byte that the device is receiving. This is done right after the byte 0x30 is sent to mode 1. Mode 2 is the next register in line after mode 1. So, the next byte that is sent is 0x4 and that value is stored into the mode 2 register. The order that the registers are in is in the PCA9685 datasheet. The byte 0x4 tells the device to configure the PWM outputs to totem pole structure. Then a stop condition is sent ending the communication and ending step 1.
Step 2 in the order to move a servo is also done is the perScaler function. To set the pre-scaler a start condition is sent followed by the device address. The microcontroller then points to the pre-scaler register using the byte 0xFE. The data that is stored in the register is 0x7D. This value will allow the PCA9685 to generate a period larger than 20ms. Note that the pre-scaler does not set the period of the PWM signal that is done by the four registers that correspond to a given PWM channel. A stop condition is sent ending I2C communication and ending step 2.

Step 3 is to put the device into wake mode. This step is done in the setup function in section 115.1.2.1. To put the device into wake mode a start condition is sent followed by the device’s address. The microcontroller points to mode 1 register with byte 0x00. The data that is stored into the mode 1 register is 0x20. This puts the device into wake mode as well as auto increment. Mode 2 is also set to the same value of 0x4 as in the preScaler function. A stop condition is called ending step 3.

Step 4 is to select a PWM channel that a servo is connected to. This is done with the Servo function. The whole point of this function is to cut down on the amount of code needed. Since picking the channel and setting the four registers to build the PWM
wave are the same every time. The servo function takes a channel and a degree. The 16 channel values are hard coded into this function. The value for each channel is in the PCA9685’s datasheet. For example, if you want to move the servo connected to channel 0. Following a start condition and the device address the byte 0x06 is sent.

Step 5 is to set the four registers OFF_H, OFF_L, ON_H, and ON_L. The value for ON_H is set to 0x0 and ON_L is set to 0xCD. The value is set to these values to set the period of the PWM wave to 20ms. These two values are the same for every PWM channel so they are hard coded and can’t be changed. The values for OFF_H and OFF_L are able to be changed based on the angle that you want the servo to move to. All the angles that can be sent to a servo are in the switch statement with the degree variable. The basic idea is that the values OFF_H and OFF_L control the length of the pulse of the PWM. As shown in figure 3 the length of the pulse is 2ms. The 2ms pulse corresponds to 145-degree angle. 145 degrees sets OFF_L to 0x7E and OFF_H to 0x2. So to set for example the servo connected to channel 0 to 145 degrees the line Servo(0,145); would be called. This will send a start condition followed by the device’s address. For channel 0 the ch variable is set to 0x06 followed by bytes 0xCD and 0x0 which sets the PWM
period for the servo connected to channel 0. Then OFF_L is set to 0x7E and OFF_H is set to 0x2. The stop condition is then sent ending step 5 and moving the angle of the servo connected to channel 0 to 145 degrees.

It should be noted that the servos don’t move based off the last position but to a set position. Meaning that each degree from 0-180 has a set position. So, if a servo is set to 90 degrees and you want to move the servo 180 degrees. If you call that servo to move to 180 degrees, the servo will only move 90 degrees not 180 degrees. Since there is only a 90-degree difference between 90 and 180.

Similarly, if the servo is at 90 degrees and you call that servo to move 90 degrees the servo will not move as it is already at 90 degrees. If you wanted that servo to move 90 degrees, you would need to call 180 or 0 degrees. Another example is if the servo is at 0 degrees, and you call the servo to rotate 180 degrees it will rotate 180 as there is a 180-degree difference between 0 and 180.

It should also be noted that the design uses two PCA9685 chips as previously described. That means there are two I2C communication channels going at the same time, one to the PCA9685 connected to the small MG90D servos and one to the PCA9685 connected to the big MG995 servos. This also means
that in the code in section 115.1.2.1 the PCA9685 connected to the big servos is denoted with a 2 at the end of a function name. For example, in the main part of the program preScaler is called and preScaler2 is then called. This is to keep the two communications separate because the data being sent each PCA9685 is different. The data is different because the degree angles of the MG995 servos correspond to different values of the MG90D servos.

In the while loop of the main program after the Bluetooth receiver receives the value being sent. The program then chooses based on that selection to perform one of the four movements. There is also a default function. This default function sets the resting point for all the servos. The default function is called at the start and the end of each of the four main functions Happy, Sad, Force, and Excited. This is because the animatronic needs to have the same movements every time. The animatronic would look weird if it moved from the end of the sad expression right into the force expression. This also handles if the user hits a button when the animatronic is in the middle of another expression. If a button is hit when the animatronic is in the middle of an expression the animatronic will finish the movement and will go to the default positions, then the animatronic performs the movement of the button that was pushed.
115.1.2.1. Code to perform four preset movements via Bluetooth DM

```c
#include "mcc_generated_files/system.h"
#include <stdio.h>
#define DELAY 2000000

//I2C
void initI2C(int BRG)
{
    I2C1BRG = BRG;
    I2C1CONbits.I2CEN = 1;
}

void startI2C(void)
{
    TMR3=0;while(TMR3<160);
    I2C1CONbits.SEN = 1;
    while(I2C1CONbits.SEN);
    TMR3=0;while(TMR3<160);
}

void stopI2C(void)
{
    TMR3=0;while(TMR3<160);
    I2C1CONbits.PEN = 1;
    while(I2C1CONbits.PEN);
    TMR3=0;while(TMR3<160);
}

void sendbyteI2C(char data)
{
    while(I2C1STATbits.TBF);
    I2C1TRN = data;
    TMR3=0;while(TMR3<160);
}

char getbyteI2C(void)
{
    I2C1CONbits.RCEN = 1;
    while(!I2C1STATbits.RBF);
    I2C1CONbits.ACKEN = 1;
    TMR3=0;while(TMR3<160);
    return(I2C1RCV);
}

//End I2C
```

59
void preScaler(void){
    startI2C();
    sendbyteI2C(0x80); // PCA9685's address
    while(I2C1STATbits.ACKSTAT);
    TMR3=0;while(TMR3<3000);
    sendbyteI2C(0x00); // Set pointer to Mode register 1
    while(I2C1STATbits.ACKSTAT);
    TMR3=0;while(TMR3<1500);
    sendbyteI2C(0x30); // Mode register 1 - settings int clock,
    auto increment 30 sleep
    while(I2C1STATbits.ACKSTAT);
    TMR3=0;while(TMR3<5000);
    sendbyteI2C(0x04); // Mode register 2 (because auto inc
    happening the address points to Mode Reg 2)
    while(I2C1STATbits.ACKSTAT);
    TMR3=0;while(TMR3<1500);
    stopI2C();
    startI2C();
    sendbyteI2C(0x80); // PCA9685's address
    while(I2C1STATbits.ACKSTAT);
    TMR3=0;while(TMR3<3000);
    //on
    sendbyteI2C(0xFE); //pre scaler
    //01000010
    while(I2C1STATbits.ACKSTAT);
    TMR3=0;while(TMR3<1500);
    sendbyteI2C(0x7D); //data for pre scaler
    //10011001
    while(I2C1STATbits.ACKSTAT);
    TMR3=0;while(TMR3<1500);
    stopI2C();
}

