UNLICENSED TI-OS WORKSTATION

Operating System Version 3.5

Logan Field and Kaushal Marimuthu

1 Introduction

TI-OS is an embedded systems project that aims to replicate the user experience of a personal computer through the hardware of the TI CC3200. Through the management of real-time graphics, application switching, and a simplified physical interface, the "closed loop" intuitive behavior of a more complex operating system is accomplished on the limited hardware of the microcontroller.

2 Design

2.1 Features

- Hierarchical Desktop-based application scheme
- Cursor Based Navigation
- Physical control interface
  - Analog Joystick
  - Two Push Buttons
  - Buzzer Speaker
  - BMA222 Accelerometer
  - 128x128 OLED Screen
  - SF006C Servo-controlled helping hand
- Breadth of utility and entertainment applications
  - Function Generator (signal generation)
  - Oscilloscope (waveform analysis)
  - AWS IoT (cloud connectivity)
  - Video Game (entertainment)
  - 3D Cube (accelerometer-controlled graphics)
  - Servo Control (motor control)

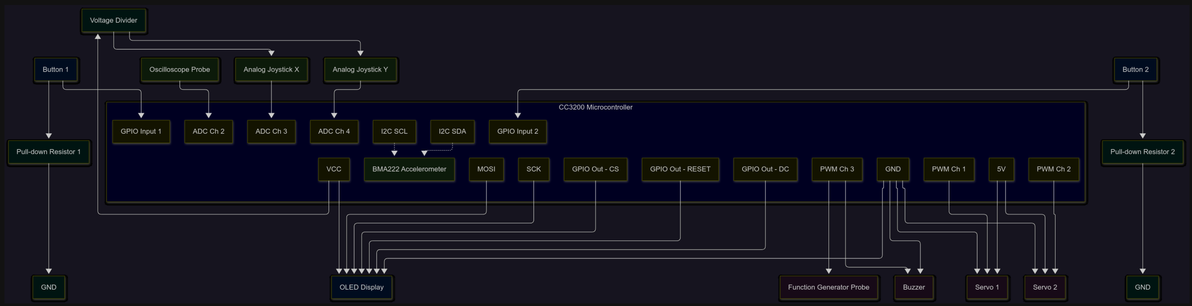


Figure 1: CC3200 System Component Connection Diagram

2.2 System Control Flow

The high level state management of the system occurs in the main.c code, which also functions as the desktop control code. The desktop is the ”central hub” of the program, and when a user selects an icon, state changes within the main code are used to instruct function calls of that applications respective RunFrame() continuously until the exit condition is satisfied.

- Hardware Initialization (Power On → System Ready)
  - Board Initialization
  - Hardware Interface Configuration
- Introduction Sequence (System Ready → Animation Complete)
  - Animated startup sequence playback
  - Introductory sound effects
  - Automatic transition to main menu
- Option Screen Navigation (Main Menu Loop)
  - Menu display rendering
  - Joystick input processing
  - Cursor hitbox detection
  - Real-time cursor position updates
  - Audio feedback for user interactions
- Application Execution (Button 1 + Option Selected → Back)
  - Application initialization and resource allocation
  - Dedicated application run frame loop
  - Real-time graphics and input processing
  - Cleanup and return to main menu on exit

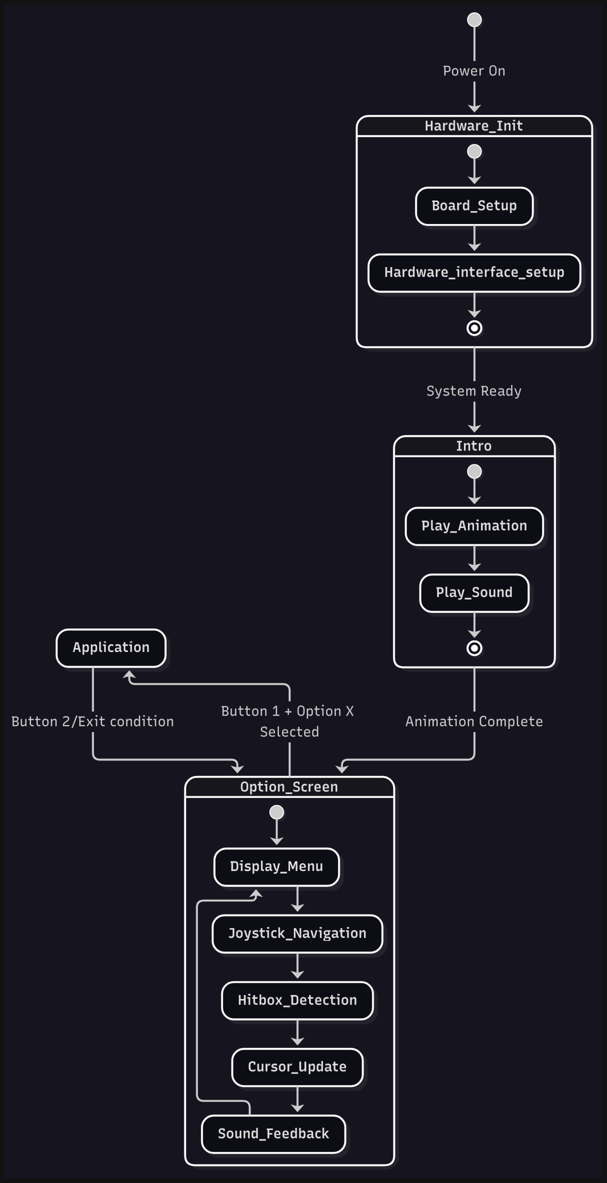
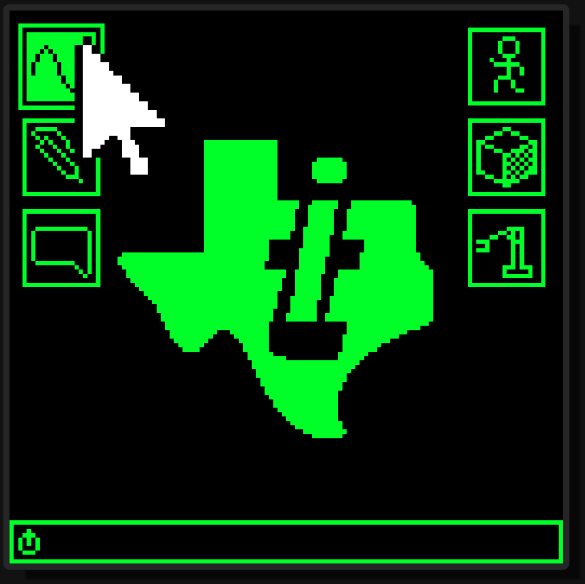


Figure 2: High Level State Diagram of Desktop

3 Desktop interface



3.1 Main Loop

Listing 1: located in main.c

```
#include <stdio.h>
void main(void)
{
    GameState gameState;

    /* Initialize hardware components
       */
    initializeHardware();

    /* Initialize SimpleLink */
    init_simplelink();

    /* Initialize game state */
    initializeGameState(&gameState);

    /* Main application loop */
    while (FOREVER) {
        /* Update sound effects */
        UpdateSoundEffects();

        /* Process input */
        processInput(&gameState);

        /* Update game logic */
        updateGameLogic(&gameState);

        /* Render the current interface
           */
        renderInterface(&gameState);
    }
}
```

The system initializes the hardware components and the wireless connectivity with Simplelink, then enters an infinite loop to continuously process input, update game logic, and render the user interface. The state-based parameters along with separation of input processing, rendering, and logic determining steps allow for a multitude of complex tasks to be accomplished while maintaining clean code flow.

3.2 Input Processing

Listing 2: located in main.c

```
\subsubsection{Button Handler}
static void ButtonHandler(void)
{
    unsigned long ulStatus;
    bool button1State, button2State;

    /* Clear the interrupt */
    ulStatus = GPIOIntStatus(
       BUTTON1_PORT, true);
    GPIOIntClear(BUTTON1_PORT, ulStatus
       );

    /* Read current button states (
       active low) */
    button1State = !(GPIOPinRead(
       BUTTON1_PORT, BUTTON1_PIN) == 0)
       ;
    button2State = !(GPIOPinRead(
       BUTTON2_PORT, BUTTON2_PIN) == 0)
       ;

    /* Update global state */
    if (button1State) {
        buttonHeld = true;
        currentButton = 1;
        screenNeedsUpdate = true;
    } else if (button2State) {
        buttonHeld = true;
        currentButton = 2;
        screenNeedsUpdate = true;
    } else {
        buttonHeld = false;
        currentButton = 0;
        screenNeedsUpdate = true;
    }
}
```

An interrupt service routine is used to process user input for the two buttons. First the interrupt is cleared, then both button states are immediately read, and translated into global state variables and flags that are monitored by functions called in the main loop. The screenNeedsUpdate flag is used to only render the display when necessary, and is used across the main code to prevent drawing artifacts such as flickering.

3.2.1 Reading Joystick

Listing 3: located in main.c

```
/* Read ADC values */
ReadADCChannel(ADC_CH_0, &voltage_57);
ReadADCChannel(ADC_CH_1, &voltage_58);
ReadADCChannel(ADC_CH_2, &voltage_59);
ReadADCChannel(ADC_CH_3, &voltage_60);

/* Calculate proportional offset from
    center */
xOffset = ((voltage_59 /
    ADC_REFERENCE_VOLTAGE) - 0.5f);
yOffset = ((voltage_60 /
    ADC_REFERENCE_VOLTAGE) - 0.5f);

/* Apply deadzone to prevent drift */
if(fabs(xOffset) >= JOYSTICK_DEADZONE)
    {
     state->cursorx -= xOffset * state->
         cursorSensitivity;
}

if(fabs(yOffset) >= JOYSTICK_DEADZONE)
    {
     state->cursory += yOffset * state->
         cursorSensitivity;
}
```

Cursor control is acheived through using the CC3200's ADC to read the joystick's position. First, normalized offsets are calculated based on experimentally determined reference voltages, then compared against a deadzone constant to avoid stick drift. Then the cursor position is updated in the x and y direction based on their respective offset. The use of cursorSensitivity allows for the adjustment of the speed of the cursor while maintaining continuous analog control.

3.3 Application Entry and State management

3.3.1 Hitbox Selection System

Listing 4: located in main.c

```
void updateGameLogic(GameState* state)
{
    /* Handle hitbox detection for menu
        selections */
    if(strcmp(state->currentInterface,
        "optionScreen") == 0) {
        if (checkHitbox(state->cursorx,
            state->cursory, 0, 0, 21,
           22)) {
            state->selectedOption = 1;
            state->
               optionBackgroundFrame =
               0;
            state->cursorFrame = 1;
...
```

The action of selecting a desktop icon and entering an application is accomplished through this system. When the game state is in the optionScreen, the position of the cursor is checked against the hitbox of each application's icon. When held over an icon, the selectedOption is updated for later use in application entry, the desktop bitmap frame is updated to one with the icon highlighted, and the cursorFrame is changed from a mouse to pointer icon to communicate that an interaction is available.

3.3.2 Application Entry

Listing 5: located in main.c

```
void handleButtonPress(GameState* state
    )
{
    if (currentButton == 1) {
        PlayButtonSound();
        Report("\n\r(Button␣1)");
        state->cursorFrame = 2;

if (strcmp(state->currentInterface, "
    optionScreen") == 0) {
    switch (state->selectedOption) {
        case 1:
            fastFillScreen(BLACK);
            state->currentInterface = "
               Function␣Generator";
            state->hideCursor = true;
            FunctionGenerator_Initialize
               ();
            break;
        case 2:
            fastFillScreen(BLACK);
            state->currentInterface = "
               Oscilloscope";
            state->hideCursor = true;
            Oscilloscope_Initialize();
            break;
...
```

This is where the selectedOption attribute is utilized. When the global variable for currentButton is updated, the function checks to see what the current selected option is, then preforms a state transation. The display is cleared, the currentInterface is updated, cursor hidden, and the application's respective initialization function is called. It is worth mentioning that this function could be optimized by combining many of the shared steps for state transitions above the switch statement, but earlier versions of the code required more unique steps for each state's entry.

