Project Report

Project: EGR 326 Embedded System Design and Build

Prepared By:

John Bushman and Charles Norton

Submitted to: Dr. Kandalaft

EGR 326 Course Project

Fall Term 2024

Dec 7, 2024

Contents

1.0 Introduction and Design Background

This project addresses the design challenge of creating an interactive system inspired by the mimic creature from fantasy games like *Dungeons & Dragons* and *Dark Souls*. The objective was to build a responsive, multi-functional system that combines creative inspiration with technical rigor, meeting customer expectations for engagement, precision, and functionality.

The solution leverages the STM32F446RETx microcontroller to integrate multiple sensors, actuators, and display technologies. Key components include a rotary encoder, ultrasonic proximity sensor, Hall effect sensor, LCD RGB graphic display, shift register, seven-segment display, and servo motor. These devices enable proximity detection, input handling, motion control, and visual feedback.

The design approach involved configuring hardware and software to manage sensor inputs, control actuators, and synchronize outputs. This includes employing efficient communication protocols and control algorithms to ensure a seamless and robust system. The following sections detail the design objectives, technologies used, and the approach taken to achieve the project's goals.

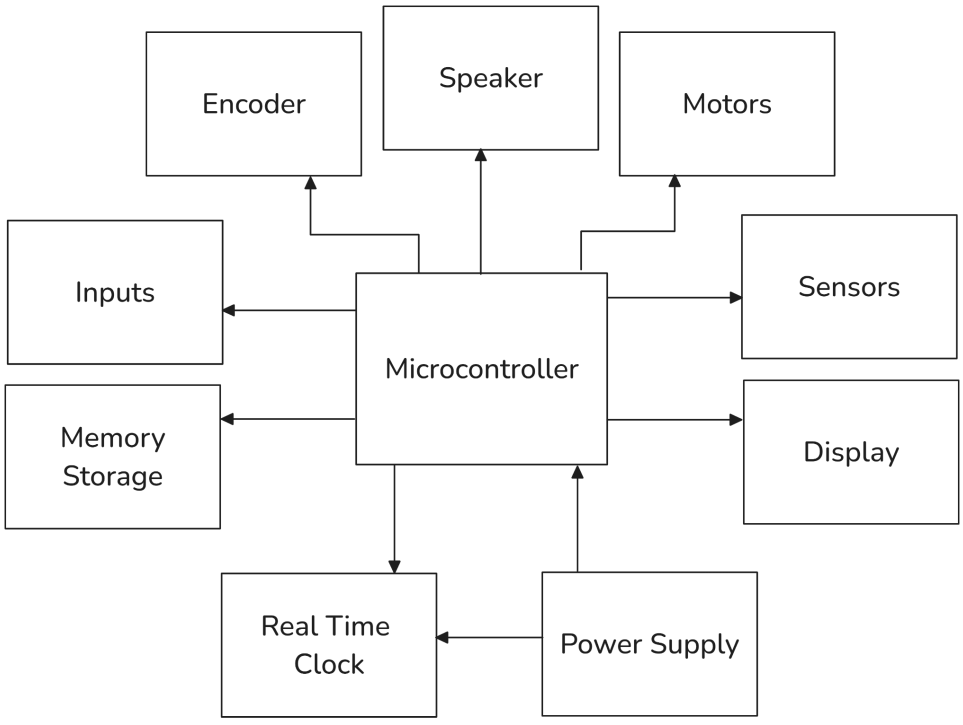

Figure 1. High-Level Block Diagram

2.0 Requirements and Functional Specifications and Verification

Table 1. Requirements and Verification Plan

Req #	Description	Customer Req. #	Design Const. #	Ver. Method	Testing Plan	Ver.
1	The system must be able to operate autonomously		1	Testing	The system will be turned on, disconnected from a computer, and then normal operations will be checked	
2	The system must implement a Microcontroller		3	Inspection		
3	The edges of the housing unit must have rounded corners		4	Inspection		
4	The edges on the housing unit must have rounded edges		4	Inspection		
5	The system must have all wires hidden from the user		4	Inspection		
6	The built-in Watchdog timer must be employed		5	Testing		
7	The system must be able to be verified without using any IDE		5	Testing	The system will be turned on, disconnected from any computer, and then normal operations will be checked	
8	The system must be constructed within the budget of 50$ beyond the kit		6	Analysis	Create a B.O.M. to keep a record of the cost of all materials used	
9	The system must include a display	1		Testing	Apply voltage to the display to ensure that it turns on	
10	The display must be able to express at least four different Expressions	1		Testing	A program will be created to cycle through each expression, and the display will be inspected	
11	The designer must draw the expressions	1		Inspection		

12	One of the digital monster expressions must be happy	1		Testing	A program will created to manually enter the digital monster happy expression, the expression will then be visually inspected on the display	
13	One of the digital monster expressions must be Hungry	1		Testing	A program will created to manually enter the digital monsters Hungry expression, the expression will then be visually inspected on the display	
14	One of the digital monster expressions must be Sleepy	1		Testing	A program will created to manually enter the digital monster Sleepy expression, the expression will then be visually inspected on the display	
15	One of the digital monster expressions must be Content	1		Testing	A program will created to manually enter the digital monsters' Content expression, the expression will then be visually inspected on the display	
16	The system must have a tongue	2		Inspection		
17	The system tongue must be moved by a motor	2		Testing	The tongue will be attached to a motor, the motor will be powered and programmed to move back and forth	
18	The system must move its tongue when approached by an object	2		Testing	A program will be created to interface between a motor and a proximity sensor. An object will be placed in front of the proximity sensor at different distances and the movement of the motor will be recorded.	
19	The system must make a sound	3		Testing	A program will be created to send a frequency to a sound maker and the devices will be audibly expected	
20	The system must make a distinct sound when happy	3		Testing	A program will be created to manually enter the happy state, the sounder will be audibly inspected	
21	The system must make a distinct sound when Hungry	4		Testing	A program will be created to manually enter the hungry state, the sounder will be audibly inspected	
22	The system must have a feeding mechanism	5		Testing	A sensor will be interfaced with the system to represent feed. When the sensor asserts a value to the system, the system will be notified by an increase in health points, indicated by the health bar.	

23	The system must have a petting mechanism	6		Testing	A sensor will be interfaced with the system to represent the petting mechanism. The mechanism will be triggered and the system will go into a happy state, this change will be visual. inspected.	
24	The system must have a happy expression when pet	6		Testing	A sensor will be interfaced with the system to represent the petting mechanism. The mechanism will be triggered and the system will go into a happy state, this change will be visually inspected.	
25	The system must be able to sense light	7		Testing	A photo sensor will be interfaced with the system and a program written to set the system into sleep mode when below 20 lux. This will be visually inspected by a change in the display.	
26	The system must go into sleep mode if the light level is low	7		Testing	A photo sensor will be interfaced with the system and a program written to set the system into sleep mode when below 20 lux. This will be visually inspected by a change in the display.	
27	The feeding countdown must be able to be displayed to the user	8		Testing	A program will be created to display a countdown after the feeding mechanism is triggered.	
28	The feeding timer must display 2 digits	8		Testing	A program will be created to display a countdown after the feeding mechanism is triggered.	
29	The system must indicate the health of the monster to the user	9		Testing	A program will be created to keep track of the health over time. As the health decreases, the system will have a visual indicator confirming whether or not the health tracking system worked properly	
31	The system must lose one health point every minute	10		Testing	A program will be created to keep track of the health over time. When in the hungry state the will health decreases, and the system will have a visual indicator confirming whether or not the digital monster is losing health points as expected	

32	The system must gain maximum health points when fed	10		Testing	A program will be created to keep track of the health of the monster and interface with a sensor. When the sensor is triggered the system will visually indicate an increase in health	
33	An encoder must be employed	11		Testing	A program will be created to interface between the microcontroller and encoder	
34	An encoder must be used to interface with a display	11		Testing	A program will be created to interface between the microcontroller and encoder. The encoder will visually change the display when moved to indicate that it is interfacing properly	
35	A menu must be implemented to set the time information of the system	12		Testing	A program will be created to display a menu. The menu will be interfaced with an encoder to set a time. The time value will be stored in an RTC and read back to ensure the value was correctly saved	
36	If the system is idle for 1 minute in the menu, the menu will be exited	12		Testing	A program will be created to implement a menu. The user will not interface with the menu for an extended period to check if the menu action cancellation worked properly	
37	The system must retain information powered off	13		Testing	A program will be created to interface between the microcontroller and an external memory. Data will be sent to the external memory. The device will then be power cycled and the data retrieved.	
38	The system must implement a Real-time clock	13		Testing	A program will be interfaced to send and return data from the RTC	
41	The system must use a USB 5V power supply to provide power to motor circuits and other sensors that require 5V	16		Testing	The power supply will be plugged in and the devices powered will be tested using a DMM to see if they are receiving the proper power	
42	The system components must be enclosed in a fixture	17		Inspection		
43	The system must make a distinct sound when Sleepy			Testing	A program will be created to manually enter the sleepy state, the sounder will be audibly inspected	

44	The system must make a distinct sound when the Content			Testing	A program will be created to manually enter the content state, the sounder will be audibly inspected	
45	The digital monster must be completed before the end of the semester			Inspection		
47	The system must have a battery to power the RTC			Testing	The power supply will be plugged in and the devices powered will be tested using a DMM to see if they are receiving the proper power	
48	The system may implement a PCB to interface the microcontroller with other components	18		Inspection		
49	The system may come with a set of instructions for the user interface		2	Inspection		
50	A help menu may be implemented		2	Testing	A program may be written to add an extra option in the menu. This will be displayed to the user.	
51	One of the digital monster expressions may be a dead expression			Testing	A program will created to manually enter the digital monster Dead expression, the expression will then be visually inspected on the display	
52	One of the digital monster expressions may be feeding			Testing	A program will created to manually enter the digital monster Feeding expression, the expression will then be visually inspected on the display	
53	The system may make a distinct sound when dead			Testing	A program will be created to manually enter the Dead state, the sounder will be audibly inspected	
54	The system may make a distinct sound when feeding			Testing	A program will be created to manually enter the Feeding state, the sounder will be audibly inspected	
55	A mechanical eye may be implemented to show distinct expressions			Testing	An eye will be designed and mounted, and the positions of the eye will be adjusted based on expression and visually inspected.	
56	A motor may be implemented to move the eye			Testing	A program will be written to interface between a motor attached to the eye	

	All I2C modules may be incorporated on the same port			Testing	A double-level shifting circuit will be created and the output of both clock and data lines will be measured with a testing code enabled	
	An audio amplification circuit may be implemented to improve sound quality/volume			Testing	The program will be run in normal operations, the sounds will play off a speaker and be audibly inspected.	