//Sets PCA in wake mode
```c
void setup(void){
  startI2C();
  sendbyteI2C(0x80); // PCA9685's address
  while(I2C1STATbits.ACKSTAT);
  TMR3=0;while(TMR3<3000);

  sendbyteI2C(0x00); // Set pointer to Mode register 1
  while(I2C1STATbits.ACKSTAT);
  TMR3=0;while(TMR3<1500);

  sendbyteI2C(0x20); // Mode register 1 - settings int clock, auto increment
  while(I2C1STATbits.ACKSTAT);
  TMR3=0;while(TMR3<1500);

  sendbyteI2C(0x04); // Mode register 2 (because auto inc happening the address points to Mode Reg 2)
  while(I2C1STATbits.ACKSTAT);
  TMR3=0;while(TMR3<1500);

  stopI2C();
}

/********************************************
//I2C settings for second PCA
********************************************

void initI2C2(int BRG)
{
  //T3CON=0x8000;
  I2C2BRG = BRG;
  I2C2CONbits.I2CEN = 1;
}

void startI2C2(void)
{
  TMR1=0;while(TMR1<160);
  I2C2CONbits.SEN = 1;
  while(I2C2CONbits.SEN);
  TMR1=0;while(TMR1<160);
}

void stopI2C2(void)
{
  TMR1=0;while(TMR1<160);
  I2C2CONbits.PEN = 1;
}
```
while(I2C2CONbits.PEN);
    TMR1=0;while(TMR1<160);
}

void sendbyteI2C2(char data) {
    while(I2C2STATbits.TBF);
    I2C2TRN = data;
    TMR1=0;while(TMR1<160);
}

char getbyteI2C2(void) {
    I2C2CONbits.RCEN = 1;
    while(!I2C2STATbits.RBF);
    I2C2CONbits.ACKEN = 1;
    TMR1=0;while(TMR1<160);
    return(I2C2RCV);
}

//End I2C

////////////////////////////////////////////////////////////////////////////////////////
//pre scaler address FE data 7D
//Sets the period of the PWMs to 20ms
//void preScaler2(void){
    startI2C2();
    sendbyteI2C2(0x80); // PCA9685's address
    while(I2C2STATbits.ACKSTAT);
    TMR1=0;while(TMR1<3000);

    sendbyteI2C2(0x00); // Set pointer to Mode register 1
    while(I2C2STATbits.ACKSTAT);
    TMR1=0;while(TMR1<1500);

    sendbyteI2C2(0x30); // Mode register 1 - settings int clock, auto increment 30 sleep
    while(I2C2STATbits.ACKSTAT);
    TMR1=0;while(TMR1<5000);

    sendbyteI2C2(0x04); // Mode register 2 (because auto inc happening the address points to Mode Reg 2)
    while(I2C2STATbits.ACKSTAT);
    TMR1=0;while(TMR1<1500);

    stopI2C2();
startI2C2();

sendbyteI2C2(0x80); // PCA9685's address
while(I2C2STATbits.ACKSTAT);
TMR1=0;while(TMR1<3000);

//on
sendbyteI2C2(0xFE); // pre scaler
//01000010
while(I2C2STATbits.ACKSTAT);
TMR1=0;while(TMR1<1500);

sendbyteI2C2(0x7D); // data for pre scaler
//10011001
while(I2C2STATbits.ACKSTAT);
TMR1=0;while(TMR1<1500);

stopI2C2();

} //Sets PCA in wake mode

void setup2(void){
    startI2C2();
    sendbyteI2C2(0x80); // PCA9685's address
    while(I2C2STATbits.ACKSTAT);
    TMR1=0;while(TMR1<3000);

    sendbyteI2C2(0x00); // Set pointer to Mode register 1
    while(I2C2STATbits.ACKSTAT);
    TMR1=0;while(TMR1<1500);

    sendbyteI2C2(0x20); // Mode register 1 - settings int clock, auto increment
    while(I2C2STATbits.ACKSTAT);
    TMR1=0;while(TMR1<1500);

    sendbyteI2C2(0x04); // Mode register 2 (because auto inc happening the address points to Mode Reg 2)
    while(I2C2STATbits.ACKSTAT);
    TMR1=0;while(TMR1<1500);

    stopI2C2();
}

//For the large servos
void Servo2(int channel, int deg)
{
    int off_L;
    int off_H;
    int ch;

    switch(channel)
    {
    case 0:
        ch = 0x06;
        break;
    case 1:
        ch = 0x0A;
        break;
    case 2:
        ch = 0x0E;
        break;
    case 3:
        ch = 0x12;
        break;
    case 4:
        ch = 0x16;
        break;
    case 5:
        ch = 0x1A;
        break;
    case 6:
        ch = 0x1E;
        break;
    case 7:
        ch = 0x22;
        break;
    case 8:
        ch = 0x26;
        break;
    case 9:
        ch = 0x2A;
        break;
    case 10:
        ch = 0x2E;
        break;
    case 11:
        ch = 0x32;
        break;
    case 12:
        if (deg >= 78 && deg <=100){

ch = 0x36;
break;
}
else{
  break;
}
break;
case 13:
  if (deg >= 80 && deg <=160){
    ch = 0x3A;
    break;
  }
  else{
    break;
  }
break;
case 14:
  ch = 0x3E;
  break;
case 15:
  ch = 0x42;
  break;
default: break;
}//End channel switch

//Big servo degree
if (0 <= channel <= 6){
  switch(deg){
    case 0:
      off_L = 0x14;
      off_H = 0x1;
      break;
    case 45:
      off_L = 0x87;
      off_H = 0x1;
      break;
    case 55:
      off_L = 0xA5;
      off_H = 0x1;
      break;
    case 70:
      off_L = 0xD7;
      off_H = 0x1;
      break;
  }
}
case 80:
    off_L = 0xF3;
    off_H = 0x1;
    break;

case 90:
    off_L = 0xFB;
    off_H = 0x1;
    break;

case 98:
    off_L = 0x00;
    off_H = 0x2;
    break;

case 100:
    off_L = 0x1A;
    off_H = 0x2;
    break;

case 105:
    off_L = 0x32;
    off_H = 0x2;
    break;

case 108:
    off_L = 0x41;
    off_H = 0x2;
    break;

case 133:
    off_L = 0x64;
    off_H = 0x2;
    break;

case 135:
    off_L = 0x6E;
    off_H = 0x2;
    break;

case 145:
    off_L = 0x7E;
    off_H = 0x2;
    break;
case 180:
    off_L = 0xDC;
    off_H = 0x2;
    break;

case 0015:
    off_L = 0x7E;
    off_H = 0x2;
    break;

default: break;
} //End deg switch

//Inputs the values and generates the PWM for that channel and degree
startI2C2();

sendbyteI2C2(0x80); // PCA9685's address
while(I2C2STATbits.ACKSTAT);
TMR1=0;while(TMR1<3000);