3.3.3 Application Exit and Cleanup Management

Listing 6: located in main.c

```
if (strcmp(state->currentInterface, "3D
    ␣Cube") == 0) {
     if (button2State) {
         Cube3D_Cleanup();
         state->currentInterface = "
             optionScreen";
         state->hideCursor = false;
         screenNeedsUpdate = false;
     } else {
         Cube3D_RunFrame();
     }
} else if (strcmp(state->
    currentInterface, "Video␣Game") ==
    0) {
     if (!VideoGame_RunFrame()) {
         if (videogameInitialized) {
             videogameInitialized =
                 false;
         }
         state->currentInterface = "
             optionScreen";
         state->hideCursor = false;
         screenNeedsUpdate = false;
     }
}
...
```

The exit of the function is accomplished by largely similar but slightly different strategies among applications. The shared functionality of function exit is the transition of the currentInterface to the option screen, and resetting the screenNeedsUpdate and hideCursor flags. The exiting functionality is implemented in the application frame rendering function so that if-else statements can be used to avoid the state conflicts of rendering and exiting an application at the same time. Some of the applications exit by checking the return value of their RunFrame function. This is done for applications where a non-traditional exit condition may be required, for example, the AWS application exits by pressing buttons 1 and 2 simultaneously.

4 Handling and Storing Bitmaps

4.1 Generating Bitmaps Using a Python Helper Program

Listing 7: located in image bitmap generator.py

```
# Convert image data to bitmap format
for y in range(output_height):
    for x in range(output_width):
        # If the pixel is white (255),
            set the corresponding bit to
             1
        if img.getpixel((x, y)) > 0:
            byte_index = y *
                bytes_per_row + x // 8
            bit_position = 7 - (x % 8)
                 # MSB first (0x80 >> (i
                 & 7))
            bitmap_data[byte_index] |=
                (1 << bit_position)

# Generate C array declaration if
   requested
c_array = None
if output_c_file or output_c_file == ""
   :
    # Extract filename without
        extension for variable name
    var_name = os.path.splitext(os.path
        .basename(image_path))[0]
    var_name = ''.join(c if c.isalnum()
         else '_' for c in var_name)

    # Generate the C array declaration
    c_array = f"//␣Monochrome␣bitmap␣
        data␣for␣{os.path.basename(
        image_path)}\n"
    c_array += f"//␣Size:␣{output_width
        }x{output_height}␣pixels\n"
    c_array += f"#define␣{var_name.
        upper()}_WIDTH␣{output_width}\n"
    c_array += f"#define␣{var_name.
        upper()}_HEIGHT␣{output_height}\
        n\n"
    c_array += f"const␣uint8_t␣{
        var_name}Bitmap[]␣=␣{{\n␣␣␣␣"

    # Add the bitmap data in hex format
    for i, byte in enumerate(
        bitmap_data):
         c_array += f"0x{byte:02X},␣"
         if (i + 1) % 12 == 0:  # 12
             values per line
              c_array += "\n␣␣␣␣"

    c_array += "\n};"

    # Write to file if a path was
        provided
    if output_c_file != "":
        with open(output_c_file, 'w')
            as f:
             f.write(c_array)
```

The Bitmap Generator Python program allows for the rapid production of bitmaps for the program by importing gifs and using bit-packing logic to convert individual pixels into compressed byte-formatted bitmaps. It also generates headers that can be directly compiled for testing the display of said bitmaps, formatting the arrays with constant definitions for width and height as well as a readable 12-hex value.

4.2 Converting Bitmap Arrays to .bin files

Listing 8: located in bitmap converter.py

```
# Extract each frame by tracking brace
    levels
frames = []
current_frame = ""
brace_level = 0
in_frame = False

for char in array_content:
    if char == '{':
        brace_level += 1
        if brace_level == 1:  # Start
            of a new frame
             in_frame = True
             current_frame = ""
    elif char == '}':
        if in_frame and brace_level ==
            1:  # End of a frame
             in_frame = False

             # Process the collected
                 frame data
             hex_values = re.findall(r'0
                 x[0-9A-Fa-f]+',
                 current_frame)
             if not hex_values:
                  # Try decimal values
                  hex_values = re.findall
                      (r'\b\d+\b',
                      current_frame)

             # Convert to bytes
             try:
                  frame_bytes = bytes([
                      int(val, 0) if val.
                      startswith('0x')
                      else int(val) for
                      val in hex_values])
                  frames.append(
                      frame_bytes)
                  print(f"Extracted␣frame
                      ␣{len(frames)},␣byte
                      ␣length:␣{len(
                      frame_bytes)}")
             except Exception as e:
                  print(f"Error␣
                      processing␣frame:␣{e
                      }")

        brace_level -= 1

    if in_frame:
        current_frame += char

return array_name, frames

def save_frames_to_binary(array_name,
    frames, output_dir):
    """Save␣extracted␣frames␣to␣binary␣
        files."""
    os.makedirs(output_dir, exist_ok=
        True)

    # Save each frame to a separate
        file
    for i, frame in enumerate(frames):
        filename = f"{array_name}_{i}.
            bin"
        filepath = os.path.join(
            output_dir, filename)

        with open(filepath, 'wb') as f:
             f.write(frame)

        print(f"Saved␣frame␣{i}␣to␣{
            filepath}␣({len(frame)}␣
            bytes)")
```

After testing and modifying the bitmaps created with the previous helper program, this python program is used to translate the headers into a format that can be directly stored in the file system through flashing. This offloads the large arrays from the SRAM, allowing many bitmaps to be used throughout the program while still remaining below the maximum size for writing to the board.

4.3 Flashing to File System and Reading in Program

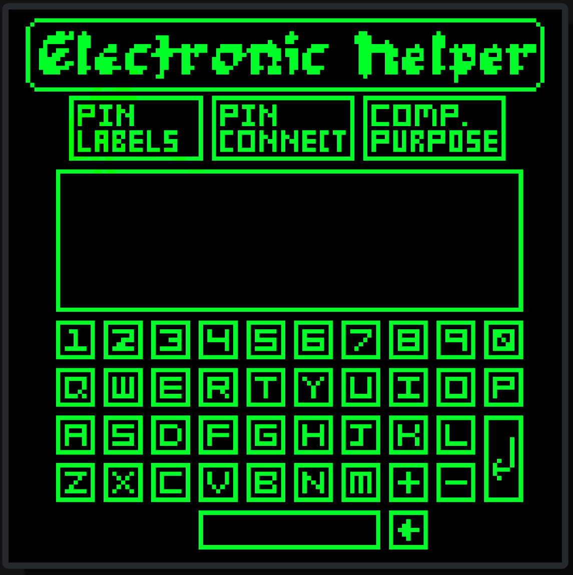
Listing 9: located in optionBackground bitmap.h

```

#include "simplelink.h"
#include <string.h>

#define OPTIONBACKGROUND_WIDTH 128
#define OPTIONBACKGROUND_HEIGHT 128
```

```
#define OPTIONBACKGROUND_FRAME_COUNT 8
#define OPTIONBACKGROUND_FRAME_SIZE
    2048  // Size of each frame in bytes

// Function to get a pointer to a
    specific frame
const uint8_t*
    get_optionBackground_frame(uint16_t
    frame_index) {
     static uint8_t frameBuffer[
        OPTIONBACKGROUND_FRAME_SIZE];

     // Ensure valid frame index
     if (frame_index >=
        OPTIONBACKGROUND_FRAME_COUNT) {
         frame_index = 0;
     }

     // Format filename
     char filename[64];
     sprintf(filename, "/
        optionBackgroundFrames_%d.bin",
        frame_index);

     // Try opening the file - cast to
        unsigned char*
     long fileHandle;
     int status = sl_FsOpen((unsigned
        char*)filename,
        FS_MODE_OPEN_READ, NULL, &
        fileHandle);

     if (status < 0) {
         // Default pattern
         memset(frameBuffer, 0,
            OPTIONBACKGROUND_FRAME_SIZE)
            ;
         frameBuffer[3] = 0x08;  //
            Default dot
         return frameBuffer;
     }

     // Read data
     long bytesRead = sl_FsRead(
        fileHandle, 0, frameBuffer,
        OPTIONBACKGROUND_FRAME_SIZE);
     sl_FsClose(fileHandle, 0, 0, 0);

     return frameBuffer;
}
```

A version of this function is included for every bitmap animation included in the program. These are saved as header files for use as includes in the programs that need to utilize their animations, and include helper functions and constants for displaying the bitmaps for individual frames. The construction of file names based on the requested frame along with a consistent labeling system for frames in the file system allows a single-function call usage when implementing animations. This is another point where programming optimization is possible. These bitmap helpers are all saved as individual helper functions, and their identical format and functionality would make it trivial to combine them all into a single object-based header. This was not done for this project because the development cycle required frequent debugging of bitmaps, and isolating them to individual headers allowed easier identification of naming and pathing errors.

5 AWS Implementation (Electronic Helper)



This component implements the “Electronic Helper,” an integration of AWS IoT and OpenAI API in TI-OS. It contains three modes: Pin Labels to describe the pins that a component has, Pin Connect to describe the connections that must be made between two components, and Component Purpose to describe what a component is used for. It manages data flow from TI-OS to AWS IoT and OpenAI API for processing, using a device shadow for state synchronization.

5.1 Information Flow

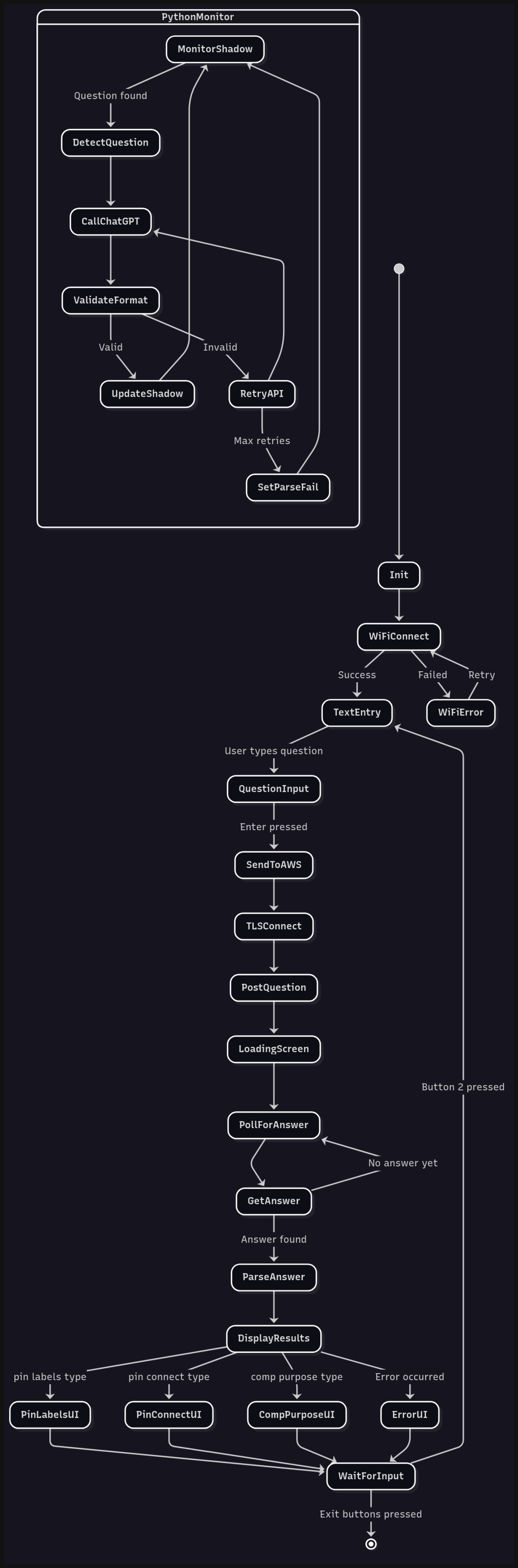


Figure 3: State Diagram of Electronic Helper

5.2 Helper Python Program

5.2.1 AWS IoT Client Initialization

Listing 10: located in iot shadow client.py

```
class IoTShadowClient:
    def __init__(self, endpoint, region
        ="us-east-1"):
        """Initialize the IoT Shadow
            client with AWS endpoint and
            region."""
        self.endpoint = endpoint
        self.region = region
        self.service = "iotdata"

        # Get credentials from the
            default AWS profile
        try:
            session = boto3.Session()
            self.credentials = session.
                get_credentials()
            if self.credentials is None
                :
                sys.exit("Error:␣No␣AWS
                    ␣credentials␣found.␣
                    Configure␣AWS␣CLI␣or
                    ␣set␣environment␣
                    variables.")
        except Exception as e:
            sys.exit(f"Error␣
                initializing␣AWS␣session
                :␣{e}")
```

This constructor is used to set up the AWS connection service using the endpoint and region, and uses the credentials to initialize the service. If that fails, it exits with an error.

