

Household Smart Scheduler

ECE 445 Design Document – Fall 2026

Project # 14

Amanda Schmitt, Maddy Sullivan, Shailee Patel

Professor: Arne Fliflet

TA: Tianda Huang

1. Introduction	1
1.1 Problem	1
1.2 Solution	1
1.3 Visual Aid	1
1.4 High-Level Requirements	1
2. Design	1
2.1 Physical Design	1
2.2 Block Diagram	1
2.2.1 Power Subsystem.....	1
2.2.2 Central Control Subsystem.....	2
2.2.3 RGB Display Subsystem	3
2.2.4 Hook Subsystem	4
2.2.5 Alert Subsystem.....	5
3. Design Verification	6
4. Costs.....	6
4.1 Labor	7
5. Ethical considerations	7
References.....	7

1. Introduction

1.1 Problem

People, especially college students, often have a calendar or planner system to keep track of their events and tasks. What they tend to struggle with, however, is leaving home on time to their various plans. While digital calendars can provide reminders about upcoming events, they focus specifically on event starting times rather than external factors such as time spent getting ready, available modes of transportation, belongings such as keys. In a household with several members following different chaotic schedules, keeping track of all of these responsibilities can become extremely difficult.

Our Household Smart Scheduler addresses this issue by implementing a shared physical board that syncs personalized calendar information, departure countdowns, and key storage in one location. Rather than checking different apps and calendars individually, users can instead view their next upcoming event and how much time remains before their required departure all in one place, regardless of their intended mode of transportation. Attached to the device will be a key-sensing system, which indicates whether a user's keys are still present on their designated hook and provides a push-notification to their phone when keys are left behind. Improving one's time management skills can decrease stress, contribute to safety, and promote one's general welfare. Leaving late can encourage individuals to rush, which can be especially dangerous while driving. Additionally, forgetting keys can create inconvenience, stress, and further delay. This system also considers social and cultural factors because it focuses on people with different work, school, and personal schedules individually. Economic consequences such as missed appointments, transportation splurges, and late arrivals are also reduced to avoid any unnecessary costs for individuals. Environmentally, this board allows members to plan appropriately if they would like to walk to specific locations and helps users avoid unnecessary trips back and forth caused by missing belongings. By combining calendar data, bus schedules, traffic, and other online data with a simple physical reminder board, our project makes everyday departures more organized, convenient, and predictable.

1.2 Solution

The Household Smart Scheduler will be a wall-mounted electronic display that provides personalized departure information for at least three household members. The main visual interface will consist of two adjacent 64x64 RGB LED matrix panels. Each user's section will display the amount of time remaining before they need to leave by walking, driving, or taking a bus. Each user will be represented by a distinct assigned color, while yellow and red will be reserved for departure warnings. The system will retrieve upcoming events over Wi-Fi and calculate when each user should leave using

$$T_{\text{departure}} = T_{\text{event}} - T_{\text{travel}} - T_{\text{buffer}}.$$

When a departure time is approaching, the corresponding countdown will change to yellow, and when the departure time is reached, it will change to red. A speaker-based audio alert will also activate when departure time is reached and play a stored music clip.

Each user will have a designated key hook with a switch that detects whether their keys are still hanging on the board. If a user's departure time is approaching and the keys remain on the hook, the system will provide a visual warning and send a notification to that user's phone. A Wi-Fi capable microcontroller on a custom PCB will coordinate calendar synchronization, departure-time calculations, key sensing, display control, phone notifications, and music playback through the speaker subsystem. The PCB will also contain power regulation, sensor interfaces, display connections, and supporting driver circuitry.

1.3 Visual Aid

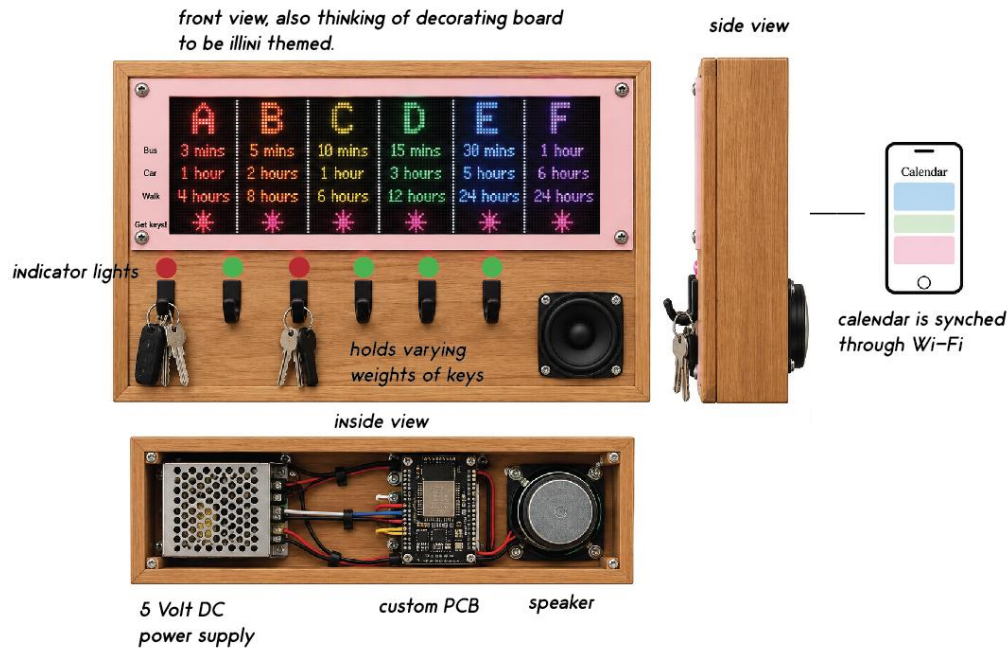


Figure #1 – Physical Appearance Concept for Design

The visual aid in Figure 1 shows how the Household Smart Scheduler would be used in a typical shared living environment. The main wall-mounted unit consists of two side by side RGB LED matrix panels that together provide a personalized section for each household member. Each user section displays their identifying initial and the amount of time remaining before they should leave for an upcoming event based on different transportation methods, such as walking, driving, or taking a bus. Directly below the display are the physical key hooks assigned to each user. When a user places their keys on the corresponding hook, the system can detect that the keys are present and use that information when determining whether a reminder should be issued before departure. The display therefore serves as a central household interface where users can quickly view both their schedule-related departure information and the status of their keys.

The Household Smart Scheduler also interacts with external systems that are not physically part of the wall-mounted device. Each user's calendar information is obtained wirelessly so that the system can determine the time and location of the user's next event and calculate an appropriate departure time. The scheduler can also communicate with the user's phone to provide a notification if the user is approaching their departure time while their keys are

still detected on the hook. When calculated departure time is reached, the display changes to a leave-now alert state, and the speaker plays a music clip to provide an audible reminder. Together, the display, key hooks, calendar connection, phone notification, and audio alert allow the system to provide a single physical location for helping household members prepare for and leave for scheduled events on time.

1.4 High-Level Requirements

To consider our project successful, our scheduler must fulfill the following:

- The system must simultaneously support at least three household members and continuously display separate bus, car, and walking departure countdowns for each active user. The displayed countdown values must update at least once every 60 seconds whenever valid scheduling data is available.
- For test events with known event times, travel time, and buffer time, the system must calculate the required departure time to within ± 60 seconds of the reference value. Once that departure time is stored locally, the system must enter the corresponding LEAVE_NOW state within ± 100 ms of the scheduled departure timestamp.
- The key-sensing and reminder system must correctly distinguish between key-present and key-removed states in at least 90 out of 100 controlled trials. When a user reaches a reminder's condition while their keys remain present, the system must initiate the corresponding phone-notification request within 1 second.
- When the system enters the LEAVE_NOW state, the visual alert and music-based speaker alert must each begin within 100 ms of the state transition, and the difference between their start times must not exceed 100 ms. The speaker output must measure between 65 dBA and 85 dBA at 1 m in a quiet indoor environment.