//on
sendbyteI2C2(ch); //set pointer to channel (on_L)
while(I2C2STATbits.ACKSTAT);
TMR1=0;while(TMR1<1500);

sendbyteI2C2(0xCD); //channel on_L data increase duty
while(I2C2STATbits.ACKSTAT);
TMR1=0;while(TMR1<1500);

sendbyteI2C2(0x0); //channel on_H data (auto increment is set so no need to change address manually) decrease duty with value
while(I2C2STATbits.ACKSTAT);
TMR1=0;while(TMR1<1500);

//off
sendbyteI2C2(off_L); //channel off_L data
while(I2C2STATbits.ACKSTAT);
TMR1=0;while(TMR1<1500);

sendbyteI2C2(off_H); //channel off_H data
while(I2C2STATbits.ACKSTAT);
TMR1=0;while(TMR1<1500);
stopI2C2();
} //End move servo

////////////////////////////////////////////////
////////////////////////////////////////////////

//receive the channel you want and the degree you want the servo to move

void Servo(int channel, int deg)
{
    int off_L;
    int off_H;
    int ch;

    switch(channel)
    {
        case 0:
            ch = 0x06;
            break;
        case 1:
            ch = 0x0A;
            break;
        case 2:
            ch = 0x0E;
            break;
        case 3:
            ch = 0x12;
            break;
        case 4:
            ch = 0x16;
            break;
        case 5:
            ch = 0x1A;
            break;
        case 6:
            ch = 0x1E;
            break;
        case 7:
            ch = 0x22;
            break;
        case 8:
            ch = 0x26;
            break;
        case 9:
            ch = 0x2A;
            break;
    }
case 10:
  ch = 0x2E;
  break;

case 11:
  ch = 0x32;
  break;

case 12:
  ch = 0x36;
  break;

case 13:
  ch = 0x3A;
  break;

case 14:
  ch = 0x3E;
  break;

case 15:
  ch = 0x42;
  break;

default: break;
} //End channel switch

//Little servo degree
if(5 <= channel <= 15){
  switch(deg){
    case 0:
      off_L = 0x22;
      off_H = 0x1;
      break;

    case 25:
      off_L = 0x6E;
      off_H = 0x1;
      break;

    case 45:
      off_L = 0xB7;
      off_H = 0x1;
      break;

    case 72:
      off_L = 0x85;
      off_H = 0x1;
      break;
  } //End switch
} //End if
case 75:
    off_L = 0xA5;
    off_H = 0x1;
    break;

case 755:
    off_L = 0xDB;
    off_H = 0x1;
    break;

case 76:
    off_L = 0xE5;
    off_H = 0x1;
    break;

case 78:
    off_L = 0xF6;
    off_H = 0x1;
    break;

case 80:
    off_L = 0xFE;
    off_H = 0x1;
    break;

case 83:
    off_L = 0x10;
    off_H = 0x2;
    break;

case 86:
    off_L = 0x16;
    off_H = 0x2;
    break;

case 90:
    off_L = 0x25;
    off_H = 0x2;
    break;

case 95:
    off_L = 0x35;
    off_H = 0x2;
    break;

case 100:
off_L = 0x45;
off_H = 0x2;
break;

case 124:
    off_L = 0x52;
    off_H = 0x2;
    break;

case 125:
    off_L = 0x5A;
    off_H = 0x2;
    break;

case 130:
    off_L = 0x5F;
    off_H = 0x2;
    break;

case 160:
    off_L = 0x75;
    off_H = 0x2;
    break;

case 180:
    off_L = 0xD6;
    off_H = 0x2;
    break;

case 0015:
    off_L = 0x4E;
    off_H = 0x1;
    break;

default: break;
} //End deg switch

//Inputs the values and generates the PWM for that channel and degree
startI2C();

sendbyteI2C(0x80); // PCA9685's address
while(I2C1STATbits.ACKSTAT);
TMR3=0;while(TMR3<3000);
//on
sendbyteI2C(ch); //set pointer to channel (on_L)
while(I2C1STATbits.ACKSTAT);
TMR3=0;while(TMR3<1500);

sendbyteI2C(0xCD); //channel on_L data increase duty
while(I2C1STATbits.ACKSTAT);
TMR3=0;while(TMR3<1500);

sendbyteI2C(0x0); //channel on_H data (auto increment is set
so no need to change address manually) decrease duty with value
while(I2C1STATbits.ACKSTAT);
TMR3=0;while(TMR3<1500);

//off
sendbyteI2C(off_L); //channel off_L data
while(I2C1STATbits.ACKSTAT);
TMR3=0;while(TMR3<1500);

sendbyteI2C(off_H); //channel off_H data
while(I2C1STATbits.ACKSTAT);
TMR3=0;while(TMR3<1500);

stopI2C();
}//End move servo

void Default(void){
//Default DON'T TOUCH

////////////////////////////////
//Small servo PCA
////////////////////////////////

    //Left eye ball
    Servo(15,90);
    Servo(14,90);

    //Left eye lids
    Servo(13,80);
    Servo(12,95);

    //Right eye ball
    Servo(11,95);
    Servo(10,95);

    //Right lid
    Servo(9,90);
    Servo(8,90);
//right ear hor
Servo(7,90);
//left ear hor
Servo(6,90);
//mouth
Servo(5,76);

////////////////////////////////////
//Big servo PCA
////////////////////////////////////

Servo2(6,90);
Servo2(5,100);

//Left ear ver
Servo2(4,90);

//Right ear ver
Servo2(3,90);
//Neck up/down Left
Servo2(2,100);
//Neck up/down Right
Servo2(1,108);
//Neck hor
Servo2(0,105);

    //for(k = 0;k<DELAY;k++) {
    //Nop();
    //}

void Excited(void){
    Default();

    for(long k = 0;k<(DELAY-1750000);k++) {
        Nop();
    }

    //Blink
    //Left lid
    Servo(13,160);
    Servo(9,76);//right lid top
    Servo(12,80);