5.2.2 Core Shadow Operations

Listing 11: located in iot shadow client.py

```
def _send_request(self, method,
    thing_name, payload=None):
    """Send a signed request to the AWS
        IoT API."""
    url = f"https://{self.endpoint}/
        things/{thing_name}/shadow"

    # Create and sign the request
    request = AWSRequest(method=method,
        url=url, data=json.dumps(
        payload) if payload else None)
    request.headers.add_header("Content
        -Type", "application/json")
        SigV4Auth(self.credentials, self.
            service, self.region).add_auth(
            request)

        # Send the request
        prepared_request = request.prepare
            ()
        response = requests.request(method=
            prepared_request.method...
            further params)
        response.raise_for_status()

        try:
            return response.json()
        except json.JSONDecodeError:
            return {"status": response.
                status_code, "text":
                response.text}

    def get_shadow(self, thing_name):
        return self._send_request("GET",
            thing_name)

    def update_shadow(self, thing_name,
        state):
        if "state" not in state:
            payload = {"state": state}
        else:
            payload = state
        return self._send_request("POST",
            thing_name, payload)

    def delete_shadow(self, thing_name):
        return self._send_request("DELETE",
            thing_name)
```

These functions implement the default CRUD routes for the IoT Device Shadow via HTTP Requests. This allows the program to get, update, and delete the device shadow's state.

5.2.3 OpenAI API Integration

Listing 12: located in iot shadow client.py

```
def get_chatgpt_response(self, question
    , api_key, model="gpt-4.1",
    question_type=None):
    """Get a response from ChatGPT API
        for the given question."""
    url = "https://api.openai.com/v1/
        chat/completions"

    headers = {
        "Content-Type": "application/
            json",
        "Authorization": f"Bearer␣{
            api_key}"
    }

    system_prompt = self.
        get_system_prompt(question_type)
    data = {
        "model": model,
        "messages": [
            {"role": "system", "content
                ": system_prompt},
            {"role": "user", "content":
                question}
        ],
        "max_tokens": 150
    }

    response = requests.post(url,
        headers=headers, json=data,
        timeout=30)
    response.raise_for_status()
    result = response.json()

    return result["choices"][0]["
        message"]["content"].strip()
```

This is where the program communicates with OpenAI API. The program is able to take the input from the shadow state as the users question, and send a POST request to OpenAI. This gives the generated answer to the question and gets returned.

5.2.4 Question Type Processing

Listing 13: located in iot shadow client.py

```
def parse_question_type(self, question)
    :
    """Parse the question to determine
        its type and extract the actual
        question."""
    question_types = {
        "pin␣labels/": "pin_labels",
        "pin␣connect/": "pin_connect",
        "comp␣purpose/": "comp_purpose"
    }

    for prefix, question_type in
        question_types.items():
        if question.startswith(prefix):
            actual_question = question[
                len(prefix):]
            return question_type,
                actual_question

    return None, question
```

```
def get_system_prompt(self,
    question_type):
    """Get the appropriate system
        prompt based on question type.
        """
    prompts = {
        "pin_labels": "You␣are␣acting␣
            as␣a␣mediating␣step␣in␣a␣
            component␣descriptor␣program
            ...",
        "pin_connect": "You␣are␣acting␣
            as␣a␣mediating␣step␣in␣a␣
            component␣descriptor␣program
            ...",
        "comp_purpose": "You␣are␣acting
            ␣as␣a␣mediating␣step␣in␣a␣
            component␣descriptor␣program
            ..."
    }

    return prompts.get(question_type, "
        ")
```

This is the pre-processing done for the incoming questions. Parse_question_type strips off the prefix in the question, splitting the question into the type and the actual question. Get_system_prompt is engineered such that different question types can have different prompts.

Prompts used for each question type: **Pin Labels**: You are acting as a mediating step in a component descriptor program. In this instruction there will be a component number. Please provide the number of pins as well as each pins label in as few letters as possible. Use the format [number of pins:] [1:label, 2:label, 3:label]. There should be no characters preceding or following the outlined format. Use the format exactly: make sure to include the [number of pins:] label. Do not use more than 3 letters for the labels.

Pin Connect: You are acting as a mediating step in a component descriptor program. In this instruction there will be a component, and something the user wants to connect to. This will be in the format: component + component to connect. Please provide information about how to connect this component in the format [pin (pin name component 1) + pin (pin name component 2)]. Keep pin names less than 5 characters. For example: [pin 1 (VCC) + pin 2 (V+), pin 2 (GND) + pin 3 (GND), pin 3 (MOSI) + pin 6 (GPIO1)]. It is very important that you follow this format exactly, with absolutely no extra text preceding, following or missing. You must use integer numbers for pin . Do not use the format pin AD0 (TCK)

Component Purpose: You are acting as a mediating step in a component descriptor program. In this instruction there will be a component number. Please provide an explanation of what the component does in 90 characters or less. Use abbreviation and shortened language if needed.

5.2.5 Response Format Validation

Listing 14: located in iot shadow client.py

```
def pin_connection_format_check(self,
    answer):
    """Check if pin connection answer
        follows exact required format.
        """
    # Expected: "[pin # (name) + pin #
        (name), pin # (name) + pin # (
        name), ...]"

    if not (answer.startswith('[') and
        answer.endswith(']')):
        return False

    inner_content = answer[1:-1].strip
        ()
    connections = [conn.strip() for
        conn in inner_content.split(',')
        ]

    for connection in connections:
        if not self.
            _validate_single_connection(
            connection):
            return False

    return True

def pin_label_format_check(self, answer
    ):
    """Check if pin label answer
        follows exact required format.
        """
    # Expected: "[number of pins: X]
        [1:label, 2:label, 3:label,
        ...]"

    pattern = r'^\[number␣of␣pins:␣(\d
        +)\]␣\[(\d+:[^,\]]+(?:,␣\d
        +:[^,\]]+)*)\]$'
    match = re.match(pattern, answer.
        strip())

    if not match:
        return False

    # Validate pin count matches
        declared count and sequential
        numbering
    # ... validation logic ...
    return True
```

These are helper functions to check the format of a GPT API response before updating the shadow state. This way, the format follows an exact standard and can be parsed by TI-OS. Pin connection formats must be pin (name) + pin (name), and pin labels must match a specific Regex pattern for ([number of pins: X] [1:label, 2:label, 3:label, ...]).

5.2.6 Retry Logic with Error Handling

Listing 15: located in iot shadow client.py

```
def get_chatgpt_response_with_retry(
    self, question, api_key, model="gpt
    -4.1"... further params):
"""Get ChatGPT response with format
    checking and retry logic."""

for attempt in range(max_retries):
    try:
        answer = self.
            get_chatgpt_response(
            question, api_key, model,
            question_type)

        # Check format based on
            question type
        format_check_passed = True
        if question_type == "pin_labels
            ":
             format_check_passed = self.
                 pin_label_format_check(
                 answer)
        elif question_type == "
            pin_connect":
             format_check_passed = self.
                 pin_connection_format_check
                 (answer)

        if format_check_passed:
             return answer
        else:
             if attempt < max_retries -
                 1:
                  # Update shadow with "
                      wait" status before
                      retrying
                  self.update_shadow(
                      thing_name, {"state"
                      : {"desired": {"
                      Answer": "wait", "
                      var": "wait"}}})
                  time.sleep(2)
                  continue
             else:
                  # Final failure -
                      update with "parse
                      fail"
                  self.update_shadow(
                      thing_name, {"state"
                      : {"desired": {"
                      Answer": "parse␣fail
                      ", "var": "parse␣
                      fail"}}})
                  return "parse␣fail"

    except Exception as e:
        # Handle API errors with retry
            logic
        # ... error handling ...
```

This function continually prompts GPT until either it uses its max entries or it gets a correctly formatted response, and updates the device shadow with the result. While waiting for responses, it keeps the shadow state on ”wait”.

5.3 Core Question-Answer Workflow

Listing 16: located in text entry.c

```
static void on_enter_pressed(const char
    * question) {
    UART_PRINT("===␣ENTER␣KEY␣PRESSED␣
        ===\n\r");
    UART_PRINT("Question␣from␣text␣
        entry:␣%s\n\r", question);

    Clear_toggle();
    const uint8_t* loading_frame_bitmap
        = get_loading_screen_frame(0);
    fastDrawBitmap(0, 0,
        loading_frame_bitmap, 128, 128,
        GREEN, BLACK, 1);

    strncpy(g_current_question,
        question, sizeof(
        g_current_question) - 1);
    g_current_question[sizeof(
        g_current_question) - 1] = '\0';

    if (!g_wifi_connected) {
        strcpy(g_current_answer, "WiFi␣
            not␣connected");
        return;
    }

    strcpy(g_current_answer, "Sending␣
        to␣AWS...");
    int socketId = tls_connect_aws();
    if(socketId >= 0) {
        int post_result =
            http_post_question(socketId,
             g_current_question);
        if (post_result == 0) {
             strcpy(g_current_answer, "
                 wait");
            while(strcmp(
                g_current_answer, "wait"
                ) == 0){
                 display_loading_screen
                     ();
                 http_get_answer(
                     socketId);
            }
        }
        sl_Close(socketId);
    }
    TextEntry_RequestExit();
}
```

When the user hits enter, this functions puts the loading screen on the display. It then connects to AWS and updates the question field in the device shadow, triggering the GPT call.

5.4 Question Parsing and UI Dispatch

Listing 17: located in AWS IoT.c

```
static question_type_t
    get_question_type(const char*
    question) {
     if (!question) return
         QUESTION_TYPE_NONE;

     int question_len = strlen(question)
         ;
     if (question_len >= 11 && strncmp(
         question, "pin␣labels/", 11) ==
         0) {
          return QUESTION_TYPE_PIN_LABELS
              ;
     } else if (question_len >= 12 &&
         strncmp(question, "pin␣connect/"
         , 12) == 0) {
          return
              QUESTION_TYPE_PIN_CONNECT;
     } else if (question_len >= 13 &&
         strncmp(question, "comp␣purpose/
         ", 13) == 0) {
          return
              QUESTION_TYPE_COMP_PURPOSE;
     }
     return QUESTION_TYPE_NONE;
}

static void display_status(void) {
     if(g_first_answer_frame){
          g_first_answer_frame = false;
          if(strcmp(g_current_answer, "
              parse␣fail") == 0){
               display_parse_fail_ui();
          }else if(strcmp(
              g_current_answer, "
              dns_lookup_failed") == 0){
               display_dns_error_ui();
          }else{
               switch (
                   g_current_question_type)
                    {
                    case
                        QUESTION_TYPE_PIN_LABELS
                        :
                        display_pin_labels_ui
                        (); break;
                    case
                        QUESTION_TYPE_PIN_CONNECT
                        :
                        display_pin_connect_ui
                        (); break;
                    case
                        QUESTION_TYPE_COMP_PURPOS
                        :
                        display_comp_purpose_ui
                        (); break;
                    default:
                        display_default_ui()
                        ; break;
               }
          }
     }
}
```

The code figures out which type of question is currently being answered in the get question type function using prefixes, and triggers the correct UI routine in the display status function.