2. Design

2.1 Physical Design

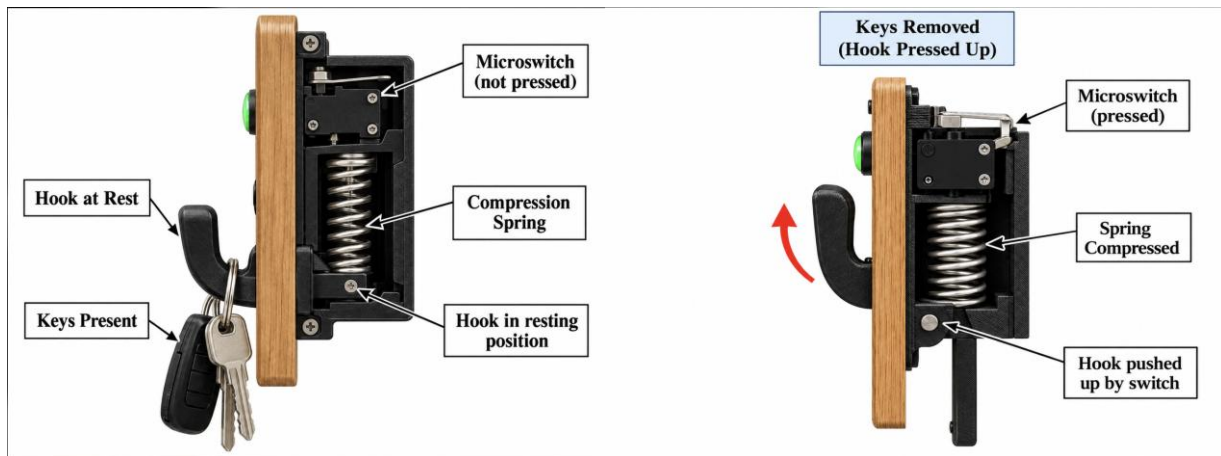


Figure #2: Physical Spring Mechanism for Hook Subsystem

The physical design of the Household Smart Scheduler centers on a wall-mounted enclosure that contains the two 64 x 64 RGB LED matrix panels in the upper portion of the device and places each user's key hook directly beneath the user's corresponding display area. Each hook is implemented as a spring-loaded mechanical assembly housed behind the front panel. When a set of keys is placed on the hook, the added weight causes the hook mechanism to move downward against the compression spring, leaving the microswitch unpressed and indicating that the keys are present. When the keys are removed, the spring returns the hook upward until the mechanism actuates the microswitch, indicating that the keys are absent. An indicator LED is positioned near each hook to provide a visual status signal associated with that user. The hook assembly is designed as a compact 3D printed mechanism so that the spring, microswitch, hook, and wiring can be contained behind the front panel while keeping only the hook and indicator LED visible to the user. This arrangement allows the physical placement of each hook to correspond directly with the scheduling information shown above it, while also creating a repeatable mechanical sensing method that can be duplicated for each household member.

2.2 Block Diagram

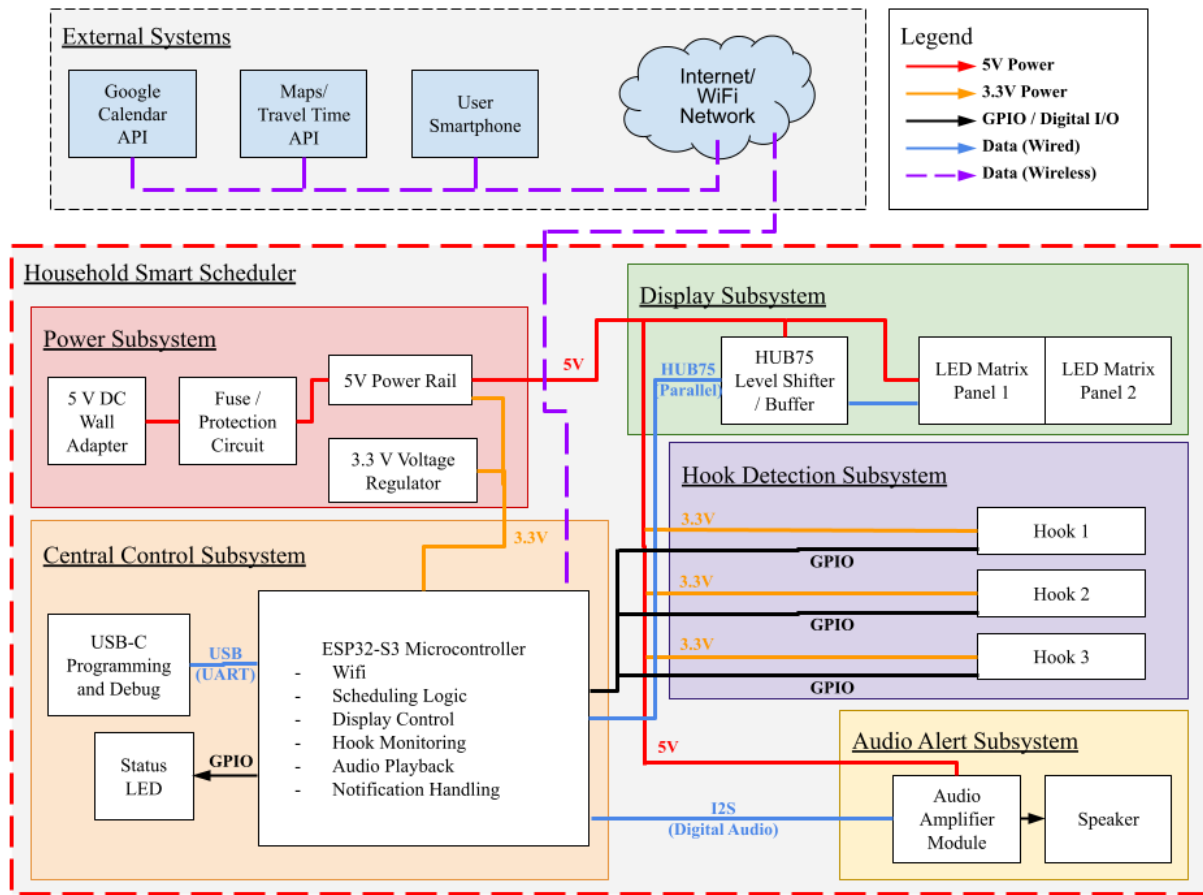


Figure #3 – Communication path between subsystems and external devices

Figure 3 shows the complete block-level architecture of the Household Smart Scheduler and the connections between its major subsystems. The system is divided into five primary subsystems: the Power Subsystem, Central Control Subsystem, Display Subsystem, Hook Detection Subsystem, and Audio Alert Subsystem. A regulated 5 V wall adapter provides the main power input. The 5 V rail directly powers the RGB LED matrices and the audio amplifier, while a 3.3 V regulator supplies the ESP32-S3 and other low-voltage logic. The ESP32-S3 acts as the central controller for the entire device. It communicates wirelessly over Wi-Fi with external services such as the calendar API, travel-time data source, and user smartphone notification service. Internally, the controller calculates departure times, maintains the scheduling departure times, maintains the scheduling state for each user, determines when alerts should occur, and coordinates the display, key hooks, and speaker.

The Display Subsystem consists of two 64 x 64 RGB LED matrix panels connected through the HUB75 interface and the required level-shifting or buffering circuitry. The ESP32 sends display data to this subsystem so that each household member can view their walking, driving, and bus departure countdowns. The Hook Detection Subsystem contains a separate microswitch and indicator of LED for each user. The microswitch signals are read by the ESP32 through GPIO inputs, allowing the controller to determine whether a user's keys are present, while GPIO outputs are used to control the corresponding hook LEDs. The Audio Alert Subsystem receives digital audio data from the ESP32 through an I2S connection and uses an audio amplifier to drive the speaker. When a user reaches the leave-now state, the ESP32 coordinates the RGB display and speaker so that the visual warning and selected music alert occur together. A USB-C programming and debugging connection is also included in the Central Control Subsystem for firmware development and system testing. This modular architecture allows each subsystem to be designed and verified independently before the full scheduler is integrated and tested as a complete system.