    //Right lid bottom
Servo(8, 160);

for (long k = 0; k < (DELAY - 1750000); k++) {
    Nop();
}

// Moves to the left

// Small servo PCA

// Values for SMALL SERVO degree as of 3/19/24
// 0, 25, 45 wrong, 72, 75, 76, 78, 80, 83, 86, 90, 95, 100, 125, 160, 180
// ball
Servo(15, 0);
// lid
Servo(13, 80);
Servo(12, 100);
// ball
Servo(11, 86);
// Servo(10, 100);
// lid
Servo(9, 90);
Servo(8, 83);

 toda

// Big servo PCA

// Values for BIG SERVO degree as of 3/19/24
// test case angle 0015
// 0, 45, 70, 90, 98, 100, 105, 133, 135, 180
// Left ear ver
Servo2(4, 70);

// Right ear ver
Servo2(3, 105);
/Neck up/down Left
Servo2(2,98);

//Neck up/down Right
Servo2(1,105);

//Neck hor
//Servo2(0,98);
Servo2(5,180);
Servo2(6,135);

/////////////////////////////////////////
//Delay Block
for(long k = 0;k<DELAY;k++ ) {
    Nop();
}

/////////////////////////////////////////
//Moves to the right

/////////////////////////////////////////
//Small servo PCA

///////////////////////////////
//Values for SMALL SERVO degree as of 3/19/24
//0, 25,45 wrong,72, 75,76, 78, 80,86, 90, 95, 100,125,160, 180

//Blink
//Left lid
Servo(13,160);
Servo(9,76);//right lid top
Servo(12,80);

//Right lid bottom
Servo(8,160);

Servo(5,76);

for(long k = 0;k<(DELAY-1750000);k++ ) {
    Nop();
}

//lid
Servo(13,80);
Servo(12,100);

//lid
Servo(9,90);
Servo(8,83);
Servo(5,86);

//Big servo PCA

//Values for BIG SERVO degree as of 3/19/24
//test case angle 0015
//0, 45, 70, 90, 98, 100, 105, 135, 180
//Neck hor
//Servo2(0,135);

Servo2(5,45);
Servo2(6,45);

for(long k = 0;k<DELAY;k++) {
    Nop();
}

//Blink
//Left lid
Servo13,160);
Servo(9,76);//right lid top
Servo(12,80);

//Right lid bottom
Servo(8,160);
Servo(5,76);

for(long k = 0;k<(DELAY-1750000);k++) {
    Nop();
}

//lid
Servo(13,80);
Servo(12,100);

//lid
Servo(9,90);
Servo(8,83);

Servo(5,86);

for(long k = 0;k<(DELAY-1750000);k++) {
    Nop();
}

Servo2(5,145);
for(long k = 0; k < DELAY; k++) {
    Nop();
}
Servo(13, 160);
Servo(9, 76); // right lid top
Servo(12, 80);

// Right lid bottom
Servo(8, 160);
Servo(5, 76);

for(long k = 0; k < (DELAY - 1750000); k++) {
    Nop();
}
Default();

} //Force

void Force(void){
    Default();
    for(long k = 0; k < (DELAY - 1750000); k++) {
        Nop();
    }

    // Values for SMALL SERVO degree as of 3/19/24
    // 0, 25, 45 wrong, 72, 75, 75, 76, 78, 80, 83, 86, 90, 95, 100, 124, 125, 130, 160, 180
    // Left eye ball
    Servo(15, 78);
    Servo(14, 72);

    // Left eye lids
    Servo(13, 130);
    Servo(12, 90);

    // Right eye ball
    Servo(11, 95);
    Servo(10, 124);

    // Right lid
    Servo(9, 76);
    Servo(8, 95);
//Neck up/down Left
Servo2(2,108);
//Neck up/down Right
Servo2(1,100);

//right ear hor
Servo(7,75);
//left ear hor
Servo(6,130);
//mouth
Servo(5,755);

////////////////////////////////////////////////////////////////////////////////
//Big servo PCA
////////////////////////////////////////////////////////////////////////////////

//Values for BIG SERVO degree as of 3/19/24
//test case angle 0015
//0, 45,55,70,90,98,100,105,108,133,135, 180
//Arms
Servo2(6,90);
Servo2(5,180);
//Left ear ver
Servo2(4,70);

//Right ear ver
Servo2(3,105);

//Neck hor
Servo2(0,105);

for(long k = 0;k<DELAY;k++) {
    Nop();
}
//Sleep
Servo(15,90);

Servo(13,160);
Servo(9,76);//right lid top
Servo(12,80);

//Right lid bottom
Servo(8,160);

Servo2(5,100);

//Left ear ver
Servo2(4,133);

//Right ear ver
Servo2(3,55);

//Neck up/down Left
Servo2(2,105);
//Neck up/down Right
Servo2(1,80);

Servo2(0,133);

for(long k = 0;k<DELAY;k++) {
    Nop();
}

Default();
}

void Sad(void){
    //Sad
    Default();

    for(long k = 0;k<(DELAY-1750000);k++) {
        Nop();
    }

    //blink
    Servo(13,160);
    Servo(9,76);//right lid top
    Servo(12,80);

    //Right lid bottom
    Servo(8,160);

    //Neck up/down Left
    //Servo2(2,108); //108 no ch]
    //Servo2(2,108); //108 no ch]

    //Neck up/down Right
    Servo2(1,100);

    for(long k = 0;k<(DELAY-1750000);k++) {
        Nop();
    }
}
Servo(13,80);
Servo(9,90);//right lid top
Servo(12,95);

//Right lid bottom
Servo(8,90);

for(long k = 0;k<(DELAY-1750000);k++) {
    Nop();
}

/////////////////////////////////
//Small servo PCA
/////////////////////////////////

//Values for SMALL SERVO degree as of 3/19/24
//0, 25,45 wrong,72, 75,755,76, 78, 80,83,86, 90, 95, 100,124,125,130,160, 180
//Left eye ball
Servo(15,95);
Servo(14,100);

//Left eye lids
Servo(13,80);
Servo(12,86);

//Right eye ball
Servo(11,90);
Servo(10,83);

//Right lid
Servo(9,90);
Servo(8,100);

//right ear hor
Servo(7,75);
//left ear hor
Servo(6,130);
//mouth
Servo(5,755);

/////////////////////////////////
//Big servo PCA
/////////////////////////////////

//Values for BIG SERVO degree as of 3/19/24
//test case angle 0015
//0, 45,55,70,90,98,100,105,108,133,135, 180
// Left ear ver
Servo2(4,133);

// Right ear ver
Servo2(3,55);
//   // Neck up/down Left
//   Servo2(2,108); // 108 no ch3
//   // Neck up/down Right
//   Servo2(1,100);
// Neck hor
Servo2(0,105);

for(long k = 0; k < DELAY; k++) {
    Nop();
}

// Full blink
Servo(13,160);
Servo(9,76); // right lid top
Servo(12,80);

// Right lid bottom
Servo(8,160);

for(long k = 0; k < (DELAY - 1750000); k++) {
    Nop();
}

Servo(13,80);
Servo(9,90); // right lid top
Servo(12,86);