3.0 System Architecture

3.1 Hardware Specifications

Figure 2 shows the created wiring diagram for the digital monster pet system

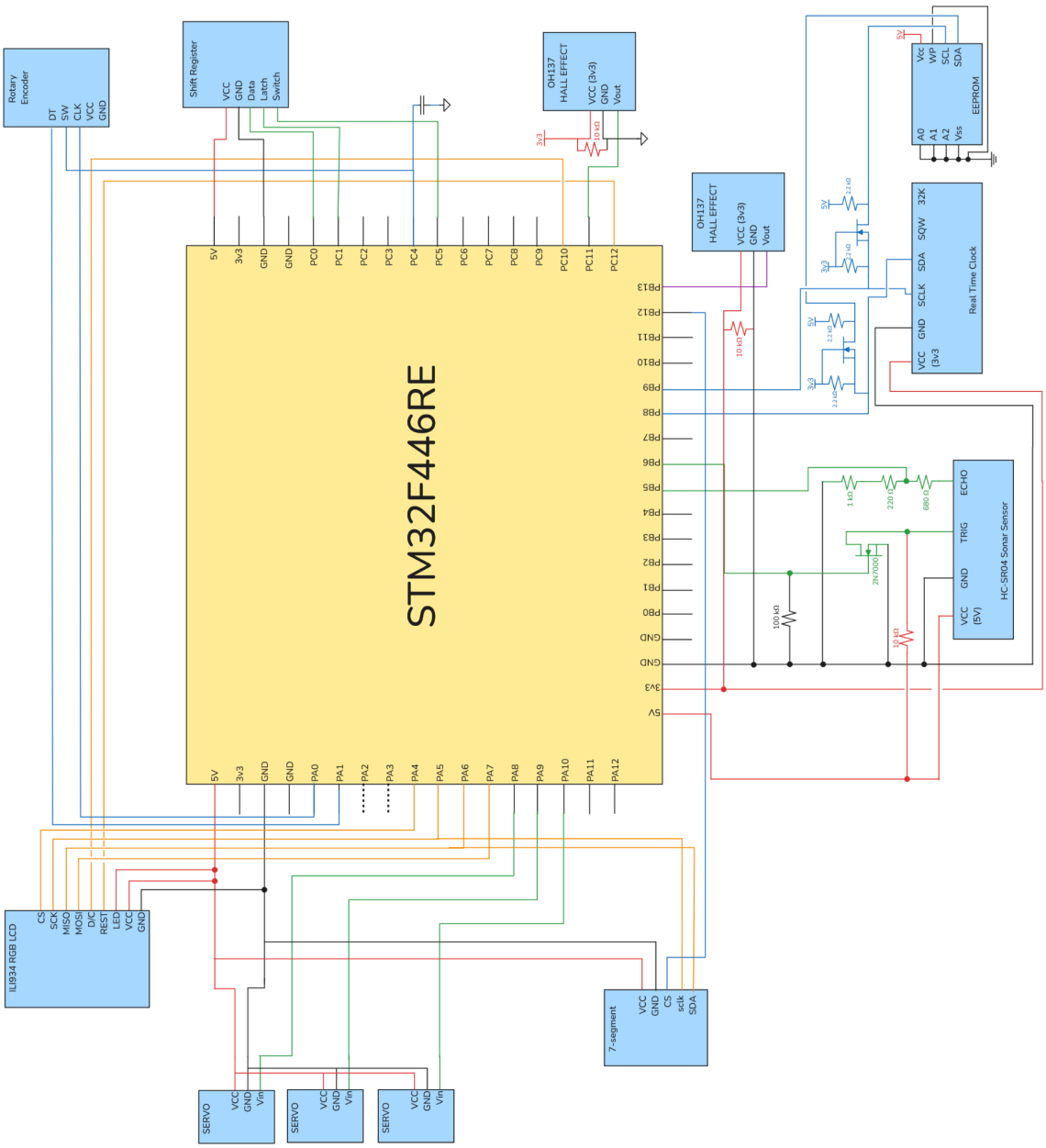

Figure 2. Wiring diagram for the digital pet monster system

3.1.1 Hardware Design

The hardware design for this project features a single STM32F446RETx microcontroller, serving as the central controller to manage all sensors, actuators, and peripherals. The system integrates various components through I2C and SPI communication protocols, enabling efficient data exchange and coordination between devices.

One of the special features of the design is the inclusion of a shift register IC to control a set of LEDs that visually display the health of the digital mimic, adding an interactive and dynamic element to the system. A speaker, paired with an LM386 audio amplification circuit, provides audio feedback to enhance user engagement.

For communication, the RTC and EEPROM share the I2C protocol, supported by a double-level shifter to handle both 5V and 3.3V logic seamlessly. This ensures compatibility and reliable operation between components with different voltage requirements. The seven-segment display and LCD RGB graphic display both utilize SPI for efficient and high-speed communication.

The hardware design balances functionality and creativity by combining these features into a cohesive system that is robust, responsive, and engaging, meeting the design objectives while maintaining technical precision.

3.2 Software Specifications

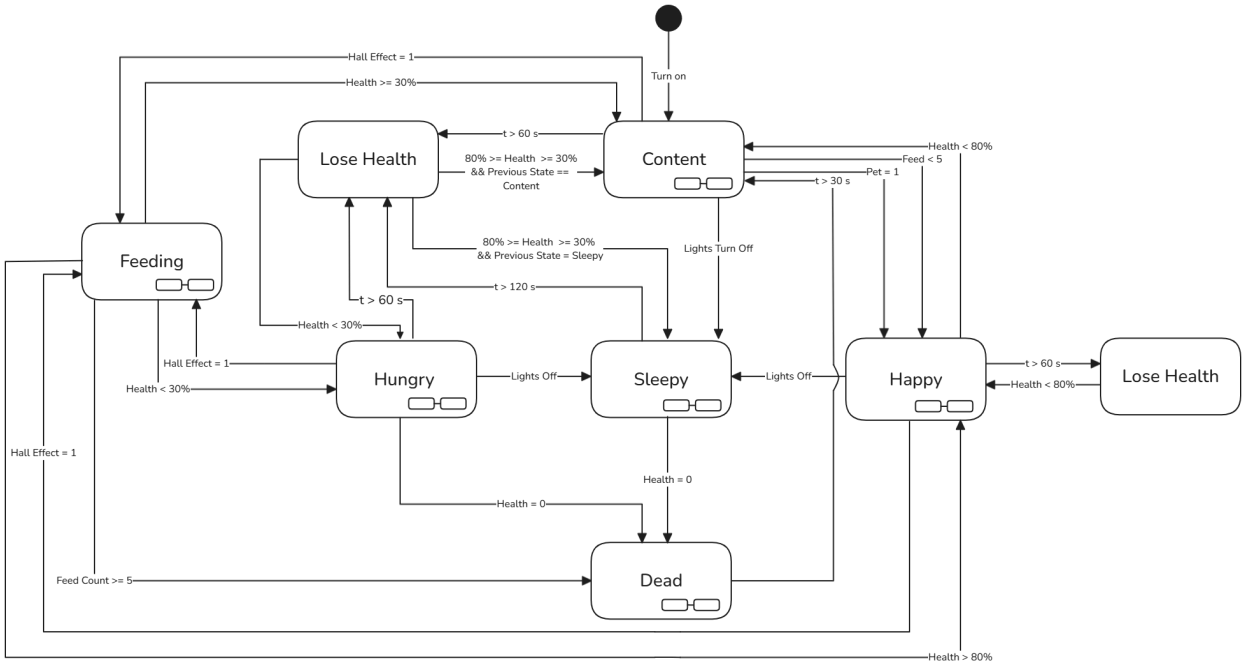

Figure 3. UML block diagram of the system

3.2.1 Software Design

A switch must be turned on after plugging the system into the wall using the AC/DC converter. Once switched on, it will default to the “content” state, and its initial health value will be 70% of its maximum health value.

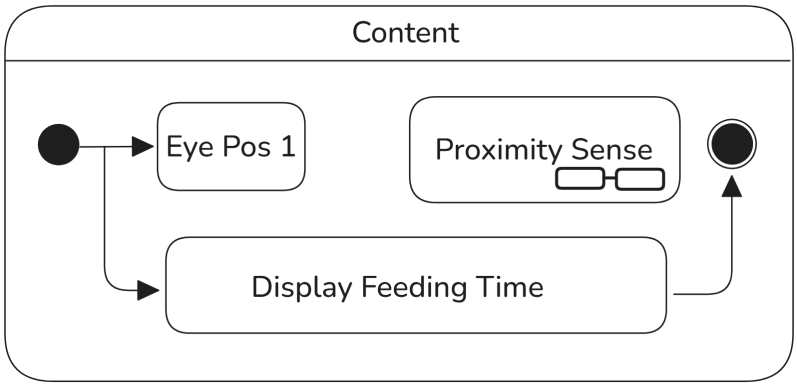

Figure 4. UML Content State Diagram

Within the content state, the monster's eye position will be neutral. During the content state, the proximity sensor and hall effect sensors are checked for petting and feeding. The feeding time is also displayed.

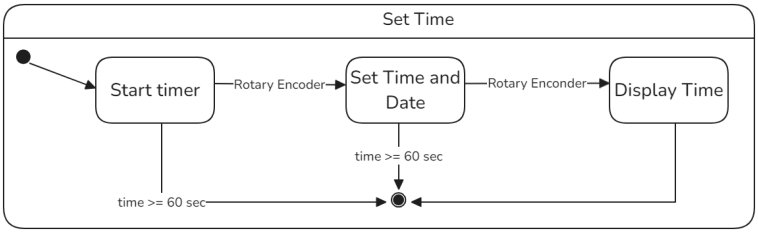

Figure 5. UML Set time diagram

When setting the time, a timer is initially started, such that if no input is made within a minute, the action of setting the time will be canceled. The time is set using the rotary encoder to select time values displayed on the LCD. After the time is set, it will be displayed on the LCD and resume to its previous state. Also, within the content state, the proximity sensor will be checked periodically to see if a person is approaching the monster.