2.2.1 Power Subsystem

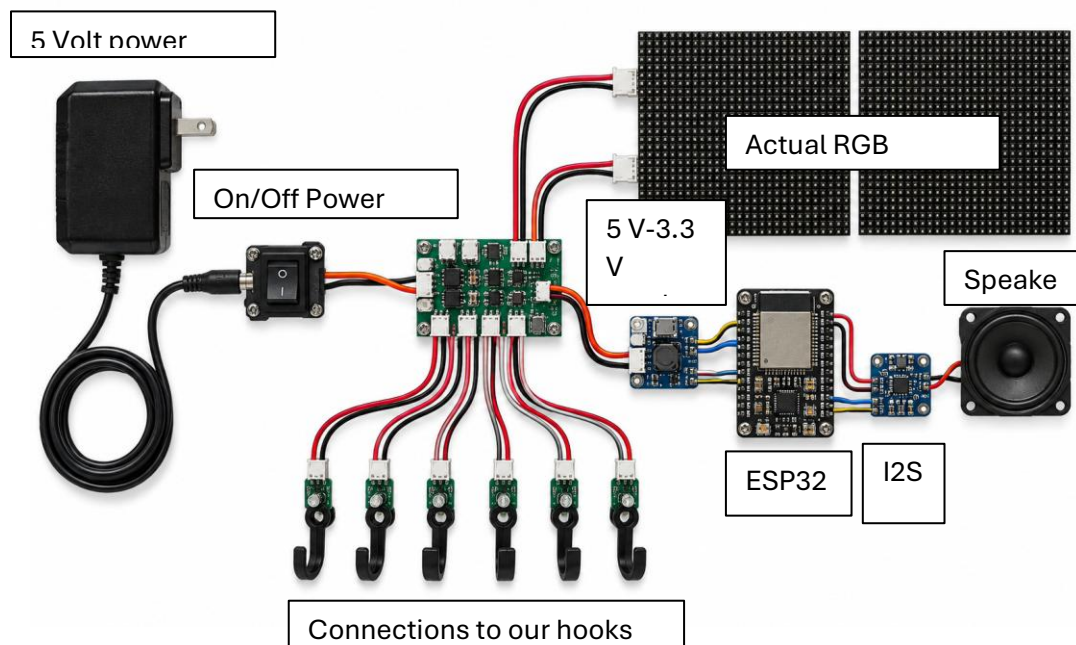


Figure #4 - Simplified Power Subsystem Diagram

The Power Subsystem is responsible for supplying a stable DC power source to all the electrical components operating within the Household Smart Scheduler. This subsystem will be powered

using an external and regulated 5 Volt DC power supply so that 120 Volts of AC power from the household does not interfere with our custom PCB. The 5 Volt input will be fed into our two RGB LED matrix panels and other components that take 5 Volt loads. Since we also have features that require 3.3 Volt power, a 3.3 Volt regulator on the control PCB will create the lower-voltage supply that is required by our ESP32 microcontroller and other control circuitry.

By keeping our 5 Volt and 3.3 Volt power domains separate, our RGB display, which requires relatively high current, can operate directly from the external supply while the ESP32 can function on its required lower operating voltage. All our subsystems will utilize common ground within our PCB so that any digital signals exchanged between parts have a consistent and reliable reference point.

The power architecture within this subsystem can thus be represented as follows:

$$5V_{IN} \rightarrow 5V_{RGB1}, \quad 5V_{RGB2}, \quad 5V_{HOOK}, \quad 5V_{ALERT}, \quad 3.3V_{REG} \rightarrow ESP32$$

The external power supply has a size requirement that is dependent on the maximum simultaneous current demand, not the average operating current. This is critical because our RGB LED matrices are the dominant electrical load, and their instantaneous power/current consumption is dependent on the icon/number being displayed, LED brightness, and the color illuminated in the pixels.

POWER BUDGET

The system will run based off a maximum current requirement modeled by:

$$I_{SYS,max} = I_{M1,max} + I_{M2,max} + I_{5V,aux} + I_{REG,in}$$

In this representation, $I_{M1,max}$ and $I_{M2,max}$ are the maximum currents drawn by RGB Matrix 1 and RGB Matrix 2, respectively. $I_{5V,aux}$ stands for the combined 5 Volt auxiliary loads, and $I_{REG,in}$ is the current that the 3.3 Volt regulator and its supplied electronics require proper function.

Using this information, the supply power required is therefor:

$$P_{SUPPLY,min} = V_{IN} I_{SYS,max}$$

Our design requires an external supply that pushes at least 10 Amps at 5 Volts, therefor having a maximum output power rating of:

The final budget for our current will be compared and verified against measured current or the datasheet standard current of our RGB panels, ESP32, hook LEDs, and alert hardware before any PCB fabrication. A supply margin will then be maintained between the 10 Amp supply limit and a calculated worst-case current so that any temporary load increases don't result in the 5 Volt rail collapsing.

3.3 VOLTAGE REGULATION

Our ESP32 operates on a standard 3.3 Volt supply; however, the system primarily takes energy from the 5 Volt load. Because of this, the control PCB must have a form of regulation between the power rail connecting the ESP32 to the external 5 Volt supply. Our current schematic makes use of an AP2112k-3.3 regulator for this conversion.

The approximate power dissipation is:

$$P_{REG} = (5.0 - 3.3)I_{OUT} \text{ for our 5 Volt to 3.3 Volt conversion, equaling } P_{REG} = 1.7I_{OUT}$$

Meaning that for every 100 mA of 3.3 Volt load, current production amounts to approximately:

$$P_{REG} = 1.7(0.100) = 0.170W$$

This relationship suggests that 0.170 Watts of heat is produced within the regulator. This is significant because the regulator must remain within its designated temperature rating when performing Wi-Fi communication and processor activity. Our final design therefore must verify that the maximum 3.3 Volt load current does not overheat our regulator.

The appropriate conversion efficiency for an ideal linear regulator, such as ours, is estimated by:

And when plugging in our 5 Volt to 3.3 Volt conversion, the equation becomes:

This implies that approximately 34% (100-66) of the power entering the regulator will dissipate rather than actually being delivered to the load at all. Since we still have a majority of system power being delivered to the 5 Volt RGB matrices, instead of the regulator, we only see this loss prevalent on our 3.3 Volt control rail.

VOLTAGE TOLERANCE

It is necessary for our power subsystem to keep both supply rails (5 Volt and 3.3 Volt) within the limits that allow all components to operate effectively. Throughout our design and verification process, we will have our 5 Volt rail held to ± 5 of its standard, or:

$$V_{5V} = 5.00V \pm 5 \text{ which amounts to: } 4.75V \leq V_{5V} \leq 5.25V$$

This requirement will also be applied to our 3.3 Volt source, but with a smaller ± 3 margin to account for the smaller magnitude of this power:

$$V_{3.3} = 3.30V \pm 3, \text{ or } 3.201V \leq V_{3.3} \leq 3.399V$$

TRANSIENT LOAD CONSIDERATIONS

We also need to ensure that our power system stays stable when our electrical load fluctuates rapidly and unexpectedly. Our RGB matrices can transition between low and high brightness display conditions, and our Wi-Fi transmission from the ESP32 often demands variable current values. A power supply that administers the correct voltage but constantly experiences voltage drooping can cause incorrect display visuals or even reset the ESP32.

The deviation of voltage during rapid current fluctuation can be determined using this capacitive relation:

$$I = C \frac{dV}{dt}, \text{ which is equivalent to } \Delta V = \frac{I\Delta t}{C} \text{ once rearranged}$$

We will need to consider this fundamental equation when selecting our local bypass and bulk capacitance values. Any small ceramic decoupling capacitors will be placed near supply pins of digital components in order to suppress any possible high-frequency disturbances. Additionally, any larger capacitances will be aligned near high current loads as to reduce any lower-frequency variation. Nevertheless, this equation will not be our only basis for selecting capacitance. We will also follow any recommended application circuits for our regulator/ICs as well as closely following the stability requirements we may find throughout our process.

GROUNDING AND POWER DISTRIBUTION

All of the components within our design must share common ground. Since we have devices that take 5 Volts and others that take 3.3 Volts, a common ground reference is crucial to maintaining proper communication. A method to help streamline this process is building our PCB with two ground panels on the top and bottom, with vias connecting components to these panels.

The matrix requires a decently high current, so return paths must be routed in a way that does not flow through the ESP32 and sensor ground paths since they are more susceptible to disturbances. When creating our PCB implementation, power and ground traces for the display must be larger than any typical GPIO traces, while satisfying:

$$I_{trace, rated} > I_{trace, max}, \text{ but allowing additional margin.}$$

POWER-ON BEHAVIOR

When the scheduler is turned on, the 5 Volt rail is then accessible by the display and voltage regulator. The 3.3 Volt rail must stabilize before any ESP32 operating can begin. The firmware we create will initialize the system in a safe start state where alerts are disabled until scheduling and sensor information is stable and established.

We plan our startup sequence to be as follows:

→ *CALENDAR SYNCHRONIZATION* → *NORMAL OPERATION*

This method prevents any invalid states during startups being registered as valid commands.