// Right lid bottom
Servo(8,100);

Servo2(0,98);
Servo2(1,105);

Servo2(2,133);

for(long k = 0; k < DELAY; k++) {
    Nop();
}
/**Full blink**
Servo(13,160);
Servo(9,76); // right lid top
Servo(12,80);

// Right lid bottom
Servo(8,160);

for(long k = 0;k<(DELAY-1750000);k++) {
   Nop();
}

Servo(13,80);
Servo(9,90); // right lid top
Servo(12,86);

// Right lid bottom
Servo(8,100);

Servo2(0,135);

for(long k = 0;k<DELAY;k++) {
   Nop();
}

// Blink
Servo(13,160);
Servo(9,76); // right lid top
Servo(12,80);

// Right lid bottom
Servo(8,160);

for(long k = 0;k<(DELAY-1750000);k++) {
   Nop();
}

Default();

}

void Happy(void){
   // happy
   Default();
for(long k = 0;k<(DELAY-1750000);k++) {
    Nop();
}

//Blink
//Left lid
Servo(13,160);
Servo(9,76);//right lid top
Servo(12,80);

//Right lid bottom
Servo(8,160);

for(long k = 0;k<(DELAY-1750000);k++) {
    Nop();
}

//Moves to the left
/////////////////////////////////
//Small servo PCA
/////////////////////////////////

//Values for SMALL SERVO degree as of 3/19/24
//0, 25,45 wrong,72, 75,76, 78, 80,83,86, 90, 95, 100,125,160, 180

//ball
Servo(15,0);
//lid
Servo(13,80);
Servo(12,100);
//ball
Servo(11,86);
Servo(10,100);
//lid
Servo(9,90);
Servo(8,83);

//right ear hor
Servo(7,90);
//left ear hor
Servo(6,90);
//mouth
Servo(5,78);

/////////////////////////////////
//Big servo PCA
/////////////////////////////////

//Values for BIG SERVO degree as of 3/19/24
//test case angle 0015
//0, 45, 70, 90, 98, 100, 105, 133, 135, 180
//Left ear ver
Servo2(4,70);

//Right ear ver
Servo2(3,105);
//Neck up/down Left
Servo2(2,98);
//Neck up/down Right
Servo2(1,105);
//Neck hor
Servo2(0,98);

//Delay Block
for(long k = 0;k<DELAY;k++) {
    Nop();
}

//Moves to the right

//Small servo PCA

//Values for SMALL SERVO degree as of 3/19/24
//0, 25, 45 wrong 72, 75, 76, 78, 80, 86, 90, 95, 100, 125, 160, 180

//Blink
//Left lid
Servo(13,160);
Servo(9,76);//right lid top
Servo(12,80);

//Right lid bottom
Servo(8,160);

for(long k = 0;k<(DELAY-1750000);k++) {
    Nop();
}

//ball
Servo(15,95);
Servo(14,80);
//lid
Servo(13,80);
Servo(12,100);

//ball
Servo(11,124);
Servo(10,100);

//lid
Servo(9,90);
Servo(8,83);

////////////////////////////////////////////////////////
//Big servo PCA
////////////////////////////////////////////////////////

//Values for BIG SERVO degree as of 3/19/24
//test case angle 0015
//0, 45, 70, 90,98,100,105, 135, 180
//Neck hor
Servo2(0,135);

for(long k = 0;k<DELAY;k++) {
    Nop();
}

//Blink
//Left lid
Servo(13,160);
Servo(9,76);//right lid top
Servo(12,80);

//Right lid bottom
Servo(8,160);

for(long k = 0;k<(DELAY-1750000);k++) {
    Nop();
}

Default();
}

int main(void)
{
    SYSTEM_Initialize();
    //I2C
    initI2C(0x9D);  //BRG value 103 wtf do we make this
    initI2C2(0x9D);
T3CON=0x8000;
T1CON=0x8000;

long k; //used for delays

//Sets the pre scaler for the PCA
//Sets the period of the PWM to 20ms
preScaler();
preScaler2();
while (1)
{
    //Sets PCA in WAKE mode
    setup();
    setup2();

    receivedChar = getU2();

    //Values for BIG SERVO degree as of 3/19/24
    //0, 45 wrong, 70, 90, 135 maybe wrong, 180

    //Values for SMALL SERVO degree as of 3/19/24
    //0, 25, 45 wrong, 72, 75, 76, 78, 80, 86, 90, 95, 100, 160, 180

    Default();

    if(receivedChar == '!' )//If happy expression
    {
        Happy();
    }
    if(receivedChar == '?')//If sad expression
    {
        Sad();
    }
    if(receivedChar == '+')
    {
        Force();
    }
    if(receivedChar == '#')
    {
        Excited();
    }
}
// End while loop
return 1;
}//End main
The mechanical sketch describes the aesthetics and potential movements of the animatronic’s features. Each body part will have a set position they will move to based on the user’s selection from the phone app. When a different expression is selected, the body parts will move from their previous set positions to the newly selected expression’s positions.

Animatronic’s Four Positions:
• Happy: Mouth will smile, eye lids completely open, ears will perk up (move towards the top of the head), paired with what could be interpreted as a happy noise from the audio system.

• Sad: Mouth will frown, eye lids will close slightly, ears will fall downwards, paired with what could be interpreted as a sad noise from the audio system.

• Excited: Mouth open to a smile, eye lids completely open, ears perked up, head/neck will have a bobbing (up and down) movement. Paired with noise that could be interpreted as excitement.

• Force: Mouth will move to a closed position, eye lids will shut slightly to show focus, ears are perked, one arm will raise up facing straight out, head/neck will tilt slightly forward. Paired with the ‘force’ power noise used in Star Wars movies and shows.

117. Team Information

Wesley Cunningham, CpE.

Erin Keller, EE.

Dylan Mueller, CpE.

Carl Richter, EE.