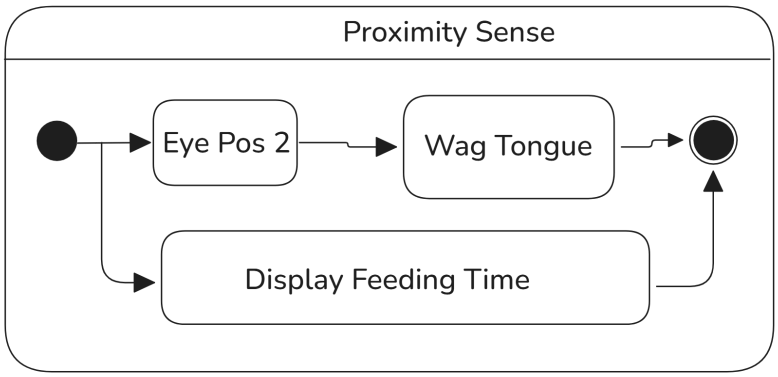

Figure 6. UML Proximity State Diagram

During the proximity check, the monster's eye opens wide; if a person is in proximity, the monster begins to wag its tongue.

While in the content state, if a hall effect sensor is activated in the head, effectively ‘petting’ the monster, then the monster will transition to the Happy state. Additionally, if a hall effect sensor is activated in the mouth of the monster (‘feeding’ the monster), then the monster will transition to the happy state. The monster will stay in the Happy state while its health is above 80% of its maximum health.

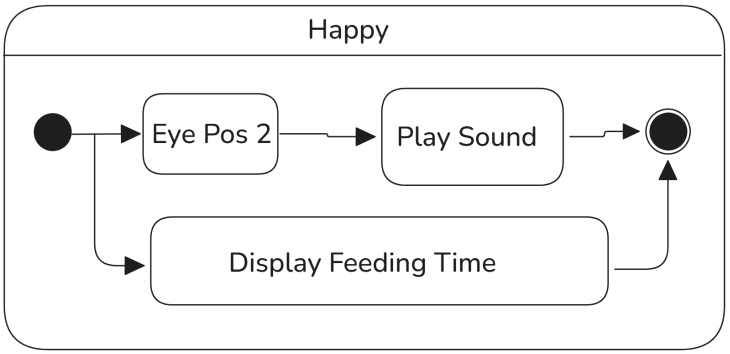

Figure 7. UML Happy State Diagram

While in the Happy state, the monster’s mechanical eye will open wide. The monster will wag its tongue and play a sound. If at any time during the Happy State,

Meanwhile, in the Content state, the health will drop by 10% of its maximum health every minute. If the monster’s health drops to zero, then the monster transitions to the Dead state. Another way the monster can transition to the dead state is by being fed more than 5 times while it is in the Happy State.

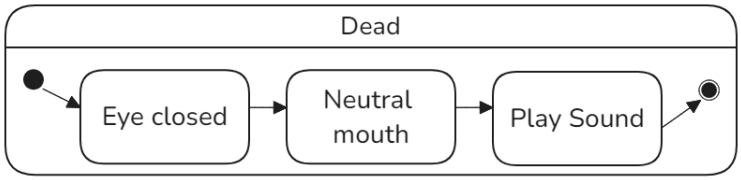

Figure 8. UML Dead State Diagram

In the Dead state, the eye closes, the mouth changes to a neutral position, and a sound plays. Once the monster is dead, the system shuts down.

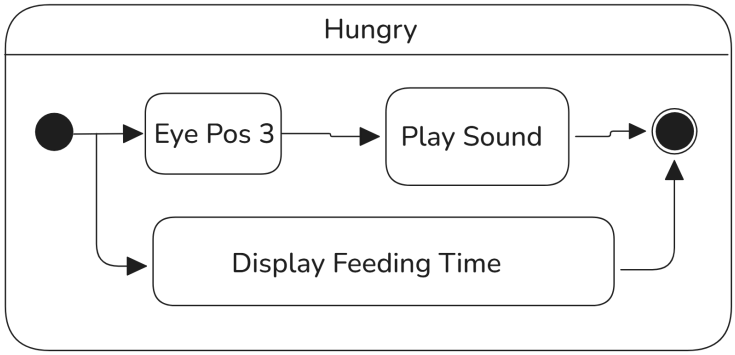

Figure 9. UML Hungry State Diagram

The monster will enter the hungry state if the health value drops below 30% of max health. Within the hungry state, the time can again be set and displayed. While in the hungry state, the monster's tongue will wag and it'll make a distinct sound. To exit the hungry state, the monster can either go to sleep when the lights are turned off, where it will continue to lose health. Or it can be fed by setting up a Hall Effect sensor. When fed, the health of the monster increases its health by 20 points. The monster can be overfed if it is currently in a happy state, where it will gain a feed counter. While feeding the monster, the monster's eye will be closed, the mouth will appear open, and a distinct sound will be played.

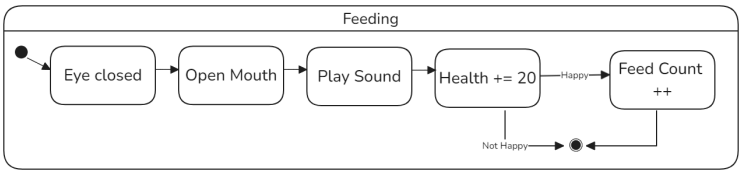

Figure 10. UML Feeding Diagram

The Sleepy state can be entered anytime from the Happy, Content, or Hungry states. The Sleepy state is entered once the ambient lighting becomes low. The monster remains in the Sleepy state while the ambient lighting remains low.

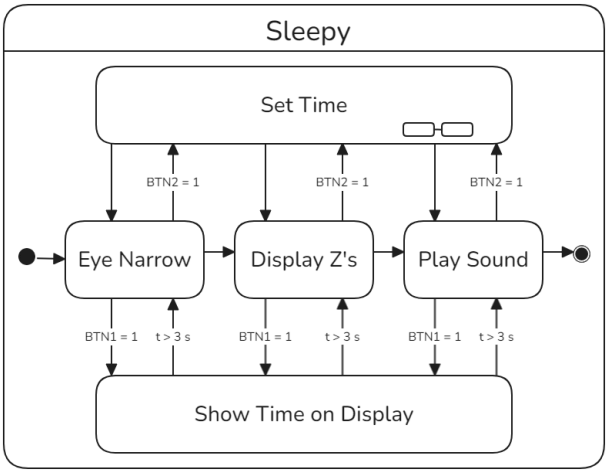

Figure 11. UML Sleepy Diagram

While in the sleepy state, the monster's eye narrows, Z-s are displayed on the screen, and the monster makes a sound.

3.2.2 Software Implementation

The software implementation for the interactive mimic creature system, as detailed in Section 3.2 of the "EGR326_FinalProject.pdf," involves several key modules that manage the system's states, sensor inputs, actuator controls, and display updates.

Software Development Environment and Tools

The software was developed using Keil IDE, with the programming language being C.

System Initialization and State Machine

Upon initialization, the system defaults to the "Content" state with an initial health value of 70% of its maximum health. Here is a code snippet illustrating the state machine:

```
// State Machine
typedef enum {
    CONTENT_STATE,
    HAPPY_STATE,
    HUNGRY_STATE,
    SLEEPY_STATE,
    DEAD_STATE
} system_state_t;

system_state_t current_state = CONTENT_STATE;

void update_state() {
    switch (current_state) {
        case CONTENT_STATE:
            // Check for petting, feeding, and proximity
            if (is_petted() || is_fed()) {
                current_state = HAPPY_STATE;
            } else if (health < 30) {
                current_state = HUNGRY_STATE;
            }
            break;
        case HAPPY_STATE:
            // Wag tongue, play sound
            if (health < 80) {
                current_state = CONTENT_STATE;
            }
```

```
                break;

            case HUNGRY_STATE:

                // Wag tongue, make sound

                if (is_fed()) {

                    health += 20;

                    current_state = CONTENT_STATE;

                } else if (health <= 0) {

                    current_state = DEAD_STATE;

                }

                break;

            case SLEEPY_STATE:

                // Sleep mode

                if (is_light_above_threshold()) {

                    current_state = CONTENT_STATE;

                }

                break;

            case DEAD_STATE:

                // Shut down system

                system_shutdown();

                break;

        }

    }
```

Proximity and Sensor Handling

The system periodically checks the proximity sensor and Hall effect sensors for petting and feeding. Here is an example of how this could be implemented:

```
// Proximity and Sensor Handling

void check_proximity() {

    uint16_t distance = read_proximity_sensor();

    if (distance < PROXIMITY_THRESHOLD) {

        // Move tongue

        move_tongue();

        // Open eye wide

        set_eye_position(EYE_OPEN_WIDE);

    }

}

void check_feeding() {

    if (is_hall_effect_sensor_triggered(HEAD_SENSOR)) {

        // Petting mechanism

        current_state = HAPPY_STATE;

    } else if (is_hall_effect_sensor_triggered(MOUTH_SENSOR)) {

        // Feeding mechanism

        health += 20;

        current_state = HAPPY_STATE;

    }

}
```

Display Management

The system updates the LCD RGB graphic display and the seven-segment display to provide visual feedback. Here is an example of how the display could be updated:

```
// Function to display a string on the LCD
void LCD_Display_String(char *str, uint8_t line) {
    // Clear the display line
    LCD_Clear_Line(line);
    // Send the string to the LCD using SPI
    HAL_SPI_Transmit(&hspi1, (uint8_t *)str, strlen(str), 100);
}

// Function to display a number on the LCD
void LCD_Display_Number(uint16_t number, uint8_t line) {
    char str[10];
    sprintf(str, "%d", number);
    LCD_Display_String(str, line);
}

// Example usage in the main loop
void update_display() {
    // Display health on LCD
    LCD_Display_String("Health: ", LINE1);
    LCD_Display_Number(health, LINE2);

    // Display time on LCD if in content state
    if (current_state == CONTENT_STATE) {
        LCD_Display_String("Time: ", LINE3);
        LCD_Display_Number(hours * 100 + minutes, LINE4);
    }
}
```

Time Setting and Health Management

The system allows setting the time using a rotary encoder and displays the feeding countdown. Here is an example of how the time setting and health management could be implemented:

```
void set_time() {
    // Start timer to cancel action if no input within a minute
    start_timer(60);
    // Use rotary encoder to select time values
    uint8_t hours = get_encoder_value();
    uint8_t minutes = get_encoder_value();
    // Display time on LCD
    LCD_Display_String("Time: ", LINE3);
    LCD_Display_Number(hours * 100 + minutes, LINE4);
    // Store time in RTC
    set_RTC_time(hours, minutes);
}
```

These code snippets illustrate how the system manages its states, handles sensor inputs, controls actuators, and updates displays, aligning with the detailed specifications outlined in Section 3.2. The complete source code can be found in Appendix D of the document.