Requirements	Verification
The Power Subsystem should accept a regulated 5 V DC input and maintain the 5 V rail between 4.75 V and 5.25 V during normal and worst-case system operation.	Connect the completed scheduler to the regulated 5 V supply. Connect a DMM across the 5 V rail and system ground at the PCB power input. Measure and record the rail voltage under the following conditions: idle system, normal display operation, active Wi-Fi communication, maximum intended display brightness, active audio playback, and simultaneous display/Wi-Fi/hook/audio operation. Record all measured voltages in a table. Verify that every measurement remains between 4.75 V and 5.25 V.
The external power supply should provide sufficient current for both RGB matrices, the control circuitry, hook subsystem, and audio subsystem during simultaneous operation without exceeding its rated continuous-current limit.	Configure both RGB matrices to the highest-current test pattern used by the project while simultaneously enabling Wi-Fi communication, all hook LEDs, and audio playback. Place a current meter in series with the 5 V input or use a bench supply with current monitoring. Record maximum system current for at least 5 minutes of operation. Verify that the measured current remains below the rated continuous output of the selected supply. Record the maximum current and corresponding operating conditions.
The 3.3 V regulator should maintain the 3.3 V rail between 3.201 V and 3.399 V during all intended operating conditions.	Connect a DMM to the 3.3 V rail and system ground at the regulator output/test point. Measure the voltage during ESP32 idle operation, Wi-Fi communication, display updates, hook-state changes, notification processing, and audio triggering. Repeat during simultaneous system activity. Record all measurements in a table and verify that every measured value remains between 3.201 V and 3.399 V.
The RGB matrices, control PCB, Hook Detection Subsystem, and Audio Alert Subsystem should share a common ground reference.	With system power disconnected, place a DMM in continuity/resistance mode. Measure between PCB ground, ESP32 ground, RGB matrix ground, hook connector ground, and audio-amplifier ground. Verify electrical continuity between all ground points and record the measured resistance between the main PCB ground and each subsystem ground.

The 5 V and 3.3 V rails should remain within their specified voltage ranges during worst-case transient load changes.	Connect oscilloscope probes to the 5 V and 3.3 V test points. Configure the system to repeatedly transition between a low-load condition and the highest-current simultaneous operating condition, including high-brightness display activity, Wi-Fi communication, and audio playback. Capture minimum and maximum rail voltages during each transition. Repeat at least 20 transitions. Record the minimum and maximum values and verify that neither rail exits its permitted range.
No regulator or PCB power component should exceed its manufacturer-rated operating temperature during 30 minutes of continuous worst-case operation.	Operate the scheduler under the highest anticipated continuous electrical load for 30 minutes. Measure the AP2112K regulator and other major power components using a thermocouple or infrared thermometer at 5-minute intervals. Record temperature versus time in a table. Compare the maximum observed temperature against the appropriate manufacturer rating and verify that all components remain below their specified operating limits.

Figure #5 Requirements/Verification for Power Subsystem

TOLERANCE ANALYSIS

Our primary source of uncertainty in the Power Subsystem is the variation in external supply voltage, changes in load current, regulator output tolerance, PCB distribution losses, connector resistance, and transient current requirements. The voltage delivered to the system is equal to:

$$V_{LOAD} = V_{SUPPLY} - V_{DROP}$$

And the conductor and connector losses are approximately:

$$V_{DROP} = I_{LOAD}R_{PATH}, \text{ meaning that } V_{LOAD} = V_{SUPPLY} - I_{LOAD}R_{PATH}$$

This relationship demonstrates how power distribution resistance is especially important for the RGB matrices. Small amounts of resistance can result in distinct voltage drops when several amps of currents are drawn. Our PCB must thus maintain sufficiently low resistance so that each matrix receives proper voltage and remains under maximum load.

2.2.2 Central Control Subsystem

Our Central Control Subsystem will behave as the primary processing unit for the Household Smart Scheduler. We chose to use an ESP32 microcontroller because it includes a 2.4 GHz Wi-Fi connectivity system, digital GPIO interfaces, and the necessary processing capability to coordinate the display, key-sensing, and alert subsystems. The ESP32 is reliant on our 3.3 Volt supply that is regulated and generated in our Power Subsystem described above.

The main purpose of the ESP32 is to retrieve the user's next calendar event, calculate the specific departure time for the event, monitor/update the current time, track the user(s)' key statuses, and communicate the collected information to the RGB Display and Alert Subsystems. All Calendar data will be collected in periods over Wi-Fi rather than constantly so that Wi-Fi interruptions do not halt the scheduler's performance.

For each user and transportation method, the required departure time is calculated using this equation:

Where T_{event} represents the start time of the user's next calendar event, T_{travel} is the calculated estimated travel time, and T_{buffer} is an additional optional input for user preparation time. Once the departure time is determined, we calculate the remaining time using:

The ESP32 that we selected stores all calculated times locally. After the calendar information is retrieved from the user, the system does not have to continue waiting for more API information/requests to determine the ideal time that the user should leave the household. This circumvents network latency, making the departure times more reliable.

CONTROL STATES

We will organize each user's departure status as a sequence of states:

$$S \in \{NORMAL, APPROACHING, LEAVENOW, LATE\}$$

When in the NORMAL state, the display remains in the user's defined color. As the departure time draws near, the system transitions to APPROACHING, where the user information is now displayed in yellow. At the calculated departure time, the system enters the LEAVENOW state, changes the display to red, and the alert system becomes triggered. If the departure time passes and the user has not indicated that they have left, the system enters the LATE state.

The control logic is represented through these equations:

$$T_{remaining} > T_{warning} \rightarrow NORMAL$$

$$0 < T_{remaining} \leq T_{warning} \rightarrow APPROACHING$$

$$T_{remaining} \leq 0 \rightarrow LEAVE NOW$$

The exact warning time can be defined within the software depending on how long a user typically takes to get ready.

TIMING

Our intent is to ensure that the user leaves at the correct time, on the dot. Because of this, our most time-sensitive operation happens when the departure time is reached. The requirement we have decided is that the scheduler must enter the LEAVENOW state within 100 ms of the scheduled departure timestamp. Because the departure time is a local variable, this factor applies to the ESP32 for processing and subsystem communication rather than relying on Wi-Fi or calendar latency. Our design constraint is therefor:

$$t_{response} \leq 100ms$$

The calendar does not perform millisecond-level polling to update. Instead, we will have the ESP32 grab the time-current calendar information via Wi-Fi, with a target interval of every five minutes. Countdown information on the display, however, will update every 60 seconds to account for the minute-by-minute decrease on the board.

CALENDAR AND WI-FI OPERATION

We will utilize the minimum permissions necessary to access the calendar information. The ESP32 will only retrieve what is necessary to complete scheduling, such as the next event's starting time or current weather conditions, and then it will refer to this stored information rather than losing scheduling functionality. We consider a successful calendar update as one that replaces the stored event information after new data is validated to prevent overwriting any valid information.

KEY-SENSOR MONITORING

The Central Control Subsystem described later also received the current digital state of each key hook. The hook states will be interpreted as KEYPRESENT or KEYABSENT. When the user reaches the LEAVENOW state, the corresponding key hook will be checked to ensure the user has left. If it is discovered that the key is still hanging on the device, then the controller initiates the phone-notification request described in our Alert Subsystem. Our maximum allowed notification initiation delay is:

$$t_{notification} \leq 1s$$

This one second window is for the detection of the reminder condition to the notification request. This does not describe the user's phone to receive the notification within one second because Wi-Fi latency and external factors can toggle this duration.

Requirements	Verification
The controller should maintain independent scheduling information for at least three users simultaneously.	Create three test users with different event times, travel times, and buffer times. Load all three users into the scheduler simultaneously. Use the serial debug interface to record each user's stored event, departure, and remaining-time values. Modify only one user's event information and verify that the other two users' stored values remain unchanged. Record expected and measured information for all users in a table.
While an Internet connection is available, the controller should retrieve each active user's next calendar event and check for updated calendar information at least once every 5 minutes.	Create known upcoming events for at least three test users with predetermined start times and locations. Connect the ESP32 to Wi-Fi and monitor calendar-update timestamps through serial debugging. Compare the retrieved event information with the source calendar. Continue monitoring for at least 15 minutes and modify one event during the test. Record every update time and the retriever event information. Verify that the controller checks for updates at intervals no greater than 5 minutes and correctly replaces stored information after a valid update.
For known T_{event} , T_{travel} , and T_{buffer} , the calculated departure time should be within ± 60 seconds of the reference value.	Prepare at least 20 sets of known event, travel, and buffer values. Independently calculate $T_{\text{departure}} = T_{\text{event}} - T_{\text{travel}} - T_{\text{buffer}}$ for each set. Enter the same values into the ESP32 and record its calculated departure timestamp through serial debugging. Compute the error between the ESP32 value and the reference value. Record all inputs, outputs, and errors in a table. Verify that every absolute error is ≤ 60 seconds.
Once a valid departure timestamp is stored locally, the controller should enter the corresponding LEAVE_NOW state within ± 100 ms of that timestamp.	Configure one debug GPIO to toggle at the reference departure timestamp and another GPIO to toggle when the firmware enters LEAVE_NOW. Connect both signals to an oscilloscope or logic analyzer. Set a departure time several seconds in the future and capture both transition. Repeat for 100 trials. Record the timing error for every trial and verify that each value lies between -100 ms and $+100$ ms.
When the key-reminder condition is satisfied while the corresponding keys remain present, the controller should initiate the phone-notification request within 1 second.	Configure T_{reminder} several seconds in the future and place the corresponding keys on the hook. Use timestamped serial logging or debug GPIOs to record the instant the reminder condition becomes true and the instant the notification request is initiated. Repeat for at least 20 trials. Record the measured latency for every trial and verify that every request is initiated within 1 second. Repeat with keys removed and verify that no request is initiated.
A temporary loss of Wi-Fi after a valid departure timestamp has been stored should not prevent the locally	Allow the controller to retrieve and store a valid test event and departure timestamp. Disconnect Wi-Fi before the departure time occurs. Monitor the controller through serial debugging