118. Parts Lists

118.1. Parts List

Table 2: Parts List
<table>
<thead>
<tr>
<th>Qty.</th>
<th>Refdes</th>
<th>Part Num.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>B07M8HTQ3F</td>
<td>battery holder case with leads for 2AA, 3 AA, and 4 AA batteries with on/off switch</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>B0CFZY6XKH</td>
<td>4 rechargeable aa batteries with charger, 2 pack (8 batteries total)</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>B0BWYBFPW8</td>
<td>small speaker with leads for electronic projects</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>B07L6FZVT1</td>
<td>MG90D Servos, pack of 4</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>B07NQJ1VZ2</td>
<td>MG995 Servo, pack of 4</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>B07KY9C6R9</td>
<td>1.2mm metal rod pack</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>explorer 16/32 board with pic24fj128ga010 microcontroller</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>breadboard</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>digilent analog discovery 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>op-amp</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>1kohm resistor</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>820ohm resistor</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>680 resistor</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>BC547B</td>
<td>transistor</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>2kohm resistor</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>2.2kohm resistor</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>10kohm resistor</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>47uF capacitor</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>1uF capacitor</td>
</tr>
</tbody>
</table>

### 118.2. Material Budget List

Table 3: Material Budget List

<table>
<thead>
<tr>
<th>Qty.</th>
<th>Part Num.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Qty.</th>
<th>Part Num.</th>
<th>Description</th>
<th>Unit Cost</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASIN</td>
<td>Description</td>
<td>Quantity</td>
<td>Unit Price</td>
</tr>
<tr>
<td>---</td>
<td>--------------------</td>
<td>--------------------------------------------------</td>
<td>----------</td>
<td>------------</td>
</tr>
<tr>
<td>1</td>
<td>B07M8HTQ3F</td>
<td>battery holder case with leads for 2AA, 3 AA, and 4 AA batteries with on/off switch</td>
<td>1</td>
<td>$9.99</td>
</tr>
<tr>
<td>1</td>
<td>B0CFZY6XKH</td>
<td>4 rechargeable aa batteries with charger, 2 pack (8 batteries total)</td>
<td>1</td>
<td>27.94</td>
</tr>
<tr>
<td>1</td>
<td>B0BWYBFPW8</td>
<td>small speaker with leads for electronic projects</td>
<td>1</td>
<td>10.99</td>
</tr>
<tr>
<td>2</td>
<td>B07L6FZVT1</td>
<td>MG90D Servos, pack of 4</td>
<td>2</td>
<td>14.99</td>
</tr>
<tr>
<td>2</td>
<td>B07NQJ1VZ2</td>
<td>MG995 Servo, pack of 4</td>
<td></td>
<td>20.99</td>
</tr>
<tr>
<td>1</td>
<td>B07KY9C6R9</td>
<td>1.2mm metal rod pack</td>
<td>1</td>
<td>5.49</td>
</tr>
</tbody>
</table>

1 explorer 16/32 board with pic24fj128ga010 microcontroller

1 breadboard

1 digilent analog discovery 2

op-amp

2 1kohm resistor

1 820ohm resistor

2 680 resistor

3 BC547B transistor

2 2kohm resistor

1 2.2kohm resistor

1 10kohm resistor

2 47uF capacitor

2 1uF capacitor

Total $126.37
Table 4: Beginning of Fall Semester Gantt Chart Schedule

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Duration</th>
<th>Start</th>
<th>Finish</th>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Project Design</strong></td>
<td>46.38 days</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
<td></td>
</tr>
<tr>
<td><strong>Midterm Report</strong></td>
<td>46.38 days</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
<td></td>
</tr>
<tr>
<td><strong>Problem Statement</strong></td>
<td>46 days</td>
<td>Wed 8/24/23</td>
<td>Sun 10/9/23</td>
<td></td>
</tr>
<tr>
<td>Need</td>
<td>25.38 days</td>
<td>Wed 8/30/23</td>
<td>Sun 9/24/23</td>
<td>DM</td>
</tr>
<tr>
<td>Objective</td>
<td>25.38 days</td>
<td>Wed 8/30/23</td>
<td>Sun 9/24/23</td>
<td>WC/CR</td>
</tr>
<tr>
<td>Background</td>
<td>25.38 days</td>
<td>Wed 8/30/23</td>
<td>Sun 9/24/23</td>
<td>WC/DM/CR</td>
</tr>
<tr>
<td>Marketing Requirements</td>
<td>25.38 days</td>
<td>Wed 8/30/23</td>
<td>Sun 9/24/23</td>
<td>EK</td>
</tr>
<tr>
<td>Engineering Requirements Specific</td>
<td>25.38 days</td>
<td>Wed 8/30/23</td>
<td>Sun 9/24/23</td>
<td>WC/DM/CR</td>
</tr>
<tr>
<td><strong>Engineering Analysis</strong></td>
<td>33.38 days</td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
<td></td>
</tr>
<tr>
<td>Power Source</td>
<td>28.38 days</td>
<td>Mon 9/4/23</td>
<td>Mon 10/2/23</td>
<td>EK</td>
</tr>
<tr>
<td>Non-Inverting Amplifier</td>
<td>28.38 days</td>
<td>Mon 10/3/23</td>
<td>Mon 11/27/23</td>
<td>DM</td>
</tr>
<tr>
<td>Audio System</td>
<td>28.38 days</td>
<td>Mon 9/4/23</td>
<td>Mon 10/2/23</td>
<td>EK</td>
</tr>
<tr>
<td>Microcontroller</td>
<td>28.38 days</td>
<td>Mon 9/4/23</td>
<td>Mon 10/2/23</td>
<td>DM</td>
</tr>
<tr>
<td>PWM Signals</td>
<td>28.38 days</td>
<td>Mon 9/4/23</td>
<td>Mon 10/2/23</td>
<td>DM</td>
</tr>
<tr>
<td>Bluetooth Receiver</td>
<td>28.38 days</td>
<td>Mon 9/4/23</td>
<td>Mon 10/2/23</td>
<td>WC/CR</td>
</tr>
<tr>
<td>3D Printed Features</td>
<td>28.38 days</td>
<td>Mon 9/4/23</td>
<td>Mon 10/2/23</td>
<td>EK/CR</td>
</tr>
<tr>
<td>Servo Motors</td>
<td>28.38 days</td>
<td>Mon 9/4/23</td>
<td>Mon 10/2/23</td>
<td>WC</td>
</tr>
</tbody>
</table>