5.5 Text Entry

Listing 18: located in text entry.c

```
// 2D Keyboard Navigation with Joystick
void ProcessJoystickInput(void) {
     ReadADCChannel(ADC_CH_2, &voltage_x
         ); // Read X/Y joystick
         positions
     ReadADCChannel(ADC_CH_3, &voltage_y
         );

     // Detect movement with deadzone,
         update cursor position
     if (joystick_x > deadzone && !
         joystick_moved_right) {
          current_col++; // Move right on
              keyboard grid
          if (current_col >=
              KEYBOARD_COLS) current_col =
               KEYBOARD_COLS - 1;
```

```
    }
    if (joystick_y > deadzone && !
        joystick_moved_down) {
        current_row++; // Move down on
            keyboard grid
        if (current_row >=
            KEYBOARD_ROWS) current_row =
             KEYBOARD_ROWS - 1;
    }
    // Similar logic for left/up
        movement
}

// Character Selection and Special
    Function Processing
void ProcessButtonInput(void) {
    if (button1_pressed && !
        button1_was_pressed) {
        char selected_char =
            keyboard_chars[current_row][
            current_col];

        if (selected_char == '<') {
             current_question[strlen(
                current_question)-1] = '
                \0'; // Backspace
        } else if (selected_char == '#'
            ) {
             // ENTER: Build final
                question with type
                prefix and trigger
                callback
             snprintf(final_question,
                sizeof(final_question),
                "%s%s",
                     GetQuestionTypePrefix
                        (),
                        current_question
                        );
             g_enter_callback(
                final_question); // Send
                 "pin labels/ft232h" etc
                .
        } else if (selected_char == '?'
             || selected_char == '!' ||
            selected_char == '*') {
             current_question_type = (
                current_question_type ==
                 selected_char) ? 0 :
                selected_char;
             // Toggle question type:
                '?' = "pin labels/", '!'
                 = "pin connect/", '*' =
                 "comp purpose/"
        } else {
                  strcat(current_question, &
                     selected_char); // Add
                     regular character
             }
        }
}
```

These functions are the display and controllers for the on screen keyboard and question selection. The joystick is used to control the cursor on screen to navigate the keyboard and type questions. Button 1 is used in order to select a character, using the row/column of the keyboard to figure out the character.

5.6 Question Display

5.6.1 Parsing Pin Labels

Listing 19: located in question display.c

```
int parsePinLabels(const char* answer,
    int* numberOfPins, char labels[][
    MAX_LABEL_LENGTH]) {
     // Parse "[number of pins: 8] [1:
        VCC, 2:GND, 3:MOSI, 4:MISO,
        ...]" format
     sscanf(answer, "[number␣of␣pins:␣%d
        ]", numberOfPins);
     ptr = strchr(answer, '['); ptr++;
        // Find labels section
     while (*ptr && labelCount < *
        numberOfPins) {
         while (*ptr != ':') ptr++; ptr
            ++; // Skip to label text
         // Extract label until comma or
             ']'
         strncpy(labels[labelCount++],
            ptr, comma - ptr);
         ptr = comma + 1; // Move to
            next label
     }
     return labelCount;
}
```

5.6.2 Parsing Pin Connections

Listing 20: located in question display.c

```
int parsePinConnections(const char*
    answer, PinConnection connections[],
     int maxConnections) {
     // Parse "[pin 1 (VCC) + pin 2 (V+)
        , pin 2 (GND) + pin 3 (GND),
        ...]" format
     while (*ptr && connectionCount <
        maxConnections) {
```

```
        sscanf(tempConnection, "pin␣%d␣
            (%63[^)])␣+␣pin␣%d␣(%63[^)])
            ",
                &connections[
                    connectionCount].
                    pin1_num, connections
                    [connectionCount].
                    pin1_name,
                &connections[
                    connectionCount].
                    pin2_num, connections
                    [connectionCount].
                    pin2_name);
        connectionCount++;
    }
    return connectionCount;
}
```

Since the responses are in a strict format, these parser functions are used to turn the responses into meaningful data that can be displayed. The Pin Labels gets parsed by taking the number of pins, and then parsing the remaining by the colon. Pin connections get parsed by the comma and plus sign and update the connections and count.

5.6.3 Displaying Pin Labels

Listing 21: located in question display.c

```
void QuestionDisplay_ShowPinLabels(
    const char* question, const char*
    answer) {
     parsePinLabels(answer, &
        numberOfPins, labels);
     // Draw component rectangle with
        pins distributed top/bottom
     drawRect(componentX, componentY,
        componentWidth, componentHeight,
         GREEN);
     for (i = 0; i < topPinsCount; i++)
        {
         drawRect(startX + i *
            pinSpacing, componentY -
            pinHeight, pinWidth,
            pinHeight, GREEN);
         sprintf(tempStr, "%d", i + 1);
            Outstr(tempStr, GREEN, BLACK
            , ...); // Pin numbers
     }
}
```

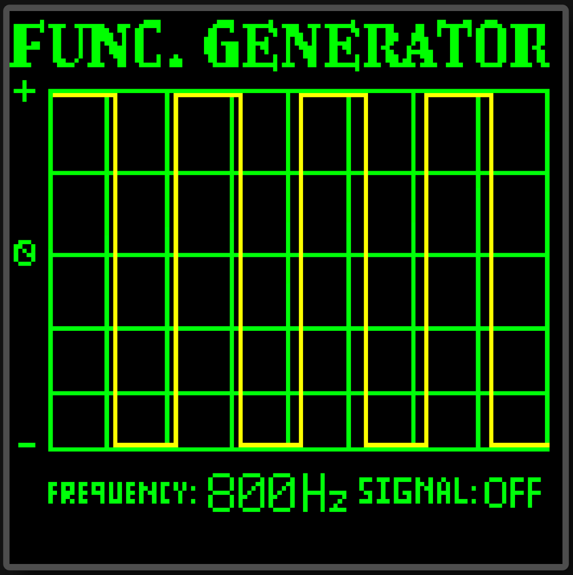
5.6.4 Displaying Pin Connections

Listing 22: located in question display.c

```
void QuestionDisplay_ShowPinConnect(
    const char* question, const char*
    answer) {
     parsePinConnections(answer,
        connections, MAX_CONNECTIONS);
     // Draw two component rectangles
        with connection lines between
        pins
     drawRect(leftRectX, rectY,
        leftRectWidth, rectHeight, GREEN
        );   // Component 1
     drawRect(rightRectX, rectY,
        rightRectWidth, rectHeight,
        GREEN); // Component 2
     for (i = 0; i < numConnections; i
        ++) {
         drawRect(leftPinX, currentPinY,
             pinWidth, pinHeight, GREEN)
            ;  // Left pin
         drawRect(rightPinX, currentPinY
            , pinWidth, pinHeight, GREEN
            ); // Right pin
         drawLine(leftPinX + pinWidth,
            currentPinY, rightPinX,
            currentPinY, BLUE); //
            Connection
     }
}
```

Pin Labels renders a rectangle as the component and draws small pin rectangles with labels indicating which pin they are. Pin Components renders two rectangles, and draws lines between them showing which pins from each component connect to one another.

6 Function Generator



6.1 Defining Signal

Listing 23: located in functiongenerator.c

```
static void HandleJoystickInput(void) {
    float joystick_x = ReadJoystickX();
        // Read ADC for frequency
       control

    if (joystick_x < JOYSTICK_LOW) {
        // Joystick right - increase
           frequency
        if (g_frequency < MAX_FREQUENCY
           ) {
            g_frequency +=
               FREQUENCY_STEP;
            if (g_frequency >
               MAX_FREQUENCY)
               g_frequency =
               MAX_FREQUENCY;
        }
    } else if (joystick_x >
       JOYSTICK_HIGH) {
        // Joystick left - decrease
           frequency
        if (g_frequency > MIN_FREQUENCY
           ) {
            g_frequency -=
               FREQUENCY_STEP;
            if (g_frequency <
               MIN_FREQUENCY)
               g_frequency =
               MIN_FREQUENCY;
        }
    }
}

void FunctionGenerator_PlayFrequency()
   {
    if (g_play_signal) {
        tone(g_frequency);  // Generate
            actual PWM square wave on
           hardware pin
        g_enabled = (g_frequency > 0);
    } else {
        tone(0);  // Stop signal
           generation
    }
}

void FunctionGenerator_Enable(bool
   enable) {
    if (enable) {
        tone(g_frequency);  // Start
           hardware signal generation
        g_enabled = true;
    } else {
        tone(0);  // Stop hardware
           output
        g_enabled = false;
    }
}
```

This code maps X axis movement on the joystick to frequency adjustments in the function. If the play flag is selected, it emits a tone, otherwise it operates in silence. The enable function is what starts and stops the generation.

6.2 Creating Display Buffer

Listing 24: located in functiongenerator.c

```
// Generates digital square wave data
   for oscilloscope visualization
static void GenerateSquareWaveBuffer(
   unsigned long frequency) {
    if (frequency == 0) {
        // Generate flat line for zero
           frequency
        for (int i = 0; i <
           SCOPE_BUFFER_SIZE; i++) {
            g_waveformBuffer[i] = 0.0;
        }
        return;
    }

    // Calculate phase increment per
       sample for continuous phase
    // Normalize frequency to 200Hz = 1
        complete cycle across screen
    float normalized_frequency =
       frequency / 200.0;
    float phase_increment = (2.0 * M_PI
        * normalized_frequency) /
       SCOPE_BUFFER_SIZE;

    for (int i = 0; i <
       SCOPE_BUFFER_SIZE; i++) {
        // Calculate continuous phase
           for this sample
        float phase = i *
           phase_increment;
        float normalized_phase = fmod(
           phase, 2.0 * M_PI);

        // Square wave: high for first
           half of cycle (0 to   ), low
            for second half (   to 2   )
        if (normalized_phase < M_PI) {
            g_waveformBuffer[i] =
               MAX_AMPLITUDE;   // High
                state (+1.0)
        } else {
            g_waveformBuffer[i] = -
                MAX_AMPLITUDE;  // Low
                state (-1.0)
        }
    }
}
```

This code fills g_waveformBuffer[] with digital square wave samples for visualization. Computes a continuous phase increment per buffer index by normalizing the target frequency to the screen width, then writes +1.0 for the first half and –1.0 for the second. This generates a flat line when frequency is zero.