scheduled LEAVE_NOW transition.	and the LEAVE_NOW debug GPIO. Verify that the controller still enters LEAVE_NOW at the stored timestamp within the ± 100 ms requirement. Repeat for at least 10 trials and record the result of each trial.
---------------------------------	---

Figure #6 Requirements/Verification for Central Control Subsystem

TOLERANCE ANALYSIS

For this subsystem, our primary timing uncertainty is reliant on network-dependent operations and locally executed operations. Calendar retrieval can face obstacles such as Wi-Fi and API latency, so network communication is not responsible for triggering departure. Instead, we calculate the departure time beforehand and store it locally.

$$\Delta T_{departure} = \Delta T_{event} + \Delta T_{travel} + \Delta T_{buffer}$$

We use the event time from the user's calendar, but travel time and user buffer time contribute uncertainty for accuracy. Because the travel conditions may change rapidly and unexpectedly, our 60-second calculation requirement only applies when the event time, travel time, and buffer are known test inputs. We have determined our timing error for the locally triggered LEAVENOW event to be:

$$e_t = |T_{actual} - T_{departure}|, \text{ and our system satisfies timing when: } e_t \leq 100ms$$

2.2.3 RGB Display Subsystem

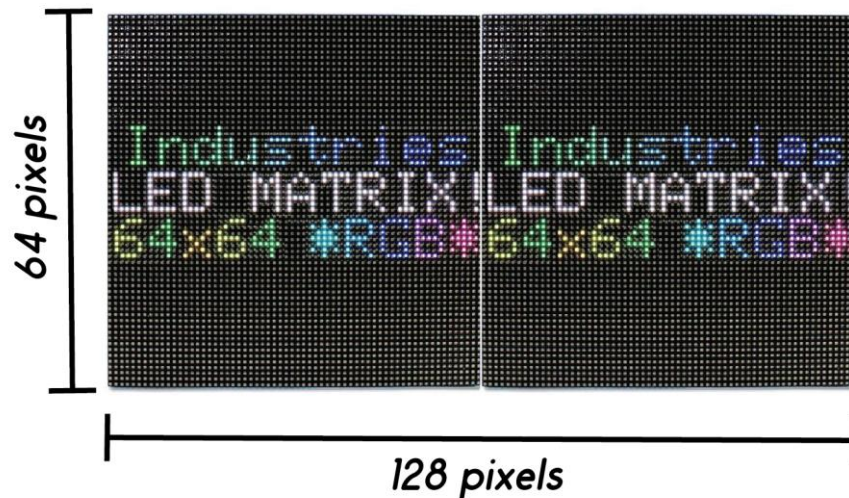


Figure # - Physical Characteristics of RGB Display Subsystem

The RGB Display Subsystem oversees the primary visual aspect of our Household Smart Scheduler. It is responsible for an accurate interface for its users and consists of two adjacent 64x64 RGB LED matrix panels to produce a combined display of 128x64 pixels. Each matrix consists of 4,096 pixels, amounting:

Since the panels are directly next to one another; they function as one visual interface. Each user will have their own section within the board containing their name and countdown information for the three transportation methods of bus, car, and walking. These options allow the user to easily choose what method to take before leaving without the use of an external calendar, transportation, or navigation application.

The ESP32 controls the matrices and alters the departure information determined by the Central Control Subsystem into text, numbers, and colors displayed on the board. The display then no longer has to calculate the departure times. Instead, it just receives the current state and countdown information from the Central Control Subsystem.

USER DISPLAY ORGANIZATION

Each user is given a distinct new operating color that is not yellow or red to not confuse the operating colors of specific states. Yellow and red are reserved globally for corresponding departure warnings, so their meanings stay consistent apart from the user's standard color.

The display follows three primary visuals: "Normal" is presented through the user's information being displayed in their assigned color, "Approaching" is represented when relevant information transitions from user color to yellow to indicate impending departure, and "Leave Now/Late" are shown in red to prompt the user to leave immediately since the departure time has been reached or passed. This color system ensures that the user can easily interpret their state without having to read the entire screen.

DISPLAY UPDATE TIMING

Countdown values presented to the user will update at least once every 60 seconds in order to make the system-level requirement. Our display must respond significantly faster when the LEAVENOW state is reached, however, to reflect the sense of urgency. Our display response time is reflected as

In this representation, t_{CC} is the time needed for the Central Control Subsystem to activate and issue the new display state, t_{UD} is the time needed to update the display, and t_{TV} is the time until updated information is transferred and visible on the display. We do not plan on holding these values to millisecond data before implementation; instead, we will measure them during the testing and verification process. The main constraint that we do want to enforce, however, is the time between the beginning of the visual alert and audible alert:

DISPLAY REFRESH AND FLICKER

In order for the RGB display to reflect a clear and stable display, continuous refreshing is required. The display-control software needs to refresh the matrices at a quick enough rate so that individual pixel changes are not distinguishable by any viewers. We will calculate the period and refresh frequency to be related by:

As an example, we can update this equation if testing reveals that the display operates at a refresh rate of 100 Hz, causing the period to become:

DISPLAY COMMUNICATION

The matrices that we have chosen to utilize a HUB75-style interface, which we will control with the ESP32. Data from the display will be transmitted from the controller digitally, but the matrices will receive their own power from the external 5 Volt Power Subsystem.

Due to the RGB matrices pulling more current than the ESP32 by a significant amount, we must power the matrices directly from the 5 Volt supply rather than through the 3.3 Volt regulator. Our ESP32 is only responsible for controlling signals but will share the same ground as the display and other circuitry to maintain common voltage reference.

Requirements	Verification
The display should present scheduling information for at least three active users simultaneously in independent user regions.	Load three users with different initials and countdown values. Command all three to be shown simultaneously. Visually inspect the combined display and verify that each user appears in a separate region with no overlapping text or numbers.

	Photograph the display and record each expected and displayed value.
Each active user region should display separate departure information for bus, car, and walking transportation modes.	Assign known bus, car, and walking countdown values to three test users. Render the information on the display. Compare all displayed values with the values stored by the controller. Record the expected and displayed values in a table. Verify that each user has all three transportation modes displayed correctly.
Each user should have a distinct normal operating color, while yellow and red should be reserved for APPROACHING and LEAVE_NOW/LATE states.	Assign each test user a unique normal color. Display the users in NORMAL and record the observed color. Force each user individually into APPROACHING and verify that the appropriate information becomes yellow. Force each user into LEAVE_NOW and verify that the appropriate information becomes red. Photograph each state and record expected versus observed colors.
Displayed countdown values should update at least once every 60 seconds whenever valid scheduling information is available.	Load countdown values greater than 5 minutes and record the display using timestamped video or a stopwatch. Observe at least five consecutive visible countdown changes. Record the elapsed time between updates. Verify that every update interval is ≤ 60 seconds.
After the controller enters LEAVE_NOW, the corresponding visual alert should begin within 100 ms.	Configure an ESP32 debug GPIO to toggle when LEAVE_NOW is entered. Place a photodiode or light sensor over a display pixel that changes during the alert. Connect the GPIO and sensor output to separate oscilloscope channels. Trigger LEAVE_NOW and measure the time from the state-transition GPIO edge to the detected visual change. Repeat at least 50 times and record each latency. Verify that all measured values are ≤ 100 ms.
The difference between visual-alert onset and audio-alert onset should not exceed 100 ms.	Place a photodiode over a pixel that changes during LEAVE_NOW and place a microphone/audio detector near the speaker. Connect both sensor outputs to an oscilloscope or logic analyzer. Trigger LEAVE_NOW and measure $\Delta t_{AV} = T_{\text{audio}} - T_{\text{display}} $. Repeat the test at least 10 times and record the measured time difference for each trial. Verify that every measured value is less than or equal to 100 ms.
The two physical RGB panels should operate as one logical display without corrupted text, visible loss of synchronization, or unintended resets during a 30-minute test.	Load a test pattern containing text, colored regions, and graphics spanning the physical boundary between the two panels. Operate the display continuously for 30 minutes while periodically changing displayed information. Observe for corrupted text, incomplete updates, synchronization loss, or resets. Record all observed failures. Requirement passes if zero such failures occur during the test.
The six-user target layout should fit within the 128 x 64 display without overlap between adjacent user regions.	Configure six user regions with initials and bus/car/walking values. Render all six simultaneously. Inspect the display for overlap or clipping of information between adjacent regions. Photograph the six-user display and record whether each region remains readable. This verifies the six-user design goal; the minimum project-success requirement remains three users.