4.0 Conclusion

The system was designed to be a responsive, multi-functional entity that combined creative inspiration with technical rigor, ensuring engagement, precision, and functionality.

Various components such as the rotary encoder, ultrasonic proximity sensor, Hall effect sensors, LCD RGB graphic display, shift register, seven-segment display, and servo motor were integrated using I2C and SPI communication protocols. This integration enabled efficient data exchange and coordination between devices, ensuring a seamless and robust system operation.

The software specifications were defined and implemented. The system defaulted to the "Content" state upon initialization, with an initial health value of 70% of its maximum health. The state machine was effectively managed, transitioning between Content, Happy, Hungry, Sleepy, and Dead states based on sensor inputs and user interactions. The time setting feature, utilizing a rotary encoder, was successfully implemented, allowing users to set and display the time on the LCD.

The display management system was also a key aspect of the project. The LCD RGB graphic display and the seven-segment display were updated periodically to reflect the monster's health, time, and other relevant information. The audio feedback system, including distinct sounds for different states, was implemented using a speaker and an LM386 audio amplification circuit, enhancing user engagement.

The system's ability to sense light levels and transition into sleep mode when the light level was below 20 lux was another significant feature. The feeding mechanism, which involved Hall effect sensors and resulted in health point increases, was thoroughly tested and verified. The system's health management, including the loss of health points over time and the gain of health points upon feeding, was implemented.

In conclusion, the project was a comprehensive demonstration of embedded system design and build principles. It showcased the integration of multiple sensors, actuators, and display technologies, all managed by a software framework. The system's functionality, precision, and user engagement were verified through extensive testing, ensuring that all requirements and specifications were met. The completion of this project highlights the capabilities of embedded systems in creating interactive and responsive devices that combine technical innovation with creative design.

5.0 Project Budget and Schedule

5.1 Budget

Table 2 shows a list of all materials used to complete this project's exterior design, for a more comprehensive list of materials used to complete the project see Appendix B.

Table 2. Materials used for the exterior design

Component	Quantity	Cost
Acrylic Paint	1	$5
Hinges	1	$5
Screws	12	$1.5
3D printing filament	718 cm^3	$17.81
Spray Paint	1	$10

5.2 Schedule

EGR 326: Timeline of Final Project									
Charles Norton	Bushman John			Start Date	10/14/2024			End Date	12/6/2024
Color Legend =>>>	15% Completion	30%	60%	90%	100%				
Task Category	Resposibility	Submition Date	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14
Possibly More filiment		N/A							
Speaker		10/26/2024	100 %	100%	100%	100%	100%	100%	100%
Programing									
Hall effect sensor set up		11/8/2024	100 %	100%	100%	100%	100%	100%	100%
Servo Motor set up		11/8/2024	15%	90%	100%	100%	100%	100%	100%
Ultra sonic sensor set up		11/8/2024	80%	80%	100%	100%	100%	100%	100%
Stepper motor set up		11/8/2024	90%	90%	100%	100%	100%	100%	100%
Rotary encoder set up		11/8/2024	80%	100 %	100%	100%	100%	100%	100%
LCD set up		11/8/2024	100 %	100%	100%	100%	100%	100%	100%
RTC set up		11/8/2024	100 %	100%	100%	100%	100%	100%	100%
MicroSD breakout set up		11/8/2024	0 %	10%	20%	60 %	90%	100%	100%
Setting up ADC library file		10/25/2024	0 %	10%	20%	60 %	100%	100%	100%
Create timer library file		11/8/2024	0 %	10%	20%	60 %	90%	100%	100%
Set up push button		11/8/2024	0 %	10%	40%	50%	90%	100%	100%
Set up temperature sensor		11/8/2024	100 %	100%	100%	100%	100%	100%	100%
Health system		11/15/2024	0 %	10%	40%	60 %	100%	100%	100%
State system		11/8/2024	0 %	10%	40%	60 %	100%	100%	100%
Hall effect adjust the hunger level		11/15/2024	0 %	10%	20%	60 %	90%	90%	100%
Hall effect for petting		11/8/2024	0 %	10%	20%	60 %	90%	100%	100%
LCD menu GUI		11/15/2024	0 %	0 %	20%	90%	90%	100%	100%
Distinct eye movement for each state		11/15/2024	0 %	0 %	40%	50%	100%	100%	100%
Distinct tail movement for each state		11/15/2024	0 %	10%	40%	90%	100%	100%	100%
Distinct sound for each state		11/15/2024	0 %	10%	20%	50%	100%	100%	100%
Servo Motor postion based on health		11/8/2024	0 %	10%	20%	50%	100%	100%	100%
Photo cell set up		11/8/2024	0 %	0 %	20%	60 %	100%	100%	100%
Photo cell effecting the sleeping state		11/15/2024	0 %	0 %	40%	90%	100%	100%	100%
watchdog timer set up		11/8/2024	0 %	10%	40%	50%	90%	100%	100%
Programing the RTC to display time info		11/8/2024	90%	100%	100%	100%	100%	100%	100%
Programing seting time		11/8/2024	0 %	10%	40%	50%	60 %	100%	100%
Program different expressions for each state		11/15/2024	0 %	10%	40%	50%	100%	100%	100%
Adding GIF to the LCD display		N/a	0 %	10%	40%	100%	100%	100%	100%

Figure 11. Created Gantt chart

Prototype									
Play sound from MicroSD break out and speaker		11/15/2024	0 %	10%	40%	60 %	90%	100%	100%
animechtronic eye		11/8/2024	0 %	10%	40%	90%	100%	100%	100%
3D print a Housing for the STM		11/15/2024	0 %	0 %	10%	50%	90%	100%	100%
arm housing with push buttons and encoder		11/22/2024	0 %	0 %	10%	50%	60 %	90%	100%
Feeding Spoon		11/22/2024	0 %	0 %	10%	50%	100%	100%	100%
Petting brush		11/22/2024	0 %	0 %	20%	90%	100%	100%	100%
Tail motor mount		11/8/2024	0 %	0 %	20%	50%	60 %	90%	100%
Feeding Hole		11/22/2024	0 %	0 %	20%	50%	90%	90%	100%
Ultra sonic sensor placement		11/8/2024	0 %	0 %	40%	60 %	90%	100%	100%
3D pring the LCD mounting		11/15/2024	0 %	0 %	40%	90%	100%	100%	100%
3D print Head housing		11/22/2024	0 %	0 %	40%	90%	90%	100%	100%
Create the body frame		11/22/2024	0 %	0 %	20%	50%	100%	100%	100%
PCB design		11/15/2024	0 %	0 %	20%	50%	100%	100%	100%
PCB creation		11/22/2024	0 %	0 %	20%	90%	90%	100%	100%
3D print a mount for the ultra sonic sensor		11/15/2024	0 %	0 %	0 %	50%	100%	100%	100%
Bug Fixes									
Ultra Sonic Sensor timing adjustment		11/8/2024	0 %	20%	40%	50%	60 %	90%	100%	100%
Rotary encoder signal bounces		11/8/2024	0 %	20%	50%	60 %	100%	100%	100%	100%
Key pad function incorperate interrupts		11/8/2024	0 %	20%	60 %	90%	90%	100%	100%	100%
Set up the stepper to step in an interrupt		11/8/2024	0 %	0 %	60 %	90%	100%	100%	100%	100%
Demo									
oject Demonstration during final class session	J C	12/4/2024	0 %	0 %	0 %	0 %	0 %	0 %	100%	100%
Final project validation walkthrough in lab	J C	12/6/2024	0 %	0 %	0 %	0 %	0 %	0 %	100%	100%

Figure 12. Created Gantt chart

6.0 Revision History

Table 3. Code Revisions History

Revision Number	Date	Author	Changes
1	Nov 2, 2024	John Bushman	Created a Final project in Keil and added file from previous labs
2	Nov 2, 2024	John Bushman	Set up the RTC connection, adding the rotary encoder, and the LCD together to create a menu system
3	Nov 10, 2024	Charles Norton	Fixed the Rotary encoder, added code for the feeding timer, and health management system with the LED bar
4	Nov 10, 2024	Charles Norton	Fixed the health bar code and made it functionally compete as well as installed
5	Nov 14, 2024	Charles Norton	EEPROM Program and library created
6	Nov 15, 2024	John Bushman	ADC code and libraries created for the photo cell
7	Nov 22, 2024	John Bushman	Added the rest of the libraries necessary to complete the project
8	Nov 29, 2024	Charles Norton	Fixed issue with the hall effect and ADC not working
9	Nov 30 2024	Charles Norton	Sounds working and added to the Main
10	Nov 30 2024	John Bushman	Main states and expressions added
11	Dec 2 2024	Charles Norton	Project functionally Complete