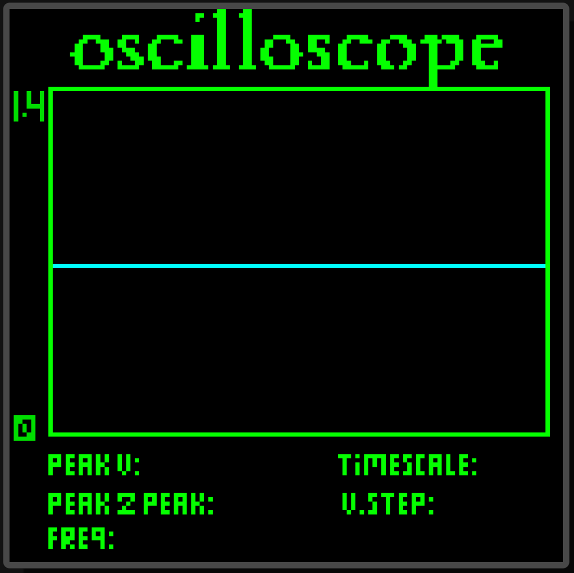
6.3 Displaying UI

Listing 25: located in functiongenerator.c

```
// Renders real-time oscilloscope
   display with grid and waveform trace
static void DrawWaveformDisplay(void) {
    if (screenNeedsUpdate) {
        screenNeedsUpdate = false;

        // Erase previous waveform
           trace
        if (g_previous_trace_valid) {
            for (int i = 1; i <
                SCOPE_BUFFER_SIZE; i++)
                {
                 drawLine(i-1,
                    g_previous_trace_y[i
                    -1], i,
                    g_previous_trace_y[i
                    ], BACKGROUND_COLOR)
                    ;
            }
        }

        // Draw oscilloscope grid lines
        for (int i = 0; i <= 4; i++) {
            int y = SCOPE_TOP + (i * (
                SCOPE_HEIGHT / 4));
            drawFastHLine(9, y, 124,
                GRID_COLOR);  //
                Horizontal grid
        }
        for (int i = 0; i <= 6; i++) {
            int x = (i * 16) + 8;
            drawFastVLine(x, SCOPE_TOP,
                 SCOPE_HEIGHT,
                GRID_COLOR);  //
                Vertical grid
        }

        // Calculate and draw new
           waveform trace
        for (int i = 0; i <
            SCOPE_BUFFER_SIZE; i++) {
             // Map amplitude (-1 to +1)
                 to screen Y coordinate
             int y = SCOPE_TOP +
                SCOPE_HEIGHT/2 - (int)((
                g_waveformBuffer[i] /
                MAX_AMPLITUDE) * (
                SCOPE_HEIGHT/2));
             if (y < SCOPE_TOP) y =
                SCOPE_TOP;
             if (y > SCOPE_TOP +
                SCOPE_HEIGHT) y =
                SCOPE_TOP + SCOPE_HEIGHT
                ;
             g_previous_trace_y[i] = y;
        }

        // Draw connected line segments
            for smooth waveform
        for (int i = 9; i <
            SCOPE_BUFFER_SIZE; i++) {
             drawLine(i-1,
                g_previous_trace_y[i-1],
                 i, g_previous_trace_y[i
                ], SCOPE_COLOR);
        }
        g_previous_trace_valid = true;
    }
}
```

This renders a real time waveform interface, displaying the (g_waveformBuffer) square wave that is being generated on the axes, with the frequency and whether its on or off. Responds to changes in scale by clearing previous traces and drawing new ones.

7 Oscilliscope



7.1 High Speed ADC Sampling

Listing 26: located in oscilliscope.c

```
// Rapidly fills voltage buffer with
   time-series ADC readings
static void BatchSampleBuffer(void) {
    unsigned long ulSample;
    unsigned long startTicks, endTicks,
        elapsedTicks;
    startTicks = SysTickValueGet(); //
        Start timing measurement

    MAP_ADCChannelEnable(ADC_BASE,
        ADC_CH_1); // Enable ADC once
        for efficiency

    // Sample entire buffer rapidly -
       no averaging for maximum speed
    for (int i = 0; i <
        SCOPE_BUFFER_SIZE; i++) {
        while(!MAP_ADCFIFOLvlGet(
            ADC_BASE, ADC_CH_1)); //
            Wait for sample ready
        ulSample = MAP_ADCFIFORead(
            ADC_BASE, ADC_CH_1); // Read
             raw ADC value

        // Convert to voltage: 12-bit
            ADC, 1.4V reference
        g_voltageBuffer[i] = (((float)
            ((ulSample >> 2) & 0x0FFF))
            * 1.4) / 4096;
        MAP_UtilsDelay(timeStep); //
            Adjustable sampling rate via
             joystick control
    }

    MAP_ADCChannelDisable(ADC_BASE,
        ADC_CH_1); // Disable ADC when
        done

    // Calculate actual sampling timing
        for frequency calculations
    endTicks = SysTickValueGet();
    elapsedTicks = (startTicks >=
        endTicks) ? (startTicks -
        endTicks) : 0;
    g_batchSampleTicks = elapsedTicks;
        // Store for frequency analysis

    g_bufferComplete = true; // Signal
        that buffer is ready for
        processing
}
```

This BatchSampleBuffer routine has the ADC capture SCOPE_BUFFER_SIZE voltage samples as rapidly as possible. It timestamps the start and end of the burst to compute the exact sampling interval, fills g_voltageBuffer[], and flags completion.

7.2 Detecting Frequency

Listing 27: located in oscilliscope.c

```
// Analyzes voltage buffer to extract
   signal frequency using zero-crossing
    detection
float ExtractFrequency_ZeroCrossing(
   void) {
    float threshold = MAX_VOLTAGE / 5.0
        f; // Set detection threshold
        (0.28V)
    int crossings = 0;
    bool above_threshold = (
        g_voltageBuffer[0] > threshold);

    if (g_batchSampleTicks == 0) return
         0.0f; // Avoid division by zero

    // Count zero crossings throughout
        the buffer
    for (int i = 1; i <
        SCOPE_BUFFER_SIZE; i++) {
        bool current_above = (
            g_voltageBuffer[i] >
            threshold);
        if (current_above !=
            above_threshold) {
```

```
            crossings++; // Detected a
               crossing
            above_threshold =
               current_above;
        }
    }

    // Calculate actual time span from
       timer measurements
    float actual_time_span = (float)
       g_batchSampleTicks /
       TIMER_FREQ_HZ;

    // Calculate frequency: (complete
       cycles) / (time span)
    // Two crossings = one complete
       cycle
    float frequency = (crossings / 2.0f
       ) / actual_time_span;

    // Apply experimental calibration
       adjustments for breadboard
       characteristics
    if(frequency <= 700) frequency +=
       100;
    else if(frequency <= 795) frequency
        += 200;
    else if(frequency <= 820) frequency
        += 300;
    else if(frequency <= 900) frequency
        += 400;

    return frequency;
}
```

ExtractFrequency_ZeroCrossing scans the waveform in the buffer to count threshold crossings, converts crossings to the cycle count over the measured time span, returning a frequency measurement. The components used on the breadboard for the controller as well as the unshielded wiring for the oscilloscope create an oscillator that has to be adjusted for in the readings. If this part of the project was to be reproduced, the frequencies in the latter if-else statements of the above code would have to be adjusted.

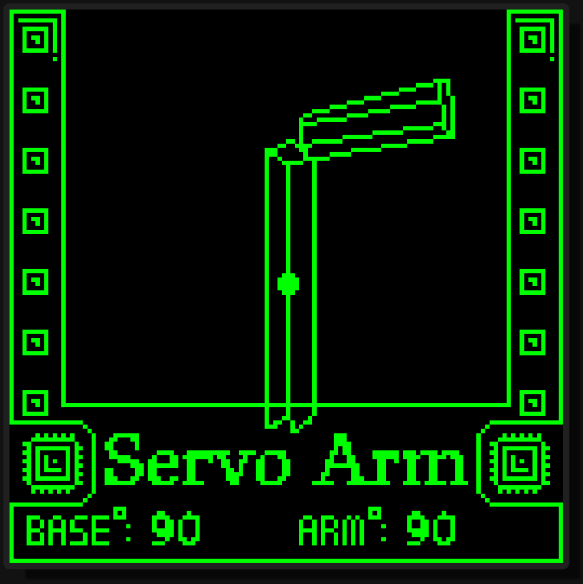
7.3 Trace Rendering and Display Loop

Listing 28: located in oscilliscope.c

```
// Erases old traces, draws new
   waveform, updates measurements
static void DrawOscilloscope(void) {
    float min_voltage, max_voltage;
    GetMinMaxVoltage(&min_voltage, &
       max_voltage); // Find signal
       range

    // STEP 1: Erase previous waveform
       trace
    if (g_previous_trace_valid) {
        for (int i = 11; i <
           SCOPE_BUFFER_SIZE - 4; i++)
           {
            drawLine(i-1,
               g_previous_trace_y[i-1],
                i, g_previous_trace_y[i
               ], BACKGROUND_COLOR);
        }
    }

    // STEP 2: Calculate new trace
       coordinates from voltage buffer
    if (g_bufferComplete) {
        for (int i = 0; i <
           SCOPE_BUFFER_SIZE; i++) {
            // Map voltage to screen Y
               coordinate (inverted -
               higher voltage = lower Y
               )
            int y = SCOPE_TOP +
               SCOPE_HEIGHT - (int)((
               g_voltageBuffer[i] /
               voltageStep) *
               SCOPE_HEIGHT);
            if (y < SCOPE_TOP) y =
               SCOPE_TOP; // Clamp to
               display area
            if (y > SCOPE_TOP +
               SCOPE_HEIGHT) y =
               SCOPE_TOP + SCOPE_HEIGHT
               ;
            g_previous_trace_y[i] = y;
               // Store for next frame'
               s erase operation
        }

        // STEP 3: Draw new waveform
           trace as connected line
           segments
        for (int i = 11; i <
           SCOPE_BUFFER_SIZE - 4; i++)
           {
            drawLine(i-1,
               g_previous_trace_y[i-1],
                i, g_previous_trace_y[i
               ], SCOPE_COLOR);
        }
        g_previous_trace_valid = true;
    }

    // STEP 4: Update measurement
       displays
    float frequency =
        ExtractFrequency_ZeroCrossing();
    char buffer[32];
    sprintf(buffer, "%.0fHz", frequency
        ); Outstr(buffer, GREEN, BLACK,
        26, 121, 128, 128);
    sprintf(buffer, "%.2f", max_voltage
        ); Outstr(buffer, GREEN, BLACK,
        31, 102, 80, 120);
    sprintf(buffer, "%.2f", max_voltage
         - min_voltage); Outstr(buffer,
        GREEN, BLACK, 48, 112, 85, 128);
}
```

DrawOscilloscope renders the full waveform on the screen: it erases the prior trace, maps each sample to a screen Y-coordinate, draws the waveform as connected line segments, and updates text overlays for frequency, peak voltage, and amplitude.

8 Servo Control



8.1 Physical PWM Control

Listing 29: located in servo$_c$ontrol.c

```
// Converts joystick input to precise
   PWM signals for servo positioning
static void SetServo1Angle(int angle) {
    unsigned long ulClockHz = 80000000;
    unsigned long ulPeriodCycles = (
        ulClockHz / SERVO_FREQ_HZ); //
        300Hz for digital servos
    unsigned long ulPulseWidthUs;

    if (angle < 0) angle = 0;
    if (angle > 180) angle = 180;

    // Map servo angle (0-180  ) to
        pulse width (1000-2000 s )
    ulPulseWidthUs = SERVO_MIN_US + ((
        SERVO_MAX_US - SERVO_MIN_US) *
        (180 - angle) / 180);

    // Calculate timer match value for
        desired pulse width
    unsigned long ulMatchCycles =
        ulPeriodCycles - ((ulClockHz /
        1000000) * ulPulseWidthUs);
    unsigned long ulMatchPrescaler =
        ulMatchCycles >> 16;
    unsigned long ulMatchValue =
        ulMatchCycles & 0xFFFF;

    // Set hardware timer match values
        for PWM generation
    TimerMatchSet(TIMERA3_BASE, TIMER_A
        , ulMatchValue);
    TimerPrescaleMatchSet(TIMERA3_BASE,
         TIMER_A, ulMatchPrescaler);

    // Update 3D visualization angle
        for this servo
    g_visualAngle1 = (180 + 1 * (float)
        (angle) * M_PI) / 180.0f;
}

bool ServoControl_RunFrame(void) {
    // Read joystick ADC inputs for
        control
    ReadADCChannel(ADC_CH_2, &voltage_x
        );  // X-axis joystick
    ReadADCChannel(ADC_CH_3, &voltage_y
        );  // Y-axis joystick

    // Convert joystick positions to
        servo angles with deadzone
        detection
    if (fabs(((voltage_x)/1.4) - 0.5)
        >= 0.1) {
         float joystickX = ((voltage_x)
            /1.4) - 0.45;  // Normalize
            to -0.5 to 0.5
         newServo1Angle = g_servo1Angle
            + (10 * joystickX);  //
            Proportional control
         if (newServo1Angle < 0)
            newServo1Angle = 0;
         if (newServo1Angle > 180)
            newServo1Angle = 180;
         SetServo1Angle(newServo1Angle);
              // Apply to hardware PWM
    }
}
```

SetServo1Angle (and its helper) map a target angle (0–180 deg) to a 1-2 ms PWM pulse width, compute the prescale and match settings for the CC3200, program the timer registers, and mirrors this angle in the 3D visualization state.