Figure #7 Requirements/Verification for RGB Display Subsystem

TOLERANCE ANALYSIS

The tolerances to focus on our display revolve around timing, physical alignment of visual characters, readability, and power variation.

Our most important synchronization lies between visual and audible alerts. We will define the synchronization error as follows:

$$e_{sync} = |t_{visual} - t_{audio}|, \text{ which is satisfied when } e_{sync} \leq 100ms$$

We can measure and confirm this requirement directly during verification without having to rely on any estimated software execution times, making this a more convenient option.

Our next tolerance concern is the actual physical alignment of the board. Due to our usage of two separate LED matrices, it can be easy to misalign the panels and make information near the boundary difficult to read. Our design enclosure must be tight and precisely aligned in a fixed position, and our written code will need to avoid placing critical characters across the boundary of these two interfaces.

Power tolerance is our final concern for this subsystem. The display brightness and stability are reliant on the 5 Volt supply remaining within its voltage limits that we established in the Power Subsystem. This means that our board must function on this range of values:

2.2.4 Hook Subsystem



Figure #8: Simple Visual of Light Indicators and Hooks

The Hook Detection Subsystem is responsible for determining whether each household member's keys are currently stored on the Household Smart Scheduler and for providing this information to the Central Control Subsystem. The system is required to support at least three active users, while the physical hook architecture is being designed with the goal of supporting up to six repeated hook assemblies. Each user is assigned a spring-loaded key hook containing a 3D-printed moving structure, compression spring, lever microswitch, and indicator LED.

The hook mechanism uses additional gravitational force produced by a set of keys to distinguish between the key-present and key-removed states. When no keys are placed on the hook, the restoring force of the compression spring pushes the moving hook assembly upward. In the fully returned position, the mechanism presses the microswitch actuator. The ESP32 therefore interprets the pressed microswitch state as keys removed. When a set of keys is placed on the hook, the additional weight pulls the moving assembly downward against the restoring force of the compression spring. Once the assembly has moved sufficiently downward, it no longer presses the microswitch actuator. The unpressed switch state is therefore interpreted as keys present. This operating behavior matches the physical design shown earlier in the document.

Each microswitch will be connected to a digital GPIO input on the ESP32 so that the controller can determine the current state of the corresponding hook. A separate indicator LED associated with each hook will provide local visual feedback. The ESP32 will control the LED through a digital output based on the detected hook state. The indicator LED will illuminate when the corresponding keys are detected as present on the hook and remain off when the keys are removed. The same key-presence information will also be passed to the reminder logic. If the Central Control Subsystem determines that the current time has reached the configured key-reminder timestamp while the hook still indicates that the user's keys are present, the system will initiate a phone-notification request.

The Hook Detection Subsystem is one of the most mechanically sensitive portions of the project because correct operation depends on several real physical parameters rather than only digital logic. These parameters include the spring constant, spring preload, moving hook mass, minimum and maximum key mass, friction between the sliding parts, microswitch actuation force, and available mechanical travel. The spring must be strong enough to return the empty hook to the microswitch when the keys are removed, but it must not be so stiff that a light set of keys is unable to pull the hook away from the microswitch.

This subsystem directly supports the third high-level requirement by determining whether a user's keys are present and providing the key-state information used to initiate the phone-notification request.

Requirements	Verification
--------------	--------------

The key-sensing mechanism should correctly distinguish between key-present and key-removed states in at least 90 out of 100 controlled trials.	Connect the completed hook to the ESP32. Alternate the physical hook between key-present and key-removed conditions until the 100 classifications have been performed. For every trial, record the actual physical state and the state reported by the ESP32 through serial debugging. Calculate the number of correct classifications. Verify that at least 90 out of 100 are correct.
With no keys present, the spring should return the hook sufficiently upward to actuate the microswitch.	Manually move the hook to its lowered key-present position and release it without providing assistance. Use the ESP32 serial output or a DMM in continuity mode to determine whether the microswitch is actuated after release. Repeat the return operation 100 times. Record each successful or unsuccessful return and verify reliable actuation of the switch.
A keyring at the selected minimum supported mass shall move the hook sufficiently downward to release the microswitch and produce the key-present state.	Measure the minimum supported test keyring using a digital scale. Allow the empty hook to return fully to its key-removed position. Place the test keyring on the hook without manually assisting its movement. Read the microswitch state through the ESP32. Remove the keys and repeat for 100 trials. Record the result of each trial and verify that the minimum supported mass reliably causes the key-present state.
A keyring at the selected maximum supported mass should not exceed the mechanism's allowed travel or permanently deform the spring, hook, or enclosure.	Measure the unloaded hook position using a caliper. Apply the selected maximum supported key mass and measure the downward displacement. Verify that the mechanism does not exceed its designed travel limit or hard stop. Remove the mass and remeasure the unloaded position. Repeat the load/unload cycle 100 times. Inspect the spring and 3D-printed parts for permanent deformation or damage and record all measurements.
The hook indicator LED shall illuminate when keys are detected as present and should turn off when the keys are removed.	Begin with the hook empty and record the LED state. Place a representative keyring on the hook and record the LED state after the mechanism settles. Remove the keys and record the LED state again. Repeat the sequence 20 times. Verify that the LED is ON in every key-present state and OFF in every key-removed state. Record expected and observed states in a table.
When the configured reminder timestamp is reached while the keys remain present, the corresponding phone-notification request should be initiated within 1 second.	Place keys on the selected user's hook and verify that the controller reports KEY_PRESENT. Configure a known reminder timestamp several seconds in the future. Record the instant the reminder condition becomes true and the instant the notification request is initiated using timestamped logging or GPIO markers. Repeat at least 20 times and record the measured latency. Verify that every notification request begins within 1 second.

Figure #9 Requirements/Verification for Hook Subsystem

TOLERANCE ANALYSIS:

The Hook Detection Subsystem is expected to be one of the most challenging mechanical portions of the Household Smart Scheduler because its operation depends on the balance between

gravitational force, spring force, friction, and microswitch actuation force. The fundamental relationship governing the spring is Hooke's Law,

$$F_s = k(x_0 + x)$$

where F_s is the spring force, k is the spring constant, x_0 is the initial spring compression/preload, and x is the spring displacement from its unloaded or preloaded position. The downward gravitational force caused by the moving mass is

$$F_g = mg$$

where m is the supported mass and $g = 9.81 \text{ m/s}^2$. For a simplified static case with negligible friction, equilibrium occurs when

$$kx = mg$$

which gives

$$x = mg / k.$$

The actual mechanism must account for more than this ideal relationship. Let m_h represent the mass of the moving hook assembly, $m_{k,min}$ represent the minimum key mass that the system is required to detect, $m_{k,max}$ represent the maximum supported key mass, F_f represent the worst-case sliding friction force, F_{sw} represent the force required to actuate the microswitch, and x_{sw} represent the mechanical travel required to change the switch state. In the no-key condition, the spring must provide enough upward restoring force to overcome the hook weight, friction, and switch actuation force. Therefore, the following condition must be satisfied:

$$F_{spring,no-key} > m_h g + F_f + F_{sw}.$$

The requirement ensures that, after the keys are removed, the hook reliably returns upward and presses the microswitch.

The opposite condition must also be satisfied when the lightest supported set of keys is placed on the hook. The total downward force produced by the moving hook and keys must be sufficient to overcome the upward spring force and friction so that the hook travels far enough away from the microswitch.