Appendix A – Electrical Schematics

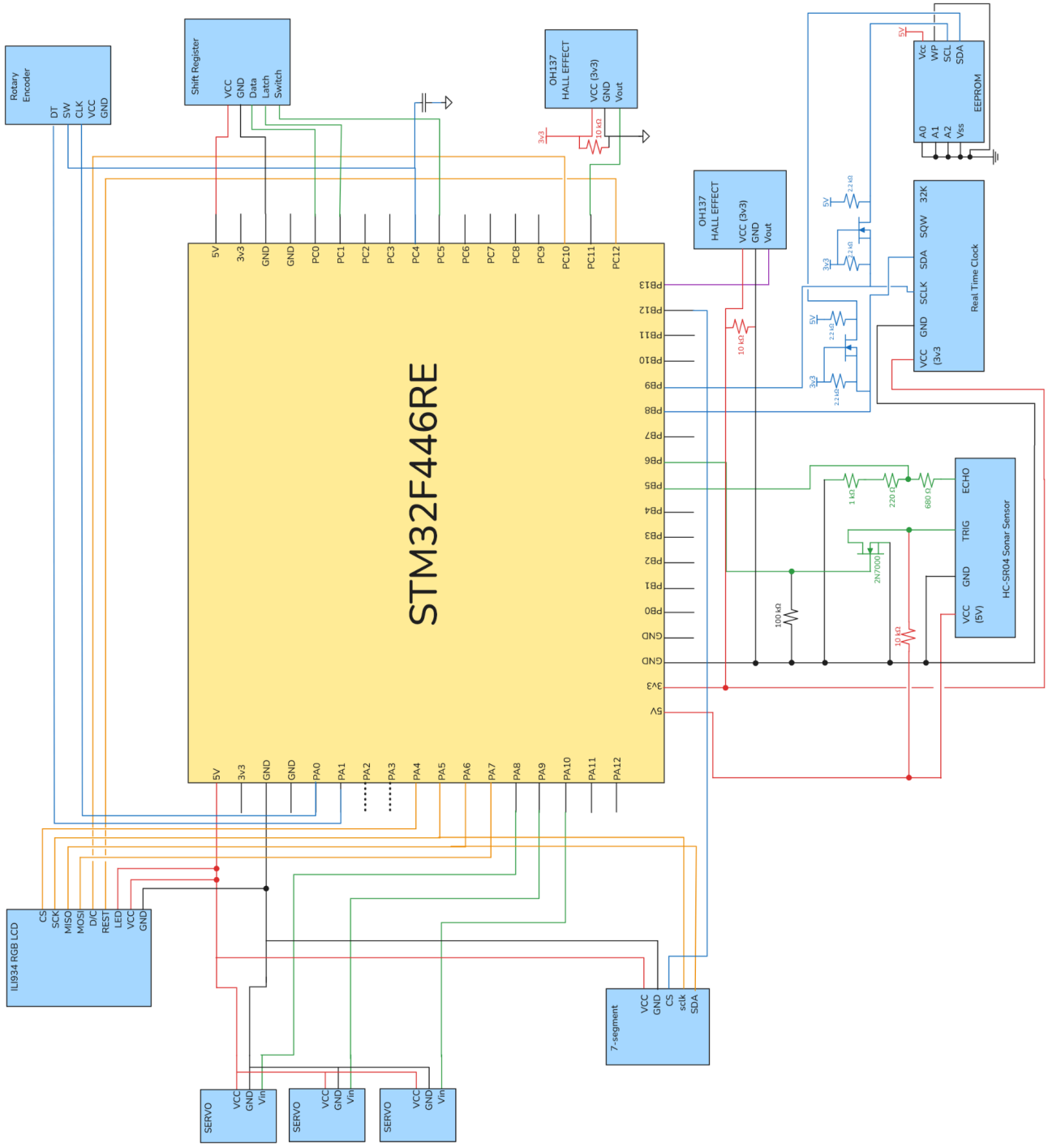

Figure A-1. Wiring diagram for the digital monster pet system

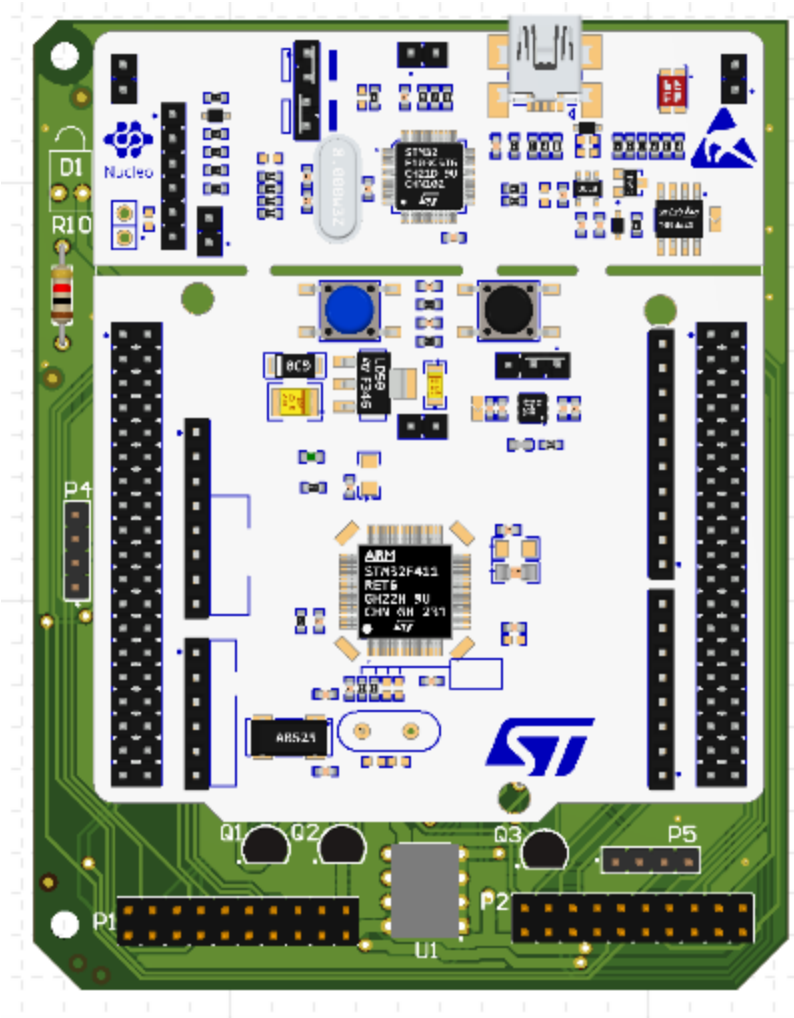

Figure A-2. PCB design for the digital monster pet system

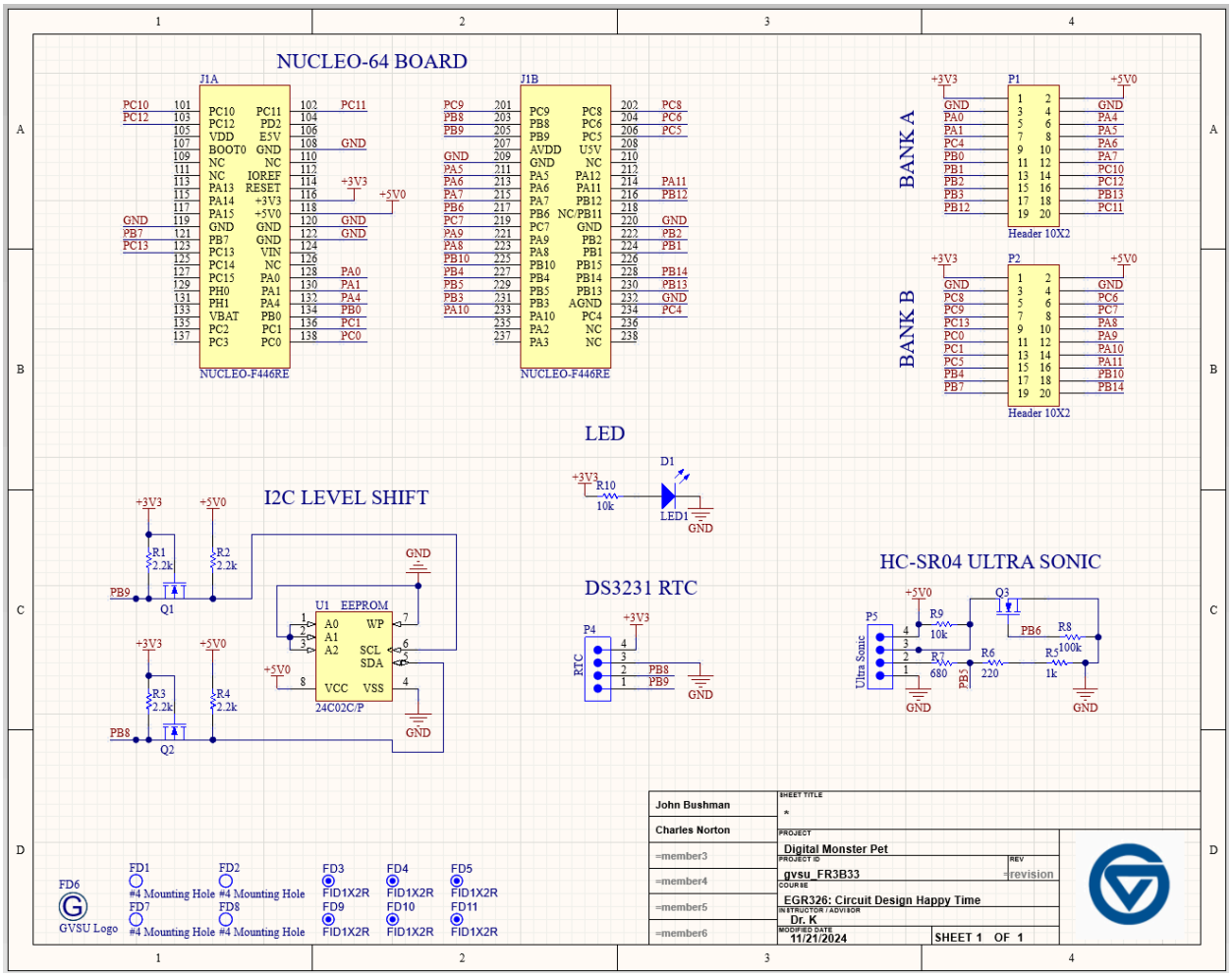

Figure A-3. PCB schematic for the digital monster pet system

Appendix B – Bill of Materials

Table 4. Bill of materials

Material	Quantity	Cost
Acrylic Paint	1	$5
Hinges	1	$5
Screws		$1.5
3D printing filament	718 cm^3	$17.81
Spray Paint	1	$10
LEDs	8	*Included in EGR 326 Kit*
SG90 Servos	3	$7
STM32F446Re	1	*Included in EGR 326 Kit*
10x2 Header Pins 2	6	*Included in EGR 326 Kit*
2n7000	3	*Included in EGR 326 Kit*
2.2kΩ resistor(s)	4	$0.04
1kΩ resistor(s)	1	$0.02
22Ω resistor(s)	1	$0.02
680Ω resistor(s)	1	$0.02
100kΩ resistor(s)	1	$0.02
10kΩ resistor(s)	6	$0.12
24C0C EEPROM	1	*Included in EGR 326 Kit*
DS32311 RTC	1	*Included in EGR 326 Kit*
HC-SR04	1	*Included in EGR 326 Kit*
Magnets	2	*Included in EGR 326 Kit*
GL5528 Photo Cell	1	*Included in EGR 326 Kit*
Lithium Battery	1	*Included in EGR 326 Kit*
Hall Effect OH137	2	*Included in EGR 326 Kit*

Table 4. Bill of materials Cont.