Table 5: Mid Fall Semester Gantt Chart Schedule

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Duration</th>
<th>Start</th>
<th>Finish</th>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accepted Technical Design</strong></td>
<td>46.38 days</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
<td></td>
</tr>
<tr>
<td><strong>Hardware Design: Phase 1</strong></td>
<td>33.38 days</td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
<td></td>
</tr>
<tr>
<td>Hardware Block Diagrams Levels 0 thru N (w/ FR tables)</td>
<td>33.38 days</td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
<td>WC/DM</td>
</tr>
<tr>
<td><strong>Software Design: Phase 1</strong></td>
<td>33.38 days</td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
<td></td>
</tr>
<tr>
<td>Software Behavior Models Levels 0 thru N (w/FR tables)</td>
<td>33.38 days</td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
<td>WC/DM</td>
</tr>
<tr>
<td><strong>Mechanical Sketch</strong></td>
<td>33.38 days</td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
<td>EK/CR</td>
</tr>
<tr>
<td><strong>Team Information</strong></td>
<td>33.38 days</td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
<td></td>
</tr>
<tr>
<td><strong>Project Schedules</strong></td>
<td>46 days</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
<td></td>
</tr>
<tr>
<td>Midterm Design Gantt Chart</td>
<td>28.38 days</td>
<td>Wed 8/30/23</td>
<td>Wed 9/27/23</td>
<td>WC/DM/CR</td>
</tr>
<tr>
<td>Final Design Gantt Chart</td>
<td>4.38 days</td>
<td>Mon 11/27/23</td>
<td>Fri 12/1/23</td>
<td>EK</td>
</tr>
<tr>
<td><strong>References</strong></td>
<td>46 days</td>
<td>Wed 8/30/23</td>
<td>Sun 10/15/23</td>
<td></td>
</tr>
<tr>
<td><strong>Midterm Parts Request Form</strong></td>
<td>50 days</td>
<td>Wed 8/30/23</td>
<td>Wed 10/19/23</td>
<td>WC/DM/CR</td>
</tr>
<tr>
<td>Midterm Presentation file submission</td>
<td>33 days</td>
<td>Wed 8/30/23</td>
<td>Mon 10/2/23</td>
<td></td>
</tr>
<tr>
<td>Midterm Design Presentations Day 1</td>
<td>0 days</td>
<td>Wed 10/4/23</td>
<td>Wed 10/4/23</td>
<td></td>
</tr>
<tr>
<td>Midterm Design Presentations Day 2</td>
<td>0 days</td>
<td>Wed 10/11/23</td>
<td>Wed 10/11/23</td>
<td></td>
</tr>
<tr>
<td><strong>Project Poster</strong></td>
<td>14 days</td>
<td>Tue 10/17/23</td>
<td>Tue 10/31/23</td>
<td></td>
</tr>
</tbody>
</table>
Table 6: End of Fall Semester Gantt Chart Schedule

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Duration</th>
<th>Start</th>
<th>Finish</th>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final Design Report</td>
<td>47 days</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
<td>EK</td>
</tr>
<tr>
<td>Abstract</td>
<td>47 days</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
<td>EK</td>
</tr>
<tr>
<td>Hardware Design: Phase 2</td>
<td>47 days</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
<td>EK</td>
</tr>
<tr>
<td>Modules 1...n</td>
<td>47 days</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
<td>EK</td>
</tr>
<tr>
<td>Simulations</td>
<td>47.38 days</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
<td>EK/DM/CR</td>
</tr>
<tr>
<td>Schematics</td>
<td>47.38 days</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
<td>EK/DM/CR</td>
</tr>
<tr>
<td>Software Design: Phase 2</td>
<td>47.38 days</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
<td>EK</td>
</tr>
<tr>
<td>Modules 1...n</td>
<td>47.38 days</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
<td>EK</td>
</tr>
<tr>
<td>Code (working subsystems)</td>
<td>47.38 days</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
<td>WC/DM/CR</td>
</tr>
<tr>
<td>System integration Behavior Models</td>
<td>47.38 days</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
<td>WC/DM</td>
</tr>
<tr>
<td>Parts Lists</td>
<td>47.38 days</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
<td>WC/DM</td>
</tr>
<tr>
<td>Parts list(s) for Schematics</td>
<td>47.38 days</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
<td>WC/DM</td>
</tr>
<tr>
<td>Materials Budget list</td>
<td>47.38 days</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
<td>WC/DM</td>
</tr>
<tr>
<td>Proposed Implementation Gantt Chart</td>
<td>47.38 days</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
<td>WC/DM</td>
</tr>
<tr>
<td>Conclusions and Recommendations</td>
<td>47.38 days</td>
<td>Tue 10/17/23</td>
<td>Sun 12/3/23</td>
<td>WC/DM</td>
</tr>
<tr>
<td>Parts Request Form for Subsystems</td>
<td>32 days</td>
<td>Wed 9/27/23</td>
<td>Sun 10/29/23</td>
<td>WC/DM/CR</td>
</tr>
<tr>
<td>Subsystems Demonstrations Day 1</td>
<td>0 days</td>
<td>Wed 11/15/23</td>
<td>Wed 11/15/23</td>
<td>WC</td>
</tr>
<tr>
<td>Subsystems Demonstrations Day 2</td>
<td>0 days</td>
<td>Wed 11/22/23</td>
<td>Wed 11/22/23</td>
<td>CR</td>
</tr>
<tr>
<td>Parts Request Form for Spring Semester</td>
<td>0 days</td>
<td>Fri 12/8/23</td>
<td>Fri 12/8/23</td>
<td>CR</td>
</tr>
</tbody>
</table>

Tables 4, 5, and 6 display the project’s Gantt Chart, which is a description of the project’s schedule. Table 4 displays the objectives during the beginning of the fall semester. During this time, the problem statement, which includes the need, objective, background, marketing requirements, and engineering requirements are written. This information defines the broad goals of the project and the societal impact it is intended to have. Once this information is completed, the engineering analysis is carried out. Each subsystem is examined and designed to ensure that the product will meet the functional requirements outlined in the problem statement. For example, the amount of power required for each subsystem is calculated in order to determine how much power must be supplied by the power source. This information allows an appropriate power source for the project to be determined.
Table 5 displays the objectives during the middle of the semester. During this time, phase one of the hardware and software design is carried out, and a mechanical sketch and project schedule is created. In addition, the midterm assignments are submitted which include a parts list, report, and a presentation. The hardware and software design begin to explain the connections between each subsystem and determine how each subsystem will satisfy the goals of the project. The project schedule improves organization and ensures the project will be completed by the deadline. The mechanical sketch helps define how each subsystem fits into the aesthetic of the final project. After the midterm assignments, a poster will be created for the project that displays the information in a way that is easier and quicker to read.

Table 6 describes the goals for the end of the semester. Phase 2 of the hardware and software design will be implemented. Simulations and schematics will be completed for the hardware, and code and system integration models will be completed for the software design. This phase of the design may point out potential incompatibilities in parts and allow for time to tweak the design before the physical implementation. Parts lists will also be created, and parts will be carefully selected to ensure they satisfy the design and requirements of the project. Then, certain subsystems will be physically implemented and demonstrated, and at the end of the semester a final report will be due containing all of the research and design work throughout the semester.

Table 7: Spring Semester Gantt Chart Hardware Schedule
Tables 7 and 8 describe the tentative schedule for the spring semester. The main objectives are to complete the design and implementation of the project, which will be split into the hardware and software aspects. The hardware, which is overseen by Carl, will consist of the 3D printed body parts, the audio circuit, power, and microcontroller. The printing of the body parts and the final assembly of the project will be the final step. The software design, which is overseen by Wesley, should be completed earlier in the semester than the hardware design to allow ample time for debugging. The main aspects of the software design will be code for the phone application and the microcontroller.