Therefore,

$$(m_h + m_{k,min})g > F_{spring} + F_f.$$

In addition to overcoming the force balance, the resulting displacement must exceed the travel needed to release the switch:

$$x_{keys} - x_{no-key} \geq x_{sw} + x_{margin}.$$

The additional term x_{margin} provides mechanical margin for manufacturing tolerances, 3D-print dimensional variation, wear, and imperfect alignment.

Spring manufacturing tolerance must also be considered. If the selected spring has a nominal spring constant k_{nom} with fractional tolerance δ_k , then the minimum and maximum possible spring constants are

$$k_{\min} = k_{\text{nom}}(1 - \delta_k)$$

and

$$k_{\max} = k_{\text{nom}}(1 + \delta_k).$$

The stiffest possible spring, $k_{\max}$, creates the most difficult case for detecting a lightweight keyring because it produces the greatest restoring force for a given displacement. The softest possible spring, $k_{\min}$, creates the most difficult case for returning the unloaded hook to the microswitch because it produces the smallest restoring force. Both extreme cases must therefore be evaluated before the spring is selected. The maximum supported key mass must also be considered. For the heaviest expected keyring, the displacement is approximately

$$x_{\max} = (m_h + m_{k,\max})g / k_{\min}$$

before additional friction effects are included. This displacement must remain less than the maximum mechanical travel available in the 3D-printed housing:

$$x_{\max} < x_{\text{travel},\max}.$$

If this condition is not satisfied, a mechanical hard stop will be required to prevent the spring or microswitch assembly from being overstressed. Final values for m_h , $m_{k,\min}$, $m_{k,\max}$, F_{sw} , k_{nom} , spring tolerance, and mechanical travel will be obtained from the selected components and measured prototype. These values will then be inserted into the preceding equations to verify that both the no-key and key-present states remain distinguishable under worst-case tolerances. Experimental verification will be performed with repeated trials using key masses at the minimum, nominal, and maximum supported values.

2.2.5 Alert Subsystem



Figure #10: Simple Visual of Speaker and Text Alerts

The Audio Alert Subsystem is responsible for providing an audible notification when a household member reaches the calculated LEAVE_NOW state. The subsystem uses the ESP32 to access stored

digital audio data, transmit that data through an I2S interface, and drive a speaker through a dedicated audio amplifier. Because a passive speaker requires significantly more current than an ESP32 GPIO pin can provide directly, the amplifier is necessary to convert the low-power digital audio signal into an electrical output capable of driving the speaker at the required volume.

The audio used for the alert will be stored locally rather than streamed from the Internet. A short music clip will be converted into an uncompressed PCM-based WAV file and stored in the ESP32's available flash memory if sufficient storage is available. Storing the file locally allows the system to begin playback without depending on Internet latency or network availability at the time of departure. If the final audio file exceeds the available internal storage, an external storage option may be considered, but local flash storage is preferred because it reduces additional hardware and minimizes playback-start latency. The selected audio clip will either be created by the project team or obtained from a source that permits its use.

When the Central Control Subsystem determines that a user has reached the stored departure timestamp, the ESP32 will transition that user into the LEAVE_NOW state. The ESP32 will then begin reading the stored PCM samples and transmitting them through the I2S peripheral. The I2S interface will provide the digital audio data to an I2S-compatible amplifier, which will generate the amplified electrical waveform required to drive the speaker. The amplifier will operate from the system's 5 V rail and share a common ground with the ESP32 and the remainder of the scheduler for electronics.

The Audio Alert Subsystem will operate in coordination with the RGB Display Subsystem. When the LEAVE_NOW state is entered, the corresponding user's display section will change to its visual warning state while the music alert begins. The difference between the start of the visual and audible alerts must not exceed 100 ms. The audio alert will continue for the duration of the stored clip or another predetermined alert interval and will then terminate automatically. The final speaker output must measure between 65 dBA and 85 dBA at 1 m in a quiet indoor environment so that the alert is noticeable without being excessively loud.

This subsystem directly supports the fourth high-level requirement by providing the audible component of the LEAVE_NOW alert within the required timing and sound-pressure limits.

Requirements	Verification
The Audio Alert Subsystem should reproduce a locally stored music clip through the speaker without requiring Internet communication at playback time.	Store the selected WAV/PCM music file in the ESP32's local memory. Disable the system's Wi-Fi connection before triggering the alert. Trigger audio playback using the firmware and verify the complete music clip plays through the speaker. Repeat 10 times and record playback success/failure for each trial. Requirement passes if all 10 offline trials successfully produce the stored alert.

The ESP32 should transmit the digital audio stream to the amplifier through the I2S interface without a firmware buffer underrun during normal playback.	Enable firmware logging for I2S or audio-buffer errors. Trigger the stored music clip and allow it to play completely. Repeat playback 20 consecutive times. Record any buffer-underrun/error flags and any observed interruption in playback. Verify that all 20 trials complete with zero reported buffer underruns.
Music playback should begin within 100 ms of the corresponding LEAVE_NOW state transition.	Configure a debug GPIO to toggle when the ESP32 enters LEAVE_NOW. Place a microphone or audio detector near the speaker and connect its output and the debug GPIO to separate oscilloscope channels. Trigger LEAVE_NOW and measure the time from the GPIO transition to the first detected audio output. Repeat at least 50 trials. Record each startup latency and verify that all values are ≤ 100 ms.
The difference between visual-alert onset and audio-alert onset should not exceed 100 ms.	Place a photodiode/light sensor over a display pixel that changes during LEAVE_NOW and a microphone/audio detector near the speaker. Connect both sensors to an oscilloscope or logic analyzer. Trigger the alert and measure $\Delta t_{AV} = T_{\text{audio}} - T_{\text{display}} $. Repeat the test 10 times and record the measured time difference for each trial. Verify that every measured value is ≤ 100 ms.
The speaker should produce a sound-pressure level between 65 dBA and 85 dBA at a distance of 1.00 meter in a quiet indoor environment.	Mount the speaker in its intended final orientation. Use a tape measure to place an A-weighted sound-level meter $1.00 \text{ m} \pm 2 \text{ cm}$ from the speaker. Record ambient sound level, then play the selected alert at the final operating volume and record the measured sound level. Repeat five times. Present the five values and their average in a table. Verify that every alert measurement is between 65 dBA and 85 dBA.
Audio playback shall terminate automatically after the selected clip or configured alert interval has completed.	Use an audio clip with a known duration or configure a known alert timeout. Trigger playback and record its start and stop timestamps using firmware logging or a microphone/audio detector. Repeat 20 times. Record the expected and actual termination times. Verify that playback terminates automatically in every trial and does not continue indefinitely.
Audio playback should not cause the 5 V rail to leave the allowable 4.75 - 5.25 V range while the other major system loads are active.	Operate both RGB panels, enable Wi-Fi activity, and trigger the audio alert simultaneously. Connect an oscilloscope or DMM to the 5 V rail at the amplifier/PCB supply point. Record minimum and maximum 5 V rail voltage during at least 20 alert activations. Verify that the rail remains between 4.75 V and 5.25 V for every trial.

Figure #11 Requirements/Verification for Alert Subsystem

TOLERANCE ANALYSIS

The Audio Alert Subsystem must satisfy three main technical constraints: sufficient audio-data throughput, acceptable playback-start latency, and adequate electrical/acoustic output. These constraints are important because the subsystem can only function correctly if the ESP32 can continuously supply audio samples, begin playback within the required timing window, and drive the speaker loudly enough to meet the project's sound pressure requirement.

For PCM audio, the required data rate is determined by the sampling frequency, number of bits per sample, and number of audio channels:

$$R_{\text{audio}} = f_s N_b N_c$$

Where f_s is the sampling frequency, N_b is the number of bits per sample, and N_c is the number of channels. The corresponding storage requirement for an audio clip of duration t_{clip} is

$$S_{\text{clip}} = (R_{\text{audio}} t_{\text{clip}}) / 8.$$

These equations will be used to select an audio format that provides sufficient sound quality while remaining within the available local memory of the ESP32. The audio file will be stored locally rather than streamed over the Internet so that the alert can begin without depending on network latency or availability. A WAV file containing uncompressed PCM audio is preferred because it allows predictable playback timing and avoids the additional processing delay associated with decoding compressed audio formats.