8.2 Generating Simulated Arm

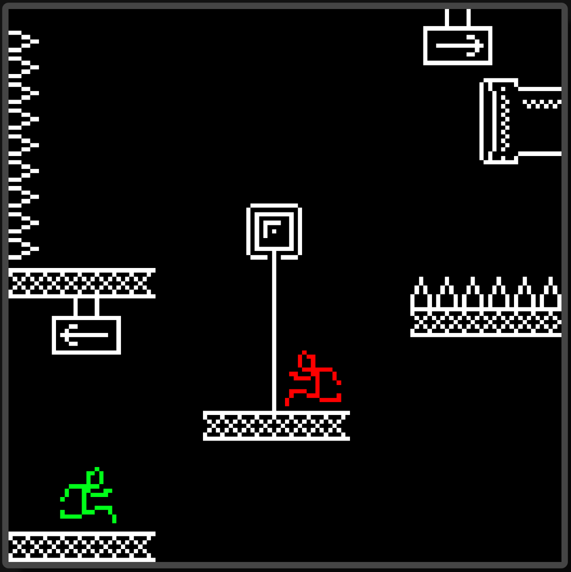
Listing 30: located in servo$_{c}ontrol.c$

```
// Creates real-time 3D wireframe
   animation matching physical servo
   positions
static void RotatePoint(float x, float
   y, float z, float* rx, float* ry,
   float* rz, int vertexIndex) {
    float temp_x = x, temp_y = y,
       temp_z = z;

    // Apply servo 2 rotation to arm
       segments (vertices 8-15)
    if (vertexIndex >= 8) {
        temp_y -= ARM_HEIGHT/2;  //
           Translate to rotation point
        // Z-axis rotation for arm
           bending
        float rot_x = temp_x * cosf(
           g_visualAngle2) - temp_y *
           sinf(g_visualAngle2);
        float rot_y = temp_x * sinf(
           g_visualAngle2) + temp_y *
           cosf(g_visualAngle2);
        temp_x = rot_x; temp_y = rot_y;
        temp_y += ARM_HEIGHT/2;  //
           Translate back
    }

    // Apply servo 1 rotation (base
       rotation around Y-axis) to all
       vertices
    *rx = temp_x * cosf(g_visualAngle1
       /2) + temp_z * sinf(
       g_visualAngle1/4);
    *ry = temp_y;  // No Y change for Y
       -axis rotation
    *rz = -temp_x * sinf(g_visualAngle1
       /4) + temp_z * cosf(
       g_visualAngle1/4);
}

static void RenderServoArm(uint16_t
   color) {
    // Calculate 3D rotations and
       project to 2D screen coordinates
```

```
    for (int i = 0; i < NUM_VERTICES; i
       ++) {
        float rx, ry, rz;
        RotatePoint(g_arm_vertices[i
           ][0], g_arm_vertices[i][1],
           g_arm_vertices[i][2], &rx, &
           ry, &rz, i);

        // Perspective projection: f /
           (z + z_offset)
        float perspective = 100.0f / (
           rz + 80.0f);
        g_projected_vertices[i][0] =
           SCREEN_CENTER_X + (int)(rx *
            perspective);
        g_projected_vertices[i][1] =
           SCREEN_CENTER_Y - (int)(ry *
            perspective);
    }

    // Erase previous wireframe by
       drawing over it in black
    if (!first_frame) {
        for (int i = 0; i < NUM_EDGES;
           i++) {
            int v1 = g_arm_edges[i][0],
                v2 = g_arm_edges[i][1];
            drawLine(
                g_prev_projected_vertices
                [v1][0],
                g_prev_projected_vertices
                [v1][1],
                      g_prev_projected_vertice
                          [v2][0],
                          g_prev_projected_vert
                          [v2][1], BLACK);
        }
    }

    // Draw new wireframe with
       connected line segments
    for (int i = 0; i < NUM_EDGES; i++)
        {
        int v1 = g_arm_edges[i][0], v2
           = g_arm_edges[i][1];
        drawLine(g_projected_vertices[
           v1][0], g_projected_vertices
           [v1][1],
                 g_projected_vertices[v2
                     ][0],
                     g_projected_vertices
                     [v2][1], color);
    }
}
```

These routines are to draw the simulation of the servo arm on the screen. RotatePoint and RenderServoArm apply 3D transformations, bending the arm at the elbow hinge using servo 2 angle, then rotating the entire arm via servo 1's angle. This projects the servo onto a 2D screen.

9 Video Game



9.1 Input Handling

Listing 31: located in video game.c

```
// Converts analog joystick ADC
   readings and button states to game
   controls
static void UpdatePlayerPhysics(void) {
    float voltage_59, voltage_60;
    bool jumpButtonPressed =
        IsJumpButtonPressed();
    unsigned long currentTime =
        GetCurrentTimeMs();

    // Read X-axis from ADC for
        horizontal control
    ReadADCChannel(ADC_CH_2, &
        voltage_59);

    // Apply horizontal acceleration
        based on analog stick with
        deadzone
    if(fabs(((voltage_59)/1.4)-.5) >=
        .1){
         g_playerVX -= (((voltage_59)
             /1.4)-.5)*(HORIZONTAL_ACCEL)
             ;
    }
```

```
    // Handle jumping with Button 1 -
       supports double jump
    if (jumpButtonPressed && !
       g_wasButton1Pressed && (
       currentTime - g_lastJumpTime >
       50) &&
        (g_isOnGround ||
           double_jump_available)) {
        if(!g_isOnGround) {
            double_jump_available =
                false;
            playing_double_jump_animation
                = true;
        } else {
            playing_jump_animation =
                true;
        }
        g_playerVY = JUMP_VELOCITY;
        g_lastJumpTime = currentTime;
    }
}
```

Reads joystick X axis movement and button inputs, translating joystick usage into horizontal acceleration, and detects jump presses with timing and double-jump logic on the button with multiple presses.

9.2 Updating Player Physics

Listing 32: located in video game.c

```
// Implements platformer physics with
   gravity, velocity limits, and
   collision handling
static void UpdatePlayerPhysics(void) {
    // Apply horizontal speed limit
    if (g_playerVX >
       MAX_HORIZONTAL_SPEED) {
        g_playerVX =
           MAX_HORIZONTAL_SPEED;
    } else if (g_playerVX < -
       MAX_HORIZONTAL_SPEED) {
        g_playerVX = -
           MAX_HORIZONTAL_SPEED;
    }

    // Apply gravity when airborne
    if (!g_isOnGround) {
        g_playerVY += GRAVITY;
    }

    // Apply horizontal damping for
       realistic movement
    g_playerVX *= HORIZONTAL_DAMPING;

    // Update position based on
       velocity
    g_playerX += g_playerVX;
    g_playerY += g_playerVY;

    // Check collision with level
       bitmap
    const uint8_t* levelBitmap =
       get_map_frame(g_currentMapFrame)
       ;
    if (levelBitmap != NULL) {
        CheckBitmapCollision(0, 0,
           levelBitmap, 128, 128, 1);
    }

    // Boundary collision - prevent
       leaving screen edges
    if (g_playerX < 0) { g_playerX = 0;
        g_playerVX = 0; }
    if (g_playerX > SCREEN_WIDTH -
       CHARACTER_WIDTH) {
        g_playerX = SCREEN_WIDTH -
           CHARACTER_WIDTH; g_playerVX
           = 0;
    }

    // Check if player fell off bottom
       - reset level
    if (g_playerY < 0) {
       VideoGame_Initialize(); return;
       }
}
```

These are further routines for player movement in the game. This enforces a max speed for the player, applies gravity when jumping/airborn, moves the player realistically, does collisions with obstacles, and checks if the player falls off the map.

9.3 Enemies

Listing 33: located in video game.c

```
// Manages enemy patrol behavior,
   animation, and player collision
static void UpdateEnemyPhysics(void) {
    for (int i = 0; i < g_enemyCount; i
       ++) {
        Enemy* enemy = &g_enemies[i];

        // Move enemy based on current
           direction and speed
        enemy->x += enemy->direction *
           enemy->speed;

        // Reverse direction at
           boundaries
        if (enemy->x <= enemy->x1) {
            enemy->x = enemy->x1;
            enemy->direction = 1; //
               Move right
        } else if (enemy->x >= enemy->
           x2) {
            enemy->x = enemy->x2;
            enemy->direction = -1; //
               Move left
        }
    }
}

static bool CheckPlayerEnemyCollision(
   void) {
    int playerX = (int)g_playerX,
       playerY = (int)g_playerY;
    int collisionY = playerY -
       CHARACTER_HEIGHT;

    for (int i = 0; i < g_enemyCount; i
       ++) {
        Enemy* enemy = &g_enemies[i];
        int enemyX = (int)enemy->x,
           enemyY = (int)enemy->y;
        int enemyCollisionY = enemyY -
           CHARACTER_HEIGHT;

        // AABB collision detection
        if (playerX < enemyX +
           CHARACTER_WIDTH &&
            playerX + CHARACTER_WIDTH >
                enemyX &&
            collisionY <
               enemyCollisionY +
               CHARACTER_HEIGHT &&
            collisionY +
               CHARACTER_HEIGHT >
               enemyCollisionY) {
            return true; // Player hit
               enemy - reset level
        }
    }
    return false;
}
```

This is the logic for enemies (in red) in the game. The enemies move in predefined patterns, and reverse direction when hitting predefined boundaries. The CheckPlayerEnemyCollision checks whether the player comes into contact with an enemy, and returns whether the player has done so, triggering a reset of the level.

9.4 Doors

Listing 34: located in video game.c

```
// Detects when player enters door
   hitboxes and transitions between map
```

```
    levels
static bool CheckDoorEntry(void) {
    int playerX = (int)g_playerX,
        playerY = (int)g_playerY;
    int collisionY = playerY -
        CHARACTER_HEIGHT;

    for (int i = 0; i < g_doorCount; i
        ++) {
        Door* door = &g_doors[i];

        // Check if player overlaps
            with door hitbox
        if (playerX < door->x + door->
            width &&
            playerX + CHARACTER_WIDTH >
                door->x &&
            collisionY < door->y + door
                ->height &&
            collisionY +
                CHARACTER_HEIGHT > door
                ->y) {

            // Player entered door -
                switch to target map
            if (g_currentMapFrame !=
                door->targetMap) {
                g_currentMapFrame =
                    door->targetMap;
                return true; // Map
                    changed
            }
        }
    }
    return false; // No door entered
}

static void AddDoor(int x, int y, int
    width, int height, int targetMap) {
    if (g_doorCount < MAX_DOORS) {
        g_doors[g_doorCount] = {x, y,
            width, height, targetMap};
        g_doorCount++;
    }
}
```

Doors are used to transition between levels. If a player is at a spot that is in a door's hitbox, save the new level to go to, and trigger a change in the map level (either going back or forward a level).

9.5 Killboxes

Listing 35: located in video game.c

```
// Defines deadly areas that reset the
    player when touched (spikes, pits,
    etc.)
static bool CheckKillboxEntry(void) {
    int playerX = (int)g_playerX,
        playerY = (int)g_playerY;
    int collisionY = playerY -
        CHARACTER_HEIGHT;

    for (int i = 0; i < g_killboxCount;
        i++) {
        Killbox* killbox = &g_killboxes
            [i];

        // Check if player overlaps
            with killbox
        if (playerX < killbox->x +
            killbox->width &&
            playerX + CHARACTER_WIDTH >
                killbox->x &&
            collisionY < killbox->y +
                killbox->height &&
            collisionY +
                CHARACTER_HEIGHT >
                killbox->y) {
            return true; // Player hit
                hazard - reset level
        }
    }
    return false; // No hazard hit
}

static void AddKillbox(int x, int y,
    int width, int height) {
    if (g_killboxCount < MAX_KILLBOXES)
        {
        g_killboxes[g_killboxCount] = {
            x, y, width, height};
        g_killboxCount++;
    }
}
```

Killboxes are areas on the map that the player has to avoid, otherwise they lose that level. AddKillbox is used to register a killbox on the map, and CheckKillboxEntry is used to check whether the player is currently hitting a hitbox. If so, trigger a reset in the level.