Material	Quantity	Cost
8 Digit 7 Seg Display	1	*Included in EGR 326 Kit*
Rotary Encoder	1	*Included in EGR 326 Kit*
Shift Register	1	0.04
LM386	1	$0.39
Total		$46.94

Appendix C – Mechanical Drawings

Appendix D – Source Code

```
/*********************************************************************************
NAME:John Buhman & Charles Norton
COURSE: EGR 326 Embedded Systems
DATE: 12/9/2024
DESCRIPTION: Main Program For Monster Pet
*********************************************************************************/

*/


/*********************************************************************************
                                                                          LIBRARIES
*********************************************************************************/
#include "main.h"

/*********************************************************************************
                                                                          Global
Variables
*********************************************************************************/
typedef struct{
    uint32_t eye;
    uint32_t eyelid;
    uint32_t tongue;
    bool is_close;
}Servo;

Servo Mimic_Servo = {1000,1000,1000,false};
bool test = 0;
bool tongue_wag = 0;

uint8_t random_seconds = 1;
uint32_t random_seconds_counter = 0;

uint8_t swPress = 0;
uint8_t foodCount = 0; // IN CASE OF ADDING OVERFEEDING
uint8_t petFlag = 0;

bool Expression_Transition = true;

unsigned long long Time_New;
unsigned long long Time_Old;

int last = 0; // FOR MENU SYSTEM
int current;  // FOR MENU SYSTEM
double period; // FOR MENU SYSTEM
int pos = 0; // FOR ROTARY ENCODER
int servo = 1000;
char value = 0; // FOR MENU SYSTEM
char timeDate[14]; // FOR RTC
```

```
char* health_ptr;


//PHOTO SENSOR STUFF
int lightLvl = 0;

///////////////// SEVEN SEGMENT STUFF ////////////////////////////////
int count = 60;
int old_count = 0;

int num[] = {0x7E,  //0
                              0x30,  //1
                              0x6D,  //2
                              0x79,  //3
                              0x33,  //4
                              0x5B,  //5
                              0x5F,  //6
                              0x70,  //7
                              0x7F,  //8
                              0x7B,  //9
                              0x01}; //-


////////////////////////// MENU STUFF /////////////////////////////
//inital screen
int menu = startUpScreen;
//init state
int state = MENU_STATE;
int prev_state;

//temp valu is for the
uint8_t tmp = 0;

uint8_t CNT_Old;
uint8_t CNT_New;



////////////////////////// RTC VALUES ////////////////////////////////
//RTC_data and pointER
char RTC_data;
char *RTC_Data_ptr = &RTC_data;

bool refresh_WDT = false;

uint8_t RTC_Hour;
uint8_t RTC_Minute;
uint8_t RTC_Second;

uint8_t RTC_Year;
uint8_t RTC_Month;
```

```
uint8_t RTC_Date;
uint8_t RTC_Day;

uint8_t menu_flag = 1;

int distance = 0;




/*******************************************************************************
                                                                          MAIN
*******************************************************************************/
int main(void)
{
    __disable_irq();

//=====initilaztion functions=====
      hallEffect_pinInit();
    Display_Init();
        I2C_init();
        Encoder_init();
      stepper_init();

        TIM1_setup();
        tim5_init();
        Timer6_Init();
        Timer7_Init();
        init_ShiftRegister();
        HealthCurrent();
        Init_seq();
        adc_init();
        WDT_init();
        sonar_gpio_init();

    __enable_irq();


    menu = mainMenu;
    health_ptr = &health;


    Store_Current_Health();

    delayMS(5);

    Retrieve_Health(); // Retrieve Current Health from EEPROM

    delayMS(1);

    HealthCurrent(); // Display Current Health on the Health Bar
```

```
      while (1)
    {
            CNT_Old = TIM2->CNT;

            if(CNT_New != CNT_Old) IWDG->KR |= 0xAAAA; // If the rotary encoder is used,
refresh the WDT

            if((menu == An_Expression_State) || (refresh_WDT == true)) {
                WDT_reset();
                refresh_WDT = false;
            }

            switch(state){
//----------------------------content-------------------------------
                case(content):
                    IWDG->KR |= 0xAAAA;

                    if(lightLvl <=2 ){ // check sleepy
                        petFlag = 0;
                        state = sleepy;
                        Play_Tone_2();
                        menu = ExpSleepy;
                        Eyelid_Move_To(2570);
                        Expression_Transition = true;
                        break;
                    }

                    if(health>=5){     // check happy via food
                        state = happy;
                        menu = ExpHappy;
                        Play_Tone_1();
                        Eyelid_Move_To(1500); // open
                        Expression_Transition = true;

                        break;
                    }

                    if(petFlag){        // check happy via petting
                        Time_Old = millis();
                        state = happy;
                        menu = ExpHappy;
                        Eyelid_Move_To(1500); // open
                        Expression_Transition = true;
                        break;
                    }

                    if(health==0){     // check dead
                        state = dead;
                        menu = ExpDead;
                        Play_Tone_3();
```

```
                    Eyelid_Move_To(2570); // closed
                    Expression_Transition = true;
                    break;
                }

                if(health <= 2){ // check hungry
                    state = hungry;
                    Play_Tone_3();
                    menu = ExpHungry;
                    Eyelid_Move_To(2200); //closed
                    Expression_Transition = true;
                    break;
                }


                break;


//-----------------------------HAPPY--------------------------------
            case(happy):
                IWDG->KR |= 0xAAAA;
                if(lightLvl <=2 ){ // check sleepy
                    petFlag = 0;
                    state = sleepy;
                    Play_Tone_2();
                    menu = ExpSleepy;
                    Eyelid_Move_To(2570);
                    Expression_Transition = true;
                    break;
                }

                if(petFlag){            // begin pet flag cool down
                    Time_New = millis();
                    if(Time_New-Time_Old > 3000){
                        petFlag = 0;
                        Expression_Transition = true;
                        break;
                    }
                    break;
                }

                if(health<5 && health >2){
                    state = content;
                    menu = ExpContent;
                    Play_Tone_2();
                    Eyelid_Move_To(1900); // open (Happy)
                    Expression_Transition = true;
                    break;
                }
```

```
                    //menu = ExpHappy;

                    break;


//-----------------------------HUNGRY-----------------------------------
                case(hungry):
                    IWDG->KR |= 0xAAAA;

                    if(lightLvl <=2 ){ // check sleepy
                        petFlag = 0;
                        state = sleepy;
                        Play_Tone_4();
                        menu = ExpSleepy;
                        Eyelid_Move_To(2570);
                        Expression_Transition = true;
                        break;
                    }

                    if(health>2){
                        state = happy;
                        menu = ExpHappy;
                        Play_Tone_1();
                        Eyelid_Move_To(1500); // open (HappY)
                        Expression_Transition = true;
                        break;
                    }

//                  if(petFlag){        // check happy via petting
//                      Time_Old = millis();
//                      state = happy;
//                      menu = ExpHappy;
//                      Eyelid_Move_To(1500); // open
//                      Expression_Transition = true;
//                      break;
//                  }

                    if(health==0){     // check dead
                        state = dead;
                        menu = ExpDead;
                        Play_Tone_3();
                        Eyelid_Move_To(2570); // closed
                        Expression_Transition = true;
                        break;
                    }


                    break;


//-----------------------------SLEEPY-----------------------------------
```

```
                case(sleepy):
                    IWDG->KR |= 0xAAAA;

                    if((lightLvl > 2) && (health < 5) && (health > 2)){ // Lights on ->
Sleepy to Content
                        state = content;
                        Play_Tone_4();
                        menu = ExpContent;
                        Eyelid_Move_To(1900); // open (content)
                        Expression_Transition = true;
                        break;
                    }

                    if((lightLvl > 2) && (health >= 5)){ // Lights on -> Sleepy to Happy
                        state = happy;
                        menu = ExpHappy;
                        Play_Tone_1();
                        Eyelid_Move_To(1500); // open
                        Expression_Transition = true;
                        break;
                    }

                    if((lightLvl > 2) && (health <= 2)){ // Lights on -> Sleepy to Hungry
                        state = hungry;
                        menu = ExpHungry;
                        Play_Tone_3();
                        Eyelid_Move_To(2200); //closed
                        Expression_Transition = true;
                        break;
                    }

                    if(health==0){     // check dead
                        state = dead;
                        menu = ExpSleepy;
                        Play_Tone_3();
                        Eyelid_Move_To(2570); // closed
                        Expression_Transition = true;
                        break;
                    }


                    break;


//------------------------------DEAD---------------------------------
                case(dead):
                    IWDG->KR |= 0xAAAA;