120. Conclusions and Recommendations EK

The project goals have been well defined throughout this report. An outline of the implementation has also been defined in this report. Overall, the project was successful.
However, miscalculations in terms of the original power consumption and number of servo motors needed led to the power source being too weak. In future implementations, a 7V battery should be used with a buck converter to provide the servos with 7V and the remaining devices with 5V. An amplifier with a gain of at least 5 should be designed and implemented to ensure the audio circuit is loud enough. The development of the audio circuit was considered lower priority in this implementation of the project, and team members focused their energy on creating a secure and dependable connection between the microcontroller and the phone application, as well as coding life-like expressions. Future implementations of the project could allow doctors to communicate to patients via the animatronic as well as allowing for a wider range of expressions and emotions to be displayed. The animatronic successfully connected to a phone application via Bluetooth and displayed multiple emotional responses based on user selections. The animatronics motions and expressions were familiar and brought a smile to people’s faces.

Students working on similar projects should budget plenty of time for adjusting the servo motors movements to produce expressions. They should start by focusing on 3D printing and building, learning to control the servo motors, and developing the phone application. Then, they should integrate control of the servo motors with the phone application and learn to feed audio data to the speaker. These tasks should be completed within four weeks. Then, students should design amplifiers, buck converters, and program the emotions for the animatronic.

121. References


122. Appendices


A small lightweight servo that is an upgraded version of the stalwart Towerpro MG90S digital servo but with more torque and greater speed. This little servo is perfect for aircraft, helicopters, multi-rotors, robots and other applications were you want a small, strong, fast servo. It is fitted with upgraded 0601-16 aluminium gears and shafts and has a balanced output.

The middle case is made of alloy which acts as a heatsink for the motor and electronics which enables them to keep cool when being worked hard. Supplied complete with a selection of 3 servo horns and mounting screws.

Features:
- High resolution
- Accurate positioning
- Fast control response
- Constant torque throughout the servo travel range
- Excellent holding power

Specs:
- Input Voltage: 4.8-6V
- Operating Torque: 2.1kg@4.8V, 2.4kg@6V
- Operating Speed: 0.10sec/60°@4.8V, 0.08sec/60°@6V
- Normal Servo Travel: 90° at 65° each way
- Gear: Top & bottom plastic, alloy middle bearing
- Bearing: Double ball bearing
- Gears: 0601-16 aluminium
- Gear Type: Digital
- Deadband Width: 1°
- Temperature Range: -25°C to 65°C
- Dimensions: 22.8 x 12.2 x 28.5mm
- Weight: 13g
- Output Spline: 23T (compatible with Futaba)
- Servo Plug: JR type
- Servo Lead Length: 250mm
- Spline: 23

Appendix C: PIC24FJ64GA004 [12] CR
Appendix F: BC547B Transistor EK
BC546/547/548/549/550

Switching and Applications
- High Voltage: BC546, BC547
- Low Noise: BC548, BC549
- Complement to BC556, BC563

NPN Epitaxial Silicon Transistor

Absolute Maximum Ratings $T_a=25^\circ$C unless otherwise noted

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{CEO}$</td>
<td>Collector-Base Voltage</td>
<td>60</td>
<td>V</td>
</tr>
<tr>
<td>$V_{CEO}$</td>
<td>BC546</td>
<td>50</td>
<td>V</td>
</tr>
<tr>
<td>$V_{CEO}$</td>
<td>BC547</td>
<td>30</td>
<td>V</td>
</tr>
<tr>
<td>$V_{CEO}$</td>
<td>BC548</td>
<td>45</td>
<td>V</td>
</tr>
<tr>
<td>$V_{CEO}$</td>
<td>BC549</td>
<td>30</td>
<td>V</td>
</tr>
<tr>
<td>$I_{CEO}$</td>
<td>Emitter-Base Voltage</td>
<td>9</td>
<td>mA</td>
</tr>
<tr>
<td>$I_{CEO}$</td>
<td>BC546</td>
<td>9</td>
<td>mA</td>
</tr>
<tr>
<td>$I_{CEO}$</td>
<td>BC547</td>
<td>5</td>
<td>mA</td>
</tr>
<tr>
<td>$S_{IN}$</td>
<td>Collector Current</td>
<td>150</td>
<td>mA</td>
</tr>
<tr>
<td>$I_{C}$</td>
<td>Collector Power Dissipation</td>
<td>500</td>
<td>mW</td>
</tr>
<tr>
<td>$T_J$</td>
<td>Junction Temperature</td>
<td>150</td>
<td>°C</td>
</tr>
<tr>
<td>$T_{ST}$</td>
<td>Storage Temperature</td>
<td>-65 to 150</td>
<td>°C</td>
</tr>
</tbody>
</table>

Electrical Characteristics $T_a=25^\circ$C unless otherwise noted

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Test Condition</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_{CM}$</td>
<td>Collector Offset Current</td>
<td>$V_{CC}=100, V_{BE}=0$</td>
<td>15</td>
<td>nA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_{CM}$</td>
<td>DC Current Gain</td>
<td>$I_{C}=1mA, V_{CC}=10V$</td>
<td>150</td>
<td>800</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>$I_{CM}$</td>
<td>Collector-Emitter Saturation Voltage</td>
<td></td>
<td>90</td>
<td>250</td>
<td>mV</td>
<td></td>
</tr>
<tr>
<td>$I_{CM}$</td>
<td>Base-Emitter Saturation Voltage</td>
<td></td>
<td>700</td>
<td>mV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_{CM}$</td>
<td>Base-Emitter On Voltage</td>
<td>$V_{CC}=15V, I_{C}=1mA$</td>
<td>600</td>
<td>720</td>
<td>mV</td>
<td></td>
</tr>
<tr>
<td>$S_i$</td>
<td>Current Gain Bandwidth-Product</td>
<td>$V_{CC}=10V, f=1MHz$</td>
<td>500</td>
<td>MHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_{CE}$</td>
<td>Output Capacitance</td>
<td>$V_{CC}=100, f=1MHz$</td>
<td>3.5</td>
<td>5</td>
<td>pF</td>
<td></td>
</tr>
<tr>
<td>$R_{BE}$</td>
<td>Breakdown Capacitance</td>
<td>$V_{CC}=5V, f=1kHz$</td>
<td>0</td>
<td>pF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NF</td>
<td>Noise Figure</td>
<td>BC546/547/548</td>
<td>2</td>
<td>4</td>
<td>dB</td>
<td></td>
</tr>
<tr>
<td>NF</td>
<td>BC546</td>
<td>2</td>
<td>4</td>
<td>dB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NF</td>
<td>BC548</td>
<td>2</td>
<td>4</td>
<td>dB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NF</td>
<td>BC549</td>
<td>1.4</td>
<td>4</td>
<td>dB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NF</td>
<td>BC556</td>
<td>1.4</td>
<td>3</td>
<td>dB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$g_e$ Classification

<table>
<thead>
<tr>
<th>Classification</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g_e$</td>
<td>150 to 250</td>
<td>250 to 450</td>
<td>450 to 600</td>
</tr>
</tbody>
</table>