9.6 Sprite Based Rendering

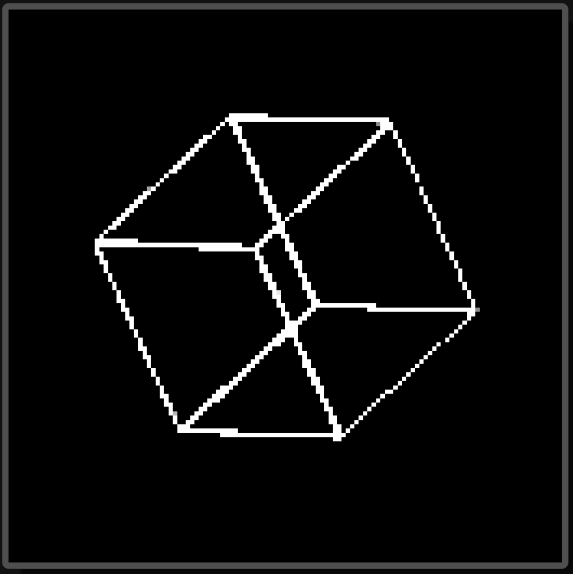
Listing 36: located in video game.c

```
// Manages character animation frames
    and sprite rendering with state-
    based selection
static const uint8_t*
    SelectCharacterBitmap(float vx, ...
    further params) {
```

```
    // Priority-based animation
       selection
    if (playingDoubleJumpAnimation) {
        return
            get_character_double_jump_frame
            ((int)characterFrame);
    } else if (playingJumpAnimation) {
        return get_character_jump_frame
            ((int)characterFrame);
    } else if (vx < -1) {
        return
            get_character_run_left_frame
            ((int)characterFrame);
    } else if (vx > 1) {
        return
            get_character_run_right_frame
            ((int)characterFrame);
    } else {
        // Idle state - use appropriate
            directional frame
        return (vx >= 0) ?
            get_character_run_right_frame
            (3) :
            get_character_run_left_frame
            (3);
    }
}

static void UpdateCharacterAnimation(
    float vx, ... further params) {
    // Handle jump animations with
       frame advancement
    if (*playingDoubleJumpAnimationPtr)
        {
        *characterFramePtr += 0.35;
        if(*characterFramePtr >=
            CHARACTER_DOUBLE_JUMP_FRAME_COUNT
             - 1) {
             *characterFramePtr =
                CHARACTER_DOUBLE_JUMP_FRAME_COUNT
                 - 1;
             if (isOnGround) { *
                playingDoubleJumpAnimationPtr
                 = false; *
                characterFramePtr = 0; }
        }
    } else if (*playingJumpAnimationPtr
       ) {
        if (*characterFramePtr <
            CHARACTER_JUMP_FRAME_COUNT -
             1) *characterFramePtr +=
            0.35;
        if (isOnGround) { *
            playingJumpAnimationPtr =
            false; *characterFramePtr =
            0; }
    } else if (isOnGround && fabs(vx) >
        0.5) {
        // Running animation - frame
           rate based on movement speed
        *characterFramePtr += (2 * fabs
           (vx)) / MAX_HORIZONTAL_SPEED
           ;
        int maxFrames = (vx > 0) ?
           CHARACTER_RUN_RIGHT_FRAME_COUNT
            :
           CHARACTER_RUN_LEFT_FRAME_COUNT
           ;
        if (*characterFramePtr >=
           maxFrames) *
           characterFramePtr = 0;
    }
}
```

SelectCharacterBitmap is used to select the correct character bitmap (whether they are jumping, running, stopped), and UpdateCharacterAnimation updates the character animation for the movement that is occurring (speed dependent for running, until landing for jump).

10 3D Cube



10.1 BMA222 Reading and Processing

Listing 37: located in cube3d.c

```
// Converts I2C accelerometer readings
   into gravity vector for physics
   simulation
bool Cube3D_RunFrame(void) {
    if(ReadAccelerometerData() ==
       SUCCESS) {
        UpdatePhysics(); // Process
            accelerometer data into
            physics
        RenderEnvironment(WALL_COLOR);
        RenderCube(WHITE);
        MAP_UtilsDelay(80000);
    } else {
        UART_PRINT("Error␣reading␣
            accelerometer!\n\r");
        MAP_UtilsDelay(800000);
    }
    return true;
}

void UpdatePhysics() {
    // Convert accelerometer readings
       to gravity direction vector
    float gravityX = g_iAccelY;   //
       Map accelerometer axes to world
       coordinates
    float gravityY = -g_iAccelZ;  //
       Flip signs based on sensor
       orientation
    float gravityZ = -g_iAccelX;  // Z-
       axis mapping

    // Calculate magnitude and
       normalize gravity vector
    float gravityMagnitude = sqrtf(
       gravityX * gravityX + gravityY *
        gravityY + gravityZ * gravityZ)
       ;

    if (gravityMagnitude > 0.001f) {
        // Normalize to unit vector
        gravityX /= gravityMagnitude;
        gravityY /= gravityMagnitude;
        gravityZ /= gravityMagnitude;

        // Apply gravity to cube
            velocity (scaled by gravity
            strength)
        g_velocityX += gravityX *
            GRAVITY_STRENGTH * TIME_STEP
            ;
        g_velocityY += gravityY *
            GRAVITY_STRENGTH * TIME_STEP
            ;
        g_velocityZ += gravityZ *
            GRAVITY_STRENGTH * TIME_STEP
            ;
    }
}
```

Cube3D_RunFrame reads raw I2C accelerometer data and saves it, then calls RenderEnviroment and RenderCube to display the simulation on the OLED screen. The first part of UpdatePhysics maps sensor axes to world gravity vectors, normalizes the result and saves it for use of other functions.

10.2 Calculating Physics

Listing 38: located in cube3d.c

```
// Implements realistic physics with
   gravity, damping, collisions, and
   stabilizing torque
void UpdatePhysics() {
    // Apply damping to velocities (
       simulates air resistance/
       friction)
    g_velocityX *= DAMPING;
    g_velocityY *= DAMPING;
    g_velocityZ *= DAMPING;

    // Update position based on
       velocity
    g_positionX += g_velocityX *
       TIME_STEP;
    g_positionY += g_velocityY *
       TIME_STEP;
    g_positionZ += g_velocityZ *
       TIME_STEP;

    // Update angular velocity with
       damping
    g_angularVelocityX *=
       ANGULAR_DAMPING;
    g_angularVelocityY *=
       ANGULAR_DAMPING;
    g_angularVelocityZ *=
       ANGULAR_DAMPING;

    // Update orientation based on
       angular velocity
    g_angleX += g_angularVelocityX *
       TIME_STEP;
    g_angleY += g_angularVelocityY *
       TIME_STEP;
    g_angleZ += g_angularVelocityZ *
       TIME_STEP;

    // Calculate stabilizing torque to
       align cube faces with gravity
    float torqueX, torqueY, torqueZ;
    CalculateStabilizingTorque(gravityX
       , gravityY, gravityZ, &torqueX,
       &torqueY, &torqueZ);
    g_angularVelocityX += torqueX;
    g_angularVelocityY += torqueY;
    g_angularVelocityZ += torqueZ;
    // Collision detection and response
        with environment boundaries
    CheckAndResolveCollisions();
}

void CheckAndResolveCollisions() {
    // Check each vertex against
       environment boundaries
    if (worldX < PHYSICS_MIN_X ||
       worldX > PHYSICS_MAX_X) {
        // Reflect velocity with
           restitution and push cube
           back
        g_velocityX -= (1.0f +
           RESTITUTION) * dotProduct *
           collisionNormalX;
        g_positionX +=
           penetrationCorrection; //
           Push back inside boundary
    }
    // Similar for Y and Z axes...
}
```

Continuing in UpdatePhysics, the function then applies damping to the velocities in order to simulate loss of energy. It also computes the orientation based on the angular velocity, and calculates stabilizing torque in order to align the cube face with gravity. This produces the realistic effect of the cube tending to land and remain on it's face. Without it, the simplified physics engine creates a cube equally likely to balance on a single corner than on it's face. It also checks if there are collisions with the boundaries, and resolves them.

10.3 Checking and Resolving Collisons

10.3.1 Transforming Cube Vertices to World Space

Listing 39: located in cube3d.c

```
for (i = 0; i < NUM_VERTICES; i++) {
    // Apply rotation
    RotatePoint(g_cube_vertices[i][0],
       g_cube_vertices[i][1],
       g_cube_vertices[i][2], &rx, &ry,
        &rz);

    // Translate to world position
    float worldX = g_positionX + rx -
       SCREEN_CENTER_X;
    float worldY = g_positionY + ry -
       SCREEN_CENTER_Y;
    float worldZ = g_positionZ + rz;
```

For each of the cubes 8 vertices, the vertex is rotated based on the cubes current orientation and translated to the cubes world position, giving the 3D location of each corner.

10.3.2 Checking the Vertex Against Boundaries

Listing 40: located in cube3d.c

```
if (worldX < PHYSICS_MIN_X) {
    collisionDetected = true;
    collisionNormalX += 1.0f;  //
       Normal pointing right

    // Push cube back inside boundary
    float penetration = PHYSICS_MIN_X -
        worldX;
    g_positionX += penetration;
}
```

Each vertex is then checked to see if it is located outside of the environment wall boundaries. Listed above is the check for the X coordinate.

10.3.3 Penetration Resolution

Listing 41: located in cube3d.c

```
float penetration = PHYSICS_MIN_X -
   worldX;  // How far through the wall
g_positionX += penetration;
                    // Push cube back
collisionNormalX += 1.0f;
                     // Remember we hit
    the left wall
```

When a vertex is confirmed to be outside of a boundary, the penetration depth is calculated to see how far beyond the wall it extended, then the entire cubes position is moved backward "penetration" amount in that coordinate direction. The collisionNormal variable is also updated for use in Velocity Reflection.

10.3.4 Velocity Reflection

Listing 42: located in cube3d.c

```
// Calculate dot product of velocity
   and normal
float dotProduct = g_velocityX *
   collisionNormalX + g_velocityY *
   collisionNormalY + g_velocityZ *
   collisionNormalZ;

// Reflect velocity based on collision
   normal
if (dotProduct < 0) {
    g_velocityX -= (1.0f + RESTITUTION)
        * dotProduct * collisionNormalX
       ;
    g_velocityY -= (1.0f + RESTITUTION)
        * dotProduct * collisionNormalY
        ;
    g_velocityZ -= (1.0f + RESTITUTION)
        * dotProduct * collisionNormalZ
        ;
}
```

The physics of bouncing is simulated by first using a dot product of the velocity and collision normal vectors to measure how much the cube is moving into the wall. If this dot product is negative, it means that the cube has moved into the wall, and its velocity must be reflected to simulate a bounce. The bounce is an adjustment to the cube's velocity vector, and is calculated based on the cubes current collison normals in each axis, the earlier calculated dot product, and the RESTITUTION factor, which is a measure of bounciness (0 = no bounce, 1 = complete velocity reflection).

10.3.5 Random Rotation

Listing 43: located in cube3d.c

```
g_angularVelocityX += (rand() % 100) /
    1000.0f - 0.05f;
g_angularVelocityY += (rand() % 100) /
    1000.0f - 0.05f;
g_angularVelocityZ += (rand() % 100) /
    1000.0f - 0.05f;
```

To further the illusion of a true physical collision, once the main calculations are completed, slight random rotational effects are added to the cube to simulate random tumbling.