                    if(health>2){
                        state = happy;
                        menu = ExpHappy;
                        Play_Tone_1();
```

```
                        Eyelid_Move_To(1500); // open (HappY)
                        Expression_Transition = true;
                        break;
                    }


                    break;

//------------------------------MENU-------------------------------------
                case(MENU_STATE):


                    break;
            }

                MENU_SCREENS(); //change LCD



            /*DISTANCE TAIL WAG*/
            //sonar_gpio_init();

            distance = abs(dist()); //get the distance value

            if(distance < 400) {Mimic_Servo.is_close = true;}
            else{Mimic_Servo.is_close = false;}

            //check servo position

            /*CHECK LIGHT LEVEL*/
            lightLvl = Read_ADC();

            if(count != old_count){
                old_count = count;
                sevenSeg_write(0x05, num[count/10]);
                sevenSeg_write(0x04, num[count%10]);
            }
        CNT_New = CNT_Old;

        }
}


/*****************************************************************************

EXTI15_10_Handler
*****************************************************************************/
    void EXTI15_10_IRQHandler(void){

        if(EXTI->PR & 1<<HALL_PIN){
            test = 1;
            HealthPlusPlus();
```

```
            }


        if(EXTI->PR & 1<<HALL2_PIN){
                petFlag = 1;
                Play_Tone_1();
            }

        if(EXTI->PR & 1<<BTN1_PIN){

        }

        EXTI->PR |= (1<<HALL_PIN)|(1<<BTN1_PIN)|(1<<HALL2_PIN);
}




/*****************************************************************************
                                                                    EXTI4_Handler
*****************************************************************************/
    void EXTI4_IRQHandler(void){
        swPress ++;
        IWDG->KR |= 0xAAAA;

//      if(state == An_Expression_State){
//          IWDG->KR |= 0x5555;
//           IWDG->RLR |= 0x0; // max reload Value
//      }

        //***TIME SET MENU ****
        if(menu == timMenu){
                //used to display the correct value when moving cursor in menu
            if(swPress%4 == 0){
                    TIM2->CNT = pos*2; //keeps the current position value
            }else{
                    TIM2->CNT = value*2; //keeps current value
            }
        //confirm selected times
            if(pos == 13){
                menu_flag = 0;
            }
        //cancel menu select
            if(pos == 12){
                menu_flag = 0;
                RTC_Second = 0; //these should probably be the current value, not zero but
I'll have to change it afer more testing
                RTC_Minute = 0;
                RTC_Hour = 0;
                RTC_Date = 0;
                RTC_Month = 0;
```

```
                RTC_Year = 0;
            }
    }

        //****MAIN MENU*****
    if(menu == mainMenu){
        menu_flag = 0;
        if(pos == 0){
            menu = timMenu;
        }else if(pos == 1){
            menu = helpMenu;
        }else if(pos == 2){
            menu = infoMenu;
        }else{
            //THIS SET YOU BACK TO THE EXPRESSIONS,
            //it needs to be adjusted such that you go back to the
            //previous state not just the content state
            state = content;
            Eyelid_Move_To(1900);
            menu_flag = 0;
            menu = ExpContent;
        }
    }

    //****HELP MENU******
    if(menu == helpMenu){
        menu_flag = 0;
    }

    //****INFO MENU******
    if(menu == infoMenu){
        menu_flag = 0;
    }

    //**START UP SCREEN***   //not set up yet
    if(menu == startUpScreen){
        menu_flag = 0;
    }
    //**SETTINGS SCREEN***  //not set up yet
    if(menu == settings){
        menu_flag = 0;
    }

        EXTI->PR |= (1<<4);
}
```

```
/*=========================================================================================
// Set_Time

 * @brief: Prompts the user to set the registers of the RTC controlling the hours, minutes and
seconds
 *
 * @param[in] NULL
 *
 * @return: NULL
 =========================================================================================*/
void Set_Time(uint8_t RTC_Hour, uint8_t RTC_Minute, uint8_t RTC_Second){
/*/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\*/
    I2C1_byteWrite(SLAVE_ADDR,HOURS_ADDR,RTC_Hour);
    I2C1_byteWrite(SLAVE_ADDR,MINUTES_ADDR,RTC_Minute);
    I2C1_byteWrite(SLAVE_ADDR,SECONDS_ADDR,RTC_Second);
}


/*=========================================================================================
// Set Date

 * @brief: Prompts the user to set the registers of the RTC controlling the month, day, year
and date
 *
 * @param[in] NULL
 *
 * @return: NULL
 =========================================================================================*/
void Set_Date(uint8_t RTC_Year, uint8_t RTC_Month, uint8_t RTC_Date){

/*/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\*/
    I2C1_byteWrite(SLAVE_ADDR,YEAR_ADDR,RTC_Year);
    I2C1_byteWrite(SLAVE_ADDR,MONTH_ADDR,RTC_Month);
    I2C1_byteWrite(SLAVE_ADDR,DATE_ADDR,RTC_Date);
}




/*=========================================================================================
// READ Date

 * @brief: Prompts the user to set the registers of the RTC controlling the month, day, year
and date
 *
 * @param[in] NULL
 *
 * @return: NULL
 =========================================================================================*/
void Read_Date(void){
    I2C1_byteRead(SLAVE_ADDR, MONTH_ADDR, RTC_Data_ptr);
    Draw_Char_BG(140, 5, MSB(RTC_data), BROWN, WHITE, &font_ubuntu_mono_24);
```

```
    Draw_Char_BG(150, 5, LSB(RTC_data), BROWN, WHITE, &font_ubuntu_mono_24);
    Draw_Char_BG(160, 5, '/', BROWN, WHITE, &font_ubuntu_mono_24);
    //read the current min from the RTC, stores value in the RTC_data pointer
    I2C1_byteRead(SLAVE_ADDR, DATE_ADDR, RTC_Data_ptr);
    Draw_Char_BG(170, 5, MSB(RTC_data), BROWN, WHITE, &font_ubuntu_mono_24);
    Draw_Char_BG(180, 5, LSB(RTC_data), BROWN, WHITE, &font_ubuntu_mono_24);
    Draw_Char_BG(190, 5, '/', BROWN, WHITE, &font_ubuntu_mono_24);
    //read the current second from the RTC, stores value in the RTC_data pointer
    I2C1_byteRead(SLAVE_ADDR, YEAR_ADDR, RTC_Data_ptr);
    Draw_Char_BG(200, 5, MSB(RTC_data), BROWN, WHITE, &font_ubuntu_mono_24);
    Draw_Char_BG(210, 5, LSB(RTC_data), BROWN, WHITE, &font_ubuntu_mono_24);
}




/*=============================================================================================
// READ Date

 * @brief: Prompts the user to set the registers of the RTC controlling the month, day, year
and date
 *
 * @param[in] NULL
 *
 * @return: NULL
 ============================================================================================*/
void Read_Time(void){
    //read the current hour from the RTC, stores value in the RTC_data pointer
    I2C1_byteRead(SLAVE_ADDR, HOURS_ADDR, RTC_Data_ptr);
    Draw_Char_BG(10, 5, MSB(RTC_data), BROWN, WHITE, &font_ubuntu_mono_24);
    Draw_Char_BG(20, 5, LSB(RTC_data), BROWN, WHITE, &font_ubuntu_mono_24);
    Draw_Char_BG(30, 5, ':', BROWN, WHITE, &font_ubuntu_mono_24);
    //read the current min from the RTC, stores value in the RTC_data pointer
    I2C1_byteRead(SLAVE_ADDR, MINUTES_ADDR, RTC_Data_ptr);
    Draw_Char_BG(40, 5, MSB(RTC_data), BROWN, WHITE, &font_ubuntu_mono_24);
    Draw_Char_BG(50, 5, LSB(RTC_data), BROWN, WHITE, &font_ubuntu_mono_24);
    Draw_Char_BG(60, 5, ':', BROWN, WHITE, &font_ubuntu_mono_24);
    //read the current second from the RTC, stores value in the RTC_data pointer
    I2C1_byteRead(SLAVE_ADDR, SECONDS_ADDR, RTC_Data_ptr);
    Draw_Char_BG(70, 5, MSB(RTC_data), BROWN, WHITE, &font_ubuntu_mono_24);
    Draw_Char_BG(80, 5, LSB(RTC_data), BROWN, WHITE, &font_ubuntu_mono_24);
}




/*=============================================================================================
// Set Date and date

 * @brief: Prompts the user to set the registers of the RTC controlling the month, day, year
and date
```

```
 *
 * @param[in] NULL
 *
 * @return: NULL
 ============================================================================================*/
void Set_TD(void){
    /*FOR REFERENCE:
        -Sec        0,1
        -Min        2,3
        -hr         4,5
        -date       6,7
        -month  8,9
        -Year       10,11
        -Canel  12,
        -Enter  13

    */
    TIM2->ARR = 27;

    while(menu_flag){
        //set up position to change based on the interrupt from the rotary encoder
        //I want to set this such that when the btn on Rotary is pressed the interupt no longer
changes position but changes value
        //this should be the math to cahnge this, I have to do this instead of using a flag due
to the inturrtp counting twice

        if(swPress%4 == 0){
                tmp = pos;
                pos = TIM2->CNT/2;
                value = timeDate[pos];
            if(pos != tmp){
                if(tmp<=11){
                    Draw_Char_BG(180+((tmp%2)*20), 30+((tmp/2)*30), (timeDate[tmp] + '0'),
WHITE, BROWN, &font_ubuntu_mono_24);
                }else if(tmp == 12){
                    Draw_String_BG(30, 210, "Cancel", WHITE, BROWN, &font_ubuntu_mono_24);
                }else{
                    Draw_String_BG(160, 210, "Enter", WHITE, BROWN, &font_ubuntu_mono_24);
                }

            }
        }else{
                value = TIM2->CNT/2;
                timeDate[pos] = value;
        }

        //PRINTS THE CURRENT POSITION AND VALUE
        if(pos<=11){
            Draw_Char_BG(180+((pos%2)*20), 30+((pos/2)*30), (value + '0'), WHITE, GRAY,
&font_ubuntu_mono_24);
        }else if(pos == 12){
            Draw_String_BG(20, 210, "Cancel", WHITE, GRAY, &font_ubuntu_mono_24);
```

```
        }else{
            Draw_String_BG(180, 210, "Enter", WHITE, GRAY, &font_ubuntu_mono_24);
        }

    }

    RTC_Second = timeDate[0]<<4 | timeDate[1];
    RTC_Minute = timeDate[2]<<4 | timeDate[3];
    RTC_Hour = timeDate[4]<<4 | timeDate[5];
    RTC_Date = timeDate[6]<<4 | timeDate[7];
    RTC_Month = timeDate[8]<<4 | timeDate[9];
    RTC_Year = timeDate[10]<<4 | timeDate[11];

    Set_Time(RTC_Hour, RTC_Minute, RTC_Second);
    Set_Date(RTC_Year, RTC_Month, RTC_Date);

}




/*=============================================================================================
// MENU SCREENS

 * @brief: DISPLAYS DIFFERENT MENU OPTIONS AND SCREEN FOR DIFFERENT STATES IN THE PROGRAM
 *
 * @param[in] NULL
 *
 * @return: NULL
 ============================================================================================*/
void MENU_SCREENS(void){
    int temp = 0;
    /*
    HOW THE MENU SYSTEM WORKS:
    NOTE: the order is not always consistant
    Enter a menu screen,
    Menu flag is set, This keeps the user in the menu loop until an option is picked
    A white rectangle clears all old text,
    The count value and position (which corrisponds to the position of the encoder) is restart
(0)

    The TIM2->ARR gets addjusted to the amount of menu options that are in the state
    Example:
    main Menu has 5 options
    TIM2->ARR = (5*2)-1;

    While Loop()
    Displays the current poisiton of the rotary encoder
    When the switch is pressed an interupt is entered that sets menu flag
    kicking the user out of the screen and into another one
```

```
    */

                switch(menu){

//-------------------------MAIN MENU-----------------------------
                case mainMenu:
                    default:
                        //RESET VARIABLES
                            menu_flag = 1;
                            TIM2->CNT = 0;
                            pos = 0;
                            TIM2->ARR = 7; //set the number of options

                            //clear screen then read the date
                            Rotate_Display(0);
                            Fill_Screen(BROWN); //Moves sp l to R, moves sp t to b /*NOTE
THIS IS JUST TO CLEAR THE OLD TXT THERE, while keep the time at top*/
                            Read_Date();

                        //display options
                            Draw_String_BG(50, 80, "Time Set:", WHITE, BROWN,
&font_ubuntu_mono_24);
                            Draw_String_BG(50, 120, "Help:", WHITE, BROWN,
&font_ubuntu_mono_24);
                            Draw_String_BG(50, 160, "More Info:", WHITE, BROWN,
&font_ubuntu_mono_24);
                            Draw_String_BG(50, 200, "Cancel", WHITE, BROWN,
&font_ubuntu_mono_24);

                            while(menu_flag){
                                Read_Time();
                                temp = pos;
                                pos = TIM2->CNT/2; //set position

                                if(temp == pos){
                                    Draw_String_BG(30, 80+(pos*40), " ", GRAY, GRAY,
&font_ubuntu_mono_24);
                                }else{
                                    Draw_String_BG(30, 80+(temp*40), " ", BROWN, BROWN,
&font_ubuntu_mono_24);
                                }


                            }
                break;


//--------------------TIME SET MENU--------------------------------
                case timMenu:


                    menu_flag = 1;
```

```
                            Fill_Rect(12, 30, 288, 226, BROWN);
                            Draw_String_BG(20, 30, "Set Second:", WHITE, BROWN,
&font_ubuntu_mono_24);
                            Draw_String_BG(20, 60, "Set Minute:", WHITE, BROWN,
&font_ubuntu_mono_24);
                            Draw_String_BG(20, 90, "Set Hour:", WHITE, BROWN,
&font_ubuntu_mono_24);
                            Draw_String_BG(20, 120, "Set date:", WHITE, BROWN,
&font_ubuntu_mono_24);
                            Draw_String_BG(20, 150, "Set Month:", WHITE, BROWN,
&font_ubuntu_mono_24);
                            Draw_String_BG(20, 180, "Set Year:", WHITE, BROWN,
&font_ubuntu_mono_24);
                            Draw_String_BG(20, 210, "Cancel", WHITE, BROWN,
&font_ubuntu_mono_24);
                            Draw_String_BG(180, 210, "Enter", WHITE, BROWN,
&font_ubuntu_mono_24);

                            Draw_String_BG(180, 30, "0", WHITE, BROWN,
&font_ubuntu_mono_24);
                            Draw_String_BG(180, 60, "0", WHITE, BROWN,
&font_ubuntu_mono_24);
                            Draw_String_BG(180, 90, "0", WHITE, BROWN,
&font_ubuntu_mono_24);
                            Draw_String_BG(180, 120, "0", WHITE, BROWN,
&font_ubuntu_mono_24);
                            Draw_String_BG(180, 150, "0", WHITE, BROWN,
&font_ubuntu_mono_24);
                            Draw_String_BG(180, 180, "0", WHITE, BROWN,
&font_ubuntu_mono_24);

                            Draw_String_BG(200, 30, "0", WHITE, BROWN,
&font_ubuntu_mono_24);
                            Draw_String_BG(200, 60, "0", WHITE, BROWN,
&font_ubuntu_mono_24);
                            Draw_String_BG(200, 90, "0", WHITE, BROWN,
&font_ubuntu_mono_24);
                            Draw_String_BG(200, 120, "0", WHITE, BROWN,
&font_ubuntu_mono_24);
                            Draw_String_BG(200, 150, "0", WHITE, BROWN,
&font_ubuntu_mono_24);
                            Draw_String_BG(200, 180, "0", WHITE, BROWN,
&font_ubuntu_mono_24);

                            Set_TD();
                menu = mainMenu;
                menu_flag = 1;


                    break;

//------------------------------HELP MENU-----------------------------------------
```

```
                case helpMenu:

                    Read_Date();
                    menu_flag = 1;
                    TIM2->ARR = 3;
                    Fill_Screen(BROWN);
                            Draw_String_BG(20, 120, "HELP MENU SCREEN", WHITE, BROWN,
&font_ubuntu_mono_24);
                            Draw_String_BG(20, 160, "Press to contiune", WHITE, BROWN,
&font_ubuntu_mono_24);
                        while(menu_flag){
                            Read_Time();
                        }
                        menu = mainMenu;
                break;


//-----------------------------------INFO MENU---------------------------------------
                case infoMenu:
                    Read_Date();
                    menu_flag = 1;
                        Fill_Screen(BROWN);
                        Draw_String_BG(20, 120, "INFO MENU SCREEN:", WHITE, BROWN,
&font_ubuntu_mono_24);
                        Draw_String_BG(20, 160, "Press to contiune", WHITE, BROWN,
&font_ubuntu_mono_24);
                    while(menu_flag){
                            Read_Time();
                        }
                        menu = mainMenu;
                break;


//-----------------------------------START UP SCREEN-----------------------------------
                case startUpScreen:
                    menu_flag = 1;
                    Fill_Screen(BROWN);
                while(menu_flag){}
                menu = mainMenu;
                break;


//---------------------------------------SETTINGS-----------------------------------------
                case settings:
                    menu_flag = 1;
                    TIM2->ARR = 3;
                                TIM2->CNT = 0;
                                pos = 0;
                    Fill_Screen(BROWN);
                    Draw_String_BG(20, 80, "Open Settings?", WHITE, BROWN,
&font_ubuntu_mono_24);
                    while(menu_flag){
```

```
                    Draw_String_BG(20, 120, "Yes", WHITE, BROWN, &font_ubuntu_mono_24);
                }
                break;


//---------------------------------------CONTENT---------------------------------------
            case ExpContent:
                if(Expression_Transition == true){
                    Rotate_Display(0);
                    Fill_Screen(BROWN);
                    Draw_Bitmap((TFT_WIDTH - contentImage->width) / 2, (TFT_HEIGHT -
contentImage->height) / 2, contentImage); // Displays a scaled image of an apple. (Small to
maintain 32kB flash limit on Keil)
                    Expression_Transition = false;
                }
            break;


//---------------------------------------HAPPY---------------------------------------
            case ExpHappy:
                if(Expression_Transition == true){
                    Rotate_Display(0);
                    Fill_Screen(BROWN);
                    Draw_Bitmap((TFT_WIDTH - HappyImage->width) / 2, (TFT_HEIGHT -
HappyImage->height) / 2, HappyImage); // Displays a scaled image of an apple. (Small to
maintain 32kB flash limit on Keil)
                    Expression_Transition = false;
                }
                break;


//---------------------------------------HUNGRY---------------------------------------
            case ExpHungry:
                if(Expression_Transition == true){
                    Rotate_Display(0);
                    Fill_Screen(BROWN);
                    Draw_Bitmap((TFT_WIDTH - HungryImage->width) / 2, (TFT_HEIGHT -
HungryImage->height) / 2, HungryImage); // Displays a scaled image of an apple. (Small to
maintain 32kB flash limit on Keil)
                    Expression_Transition = false;
                }
                break;


//---------------------------------------SLEEPY---------------------------------------
            case ExpSleepy:
                if(Expression_Transition == true){
                    Rotate_Display(0);
                    Fill_Screen(BROWN);
                    Draw_Bitmap((TFT_WIDTH - SleepyImage->width) / 2, (TFT_HEIGHT -
SleepyImage->height) / 2, SleepyImage); // Displays a scaled image of an apple. (Small to
```

```
maintain 32kB flash limit on Keil)
                        Expression_Transition = false;
                    }
                    break;

            }

}


void TIM5_IRQHandler(void)
{
    random_seconds_counter++;
    if(state != MENU_STATE){
        count --;
    if(count <= 0){
        count = 60;
        HealthMinusMinus();


        Store_Current_Health();

        }
    }

    tongue_wag = (tongue_wag)? 0:1;

        if(Mimic_Servo.is_close || petFlag){
            Tongue_Move_To((tongue_wag)? 1000:2000);
        }

    if(random_seconds_counter == random_seconds){
        random_seconds_counter = 0;
        Eye_Move_To(rand()%((1800-1000)+1)+1000 );
        random_seconds = rand()%(8)+1;
    }

    TIM5->SR &= ~0x0001U;
}

/// @brief function just to store health in the RTC
void Store_Current_Health(void){
    I2C1_byteWrite(SLAVE_ADDR,HEALTH_ADDR,health);
}

/// @brief fucntion just to retrieve health from the EEPROM
void Retrieve_Health(void){
    I2C1_byteRead(SLAVE_ADDR,HEALTH_ADDR, health_ptr);
}

/// @brief Initialize Watchdog Timer
void WDT_init(void){
```

```
    IWDG->KR |= 0x5555; // Write to Access PR and RLR
    IWDG->PR |= 0x6; // Prescalar max timeout 32768ms
    IWDG->RLR |= 0xFFF; // max reload Value
    IWDG->KR |= 0xCCCC; // Write to Start Watchdog Timer
}

/// @brief Reset Watchdog Timer
void WDT_reset(void){
    IWDG->KR |= 0xAAAA; // Write to Reset Counter
}



// EOF
```