10.4 3D rendering

Listing 44: located in cube3d.c

```
// Transforms 3D cube vertices to 2D
    screen coordinates and renders
    wireframe
void RotatePoint(float x, float y,
    float z, float* rx, float* ry, float
    * rz) {
    // Apply X-axis rotation (pitch)
    float temp_y = y * cosf(g_angleX) -
        z * sinf(g_angleX);
    float temp_z = y * sinf(g_angleX) +
        z * cosf(g_angleX);

    // Apply Y-axis rotation (yaw)
    float temp_x = x * cosf(g_angleY) +
        temp_z * sinf(g_angleY);
    temp_z = -x * sinf(g_angleY) +
       temp_z * cosf(g_angleY);

    // Apply Z-axis rotation (roll)
    *rx = temp_x * cosf(g_angleZ) -
        temp_y * sinf(g_angleZ);
    *ry = temp_x * sinf(g_angleZ) +
        temp_y * cosf(g_angleZ);
    *rz = temp_z;
}

void ProjectPoint(float x, float y,
    float z, int* px, int* py) {
    // Translate point based on cube
        position
    x += g_positionX - SCREEN_CENTER_X;
    y += g_positionY - SCREEN_CENTER_Y;
    z += g_positionZ;

    // Apply perspective projection
    float f = 200.0f, z_offset = 100.0f
        ;
    if (fabsf(z + z_offset) > 0.001f) {
        float perspective = f / (z +
            z_offset);
        *px = SCREEN_CENTER_X + (int)(x
             * perspective);
        *py = SCREEN_CENTER_Y - (int)(y
             * perspective); // Y
            inverted for screen
    }
}

void RenderCube(uint16_t color) {
    // Calculate projected vertices for
        current frame
    for (int i = 0; i < NUM_VERTICES; i
        ++) {
        float rx, ry, rz;
        RotatePoint(g_cube_vertices[i
            ][0], g_cube_vertices[i][1],
             g_cube_vertices[i][2], &rx,
             &ry, &rz);
        ProjectPoint(rx, ry, rz, &
            g_projected_vertices[i][0],
            &g_projected_vertices[i][1])
            ;
    }

    // Erase previous frame by drawing
        black lines
    if (!g_first_frame) {
        for (int i = 0; i < NUM_EDGES;
            i++) {
             int v1 = g_cube_edges[i
                ][0], v2 = g_cube_edges[
                i][1];
            drawLine(
                g_prev_projected_vertices
                [v1][0],
                g_prev_projected_vertices
                [v1][1],
                    g_prev_projected_vertices
                        [v2][0],
                        g_prev_projected_vertices
                        [v2][1], BLACK);
        }
    }

    // Draw new wireframe edges
    for (int i = 0; i < NUM_EDGES; i++)
        {
        int v1 = g_cube_edges[i][0], v2
            = g_cube_edges[i][1];
        drawLine(g_projected_vertices[
            v1][0], g_projected_vertices
            [v1][1],
                g_projected_vertices[v2
                    ][0],
                    g_projected_vertices
                    [v2][1], color);
    }
}
```

RotatePoint uses Euler rotations to project the 3D cube onto a 2D frame. ProjectPoint projects the rotated cube onto the axes. This is done by first generating a simulated camera "focal length" and offset in the z direction, the calculation a perspective value based on the combination of the current z coordinate, focal length, and z offset. This perspective factor is used to scale the coordinates in the x and y direction to reflect distance changes in the z direction. RenderCube then uses the DrawLine function to draw the cube on the screen by connecting vertices that represent edges. Finding the correct vertices to connect is accomplished by referencing a list of the cubes edges defined as a constant earlier in the program. The coordinates of the projected vertices are also stored for use in the next frame for a "double buffering" rendering technique, where black lines are drawn over the previous wireframe cube instead of using fillScreen to achieve higher framerates.

11 Sound Effects

11.1 Tone Function

Listing 45: located in sound Effects.c

```
// Converts frequency values to precise
    timer/prescaler settings for PWM
   audio output
static unsigned long
   CalculateOptimalPrescaler(unsigned
   long frequency) {
    unsigned long raw_period =
        TIMER_CLOCK_HZ / frequency; //
        80MHz / target frequency
    unsigned long best_prescaler = 0,
        best_error = 0xFFFFFFFF;

    // If raw period fits in 16-bit
        timer range, no prescaling
        needed
    if (raw_period >= MIN_PERIOD &&
        raw_period <= MAX_PERIOD) {
         return 0;  // No prescaling
            needed
    }

    // Search through prescaler values
        0-255 to find best frequency
        match
    for (unsigned long prescaler = 0;
        prescaler <= 255; prescaler++) {
         unsigned long prescaled_clock =
             TIMER_CLOCK_HZ / (prescaler
             + 1);
         unsigned long test_period =
            prescaled_clock / frequency;

         if (test_period < MIN_PERIOD ||
             test_period > MAX_PERIOD)
            continue;

         // Calculate frequency error
            with this prescaler
         unsigned long actual_freq =
            prescaled_clock /
            test_period;
         unsigned long error = (
            actual_freq > frequency) ? (
            actual_freq - frequency) : (
            frequency - actual_freq);

         if (error < best_error) {
             best_error = error;
             best_prescaler = prescaler;
             if (error == 0) break; //
                Perfect match found
         }
    }
    return best_prescaler;
}

void tone(unsigned long frequency) {
    unsigned long ulPrescaler =
        CalculateOptimalPrescaler(
        frequency);
    unsigned long ulPrescaledClock =
        TIMER_CLOCK_HZ / (ulPrescaler +
        1);
    unsigned long ulPeriod =
        ulPrescaledClock / frequency;

    // Configure timer with calculated
        values
    MAP_TimerPrescaleSet(TIMERA2_BASE,
        TIMER_B, ulPrescaler);
    MAP_TimerLoadSet(TIMERA2_BASE,
        TIMER_B, ulPeriod);
    MAP_TimerMatchSet(TIMERA2_BASE,
        TIMER_B, ulPeriod / 2); // 50%
        duty cycle
}
```

CalculateOptimalPrescaler searches through the prescaler settings for the value that gives the lowest error for the frequency, and uses that if needed. tone then uses that prescaler in order to configure the timer for a square wave function at the requested frequency.

11.2 Storing Sound Effects

Listing 46: located in sound Effects.c

```
// Manages sound effect arrays and non-
    blocking sequential playback with
    timing
const unsigned long SUCCESS_TONES[] =
    {1047, 1319, 1568}; // C6, E6, G6
    frequencies
const unsigned long SUCCESS_DURATIONS[]
     = {1, 1, 10};      // Duration in
    time units

const unsigned long THEME_TONES[] = {
    0, 294, 294, 294, 294, 100, 0, 494,
         494, 494, 494, 0, // Musical
        sequence
    294, 294, 294, 294, 0, 494, 494,
        494, 494, 0, 698, 698
    // ... (207 total notes for
        complete melody)
};
const unsigned long THEME_DURATIONS[] =
    {
    1.2, 1.2, 1.2, 1.2, 1.0, 0.2, 0.8,
        1.2, 1.2, 1.2, 1.2, 12.2
    // ... (207 corresponding durations
        )
};

typedef struct {
    const unsigned long *tones;       //
         Pointer to current melody's
        frequency array
    const unsigned long *durations;   //
         Pointer to current melody's
        duration array
    int melodyLength;                 //
         Total number of notes in
        current melody
    int currentNoteIndex;             //
         Index of currently playing note
    unsigned long noteStartTime;      //
         Timestamp when current note
        started
    bool playing;                     //
         Is any sound currently playing
    bool looping;                     //
         Should current melody loop
        continuously
} SoundState;

void PlaySoundEffect(const unsigned
    long *tones, const unsigned long *
    durations, int length) {
    g_SoundState.tones = tones;
                     // Set melody
        arrays
    g_SoundState.durations = durations;
    g_SoundState.melodyLength = length;
    g_SoundState.currentNoteIndex = 0;
              // Start at first note
    g_SoundState.playing = true;
    g_SoundState.noteStartTime =
        getTimeMs(); // Record start
        time
    tone(tones[0]);
                                 //
        Begin playing first note
}

void UpdateSoundEffects() {
    if (!g_SoundState.playing) return;

    unsigned long currentTime =
        getTimeMs();
    // Check if current note duration
        has elapsed
    if (currentTime - g_SoundState.
        noteStartTime >= g_SoundState.
        durations[g_SoundState.
        currentNoteIndex]) {
         g_SoundState.currentNoteIndex
            ++;       // Advance to next
            note

         if (g_SoundState.
            currentNoteIndex >=
            g_SoundState.melodyLength) {
```

```
            if (g_SoundState.looping) {
                    // Restart melody
                if looping
                g_SoundState.
                    currentNoteIndex =
                    0;
                tone(g_SoundState.tones
                    [0]);
            } else {
                g_SoundState.playing =
                    false; // End
                    playback
                tone(0);

                    // Stop PWM output
            }
            return;
        }

        g_SoundState.noteStartTime =
            currentTime;
        tone(g_SoundState.tones[
            g_SoundState.
            currentNoteIndex]); // Play
            next note
    }
}
```

This routine defines preset sound tones for success tones and theme melodies, and maintains the state of the playing sound. PlaySoundEffect is used to play a certain sound by updating the state, and UpdateSoundEffects keeps the sound that is currently playing progressing.

12 Challenges

- Prompting GPT to keep its answers in a certain format in order to create a parse-able response took a significant amount of trial and error. Drawing the components meant that the pin numbers and labels had to be stored in a list of exact format and length, and the chat gpt API tended to make slight adjustments to the format that would break this final step. Most commonly, it would break the list of pins into sections based on their function, or add additional titles or preambles to it's response. This variable format change was too difficult to parse, so a number of error messages were defined, then a re-prompting routine to throw out the incorrect response and request a new, correctly formatted answer.

- Adding the time scaling functionality to the oscilliscope proved very difficult, as it required the zero-crossing frequency calculations to adjust to a variable window period. Before adding the feature, it was trivial to experimentally determine the relationship between the zero crossings and the resultant frequency by adjusting a constant period, but the delay between each sample introduced a non-linear element. This necessitated the actual timing of the periods with SysTick, which introduced a level of higher accuracy, but also amplified the effects of component-frequency interference throughout the different frequencies.

- Managing storage was a consistent issue throughout the project. Discovering that the bitmaps could be read and stored in the file system was a massive step forward in resolving this issue, but the current codebase was still to large to be built and flashed to the RAM. This was resolved by modifying the addresses and length of the memory for the program in the linker configuration file. However, even this produced many errors, and the RAM_BASE and origin addresses that provided more memory space while not breaking the project had to be determined through trial and error.

13 Future Work

- Implement FFT in order to have more customizable waveforms in the function generator and oscilloscope.

- Run the GPT python code on a cloud based virtual machine, enabling constant polling with 0 actions needed from the user.

- Build a container for TI-OS in order to remove exposed wiring/connections, making it seem more like a computer.

14 Acknowledgment of AI Usage

Anthropic's Claude Sonnet 4 was used extensively throughout the development of this project. Uses include:

- Generation of boilerplate display code.

- Library usage and calculations required for bitmap generation and conversion helper programs.

- ChatGPT API usage in iot_shadow_client helper program.

- Code refactoring, labeling and organization.

- Collision checking and physics calculations in the video game application.

- Physics and project point rendering in the 3D cube simulation and servo arm application display.

- PWM constants and calculations for the servo arm and sound generation applications.

15 Bill of Materials

Component	Qty	Cost	Source
TI CC3200 LaunchPad	1	\$0	Lab
Pan-Tilt Servo	1	\$13	Alibaba
Analog Joystick	1	\$4	Alibaba
Buzzer	2	\$2	Alibaba
OLED Display	1	\$0	Lab
Buttons	1	\$1	Alibaba
Wires & Tools	—	\$5	Personal
Breadboard	1	\$5	Owned
GPT API Key	—	\$10	OpenAI
Total		**\$40**	