Once playback begins, the audio subsystem must continuously provide new samples at the selected sampling frequency. The interval between consecutive audio samples is

$$T_s = 1 / f_s.$$

The firmware and I2S peripheral must therefore sustain this sample interval without interruptions. A failure to provide samples at the required rate could result in audible distortion, gaps, or buffer underruns. The second major tolerance is playback-start latency. The total delay between the LEAVE_NOW state transition and the production of audible sound can be modeled as

$$t_{\text{audio,start}} = t_{\text{dispatch}} + t_{\text{storage}} + t_{\text{buffer}} + t_{\text{I2S}} + t_{\text{amp}}.$$

Here, t_{dispatch} is the firmware response time after the state transition, t_{storage} is the time required to access the first audio samples, t_{buffer} is the time needed to fill the initial playback buffer, t_{I2S} is the delay required to begin the I2S transmission and t_{amp} is the amplifier response time. The design must satisfy

$$T_{\text{audio,start}} \leq 100 \text{ ms}.$$

The buffer itself contributes to a predictable timing term. If the playback buffer contains N_{buf} samples, then

$$t_{\text{buffer}} = N_{\text{buf}} / f_s.$$

This creates an engineering tradeoff. Increasing the buffer size improves resistance to temporary processor delays and reduces the chance of an audio underrun, but it also increases memory usage and may increase startup latency. The final buffer size will therefore be chosen so that playback remains stable while preserving sufficient margin below the 100 ms timing requirement.

The visual and audio alerts must also remain synchronized. Their activation difference can be defined as

$$\Delta t_{\text{AV}} = | T_{\text{audio}} - T_{\text{display}} |.$$

The design requirement is

$$\Delta t_{AV} \leq 100 \text{ ms.}$$

This value will be measured during verification using GPIO timing markers generated when the firmware begins the display update and audio playback sequence. The timing signals can then be observed simultaneously using an oscilloscope or logic analyzer.

The speaker and amplifier must also satisfy the electrical power requirements of the system. For a speaker with nominal impedance R_L , the approximate RMS output power is

$$P_{\text{speaker}} = V_{\text{RMS}}^2 / R_L.$$

The corresponding RMS voltage and current are

$$V_{\text{RMS}} = \sqrt{(P_{\text{speaker}} R_L)}$$

and

$$I_{\text{RMS}} = \sqrt{(P_{\text{speaker}} / R_L)}.$$

These relationships determine the voltage and current capability required from the amplifier and demonstrate why the passive speaker cannot be driven directly from an ESP32 GPIO pin. The amplifier must instead provide sufficient output current while operating from the system's 5 V rail.

If the amplifier has efficiency η_{amp} , its approximate required input power is

$$P_{\text{in}} = P_{\text{speaker}} / \eta_{\text{amp}}$$

and its approximate supply current is

$$I_{\text{amp,in}} = P_{\text{speaker}} / (\eta_{\text{amp}} V_{\text{supply}}).$$

The final value of $I_{\text{amp,in}}$ will be included in the Power Subsystem's worst-case current budget so that the audio alert cannot cause the 5 V rail to collapse while the RGB matrices and other electronics are operating simultaneously.

Finally, the electrical speaker power must produce sufficient acoustic output. If the selected speaker provides a sensitivity rating $S_{1W,1m}$, the approximate sound-pressure level at 1 m can be estimated by

$$\text{SPL}_{1m} \approx S_{1W,1m} + 10 \log_{10}(P_{\text{speaker}} / 1 \text{ W}).$$

The effect of measurement distance can be approximated by

$$\text{SPL}_2 = \text{SPL}_1 - 20 \log_{10}(r_2 / r_1).$$

These acoustic equations will be used to select the speaker and amplifier combination, but the final requirement of 65 dBA to 85 dBA at 1 m will be verified experimentally because enclosure geometry, wall reflections, mounting, and room acoustics will affect the measured value. This subsystem therefore combines timing, digital-audio throughput, electrical power, and acoustic-output constraints, while still remaining less tolerance-sensitive than the mechanical Hook Detection Subsystem.

3. Design Verification

VERIFYING THE POWER SUBSYSTEM

TEST EXAMPLE #1:

Connect our finished scheduler to the 5 Volt power supply. Measure the voltage between 5 Volt and ground using a functioning digital multimeter.

Situations to record the voltage under:

- Idle System
- Normal Display Operation
- Wi-Fi Communication
- Display set to High Brightness
- Simultaneous Display, Wi-Fi, Hook, and Alert Operation

During each test, the voltage needs to be between 4.75 Volts and 5.25 Volts, as specified.

TEST EXAMPLE #2:

Do the exact same test as 'TEST EXAMPLE #1', but now measuring at the 3.3 Volt test points.

The subsystem passes if the measured voltages appear between 3.201 Volts and 3.399 Volts.

TEST EXAMPLE #3:

Test the worst-case load. Configure both of the RGB panels to the highest-current display possible and enable Wi-Fi communication and alert output at the same time. Measure the 5 Volt test point(s), the 3.3 Volt test point(s), and the current input to ensure they remain within their specified limits, and the input current is less than 10 Amperes.

TEST EXAMPLE #4:

Switch the display between low current and high current patterns at varying speeds and observe the power consumption from the 5 Volt and 3.3 Volt rails with the use of an oscilloscope.

$$\text{Define: } \Delta V_{transient} = |V_{no\ min\ al} - V_{peak}|$$

The transient state should not push either power rail to leave operating intervals, and no ESP32 reset, display inaccuracy, or false alert activation should take place.

VERIFYING THE CENTRAL CONTROL SYSTEM

TEST EXAMPLE #1:

To test departure calculation, we will provide the ESP32 with a predetermined event and its corresponding event start time, travel times, and buffer values. We will compare the calculated departure timestamp with a calculated reference. The test passes if the difference is no greater than our allotted 60 seconds.

TEST EXAMPLE #2:

To ensure our 100 ms timing constraint, we can preload a departure time into the controller, record the scheduled timestamp at the same time that the LEAVENOW output is asserted, calculate: $e_t = T_{actual} - T_{departure}$, and check if the e_t is less than or equal to 100 ms each trial.

TEST EXAMPLE #3:

To test Wi-Fi interruption, we'll retrieve the next event, disconnect the Wi-Fi before the stored departure time occurs, and check to see that the LEAVENOW state still occurs at the original calculated time.

TEST EXAMPLE #4:

To justify our key-notification feature, we will put the system into the LEAVENOW state while the key sensor notices that the keys are still present. We will measure how long it takes for a phone-notification request to be placed, which needs to be within 1 second.

VERIFYING THE RGB DISPLAY SUBSYSTEM

TEST EXAMPLE #1:

To ensure that our display meets the capacity test of at least three users with their names and transportation options present, we will consider this test passed if we successfully program the board to show distinguishable information about three users without overlapping or corrupted information.

TEST EXAMPLE #2:

To test that the countdown mechanism properly updates, we will program a known departure time and display the countdown values to this specific event. If the information updates every 60 seconds, then the test passes.

TEST EXAMPLE #3:

To test the warning-states for accuracy, we will push the controller through the NORMAL, APPROACHING, LEAVENOW, and LATE states. While doing this, we will crosscheck the colors to ensure they move from original to yellow to red.

TEST EXAMPLE #4:

To ensure that the visual and audio cues are synced correctly, we will record the moments where the visual leave indication becomes visible and when the audible alert is heard. We want to ensure that the time is less than or equal to 100 ms in order to pass this test.

TEST EXAMPLE #5:

Finally, the display needs to present an accurate representation of our information. While programmed with the calendar information, countdown values, and updated warning states, we must watch and analyze the board to make sure there is no unintended loss of user information or board display glitches, such as flickering, frozen text, or unreadability.

VERIFYING THE HOOK SUBSYSTEM

TEST EXAMPLE #1:

We will thoroughly ensure that the hook detection system properly recognizes when a key is present or not by performing 100 controlled trials where we place and remove the keys on the hooks. We will test each hook separately and record the actual state of the keys to the KEY PRESENT, and KEY ABSENT states that the ESP32 reads. This subsystem passes if at least 90 out of 100 trials are correctly classified, or the accuracy measures 90 or above. The accuracy of this test is calculated through:

$$Accuracy = \frac{N_{correct}}{N_{total}} \times 100$$

TEST EXAMPLE #2:

To ensure that the system detects keys of different masses, we will test the hook with the minimum and maximum supported key masses that we discover during the prototyping process. The minimum key mass needs to be sufficient to produce the KEY PRESENT state. We will then test increasing masses up to the maximum mass condition, ensuring that the KEY PRESENT state is always triggered. We also want to make sure that the maximum mass or any other lesser masses do not exceed the maximum available mechanical movement of the mechanism. The test thus passes if the key masses are properly detected, and the spring, microswitch, 3D-printed housing, or hook do not become damaged.

TEST EXAMPLE #3:

To verify our spring-return feature, we plan to place keys on the hook, remove them, and allow the spring to return back to its equilibrium position on its own. The spring needs to move far enough upwards to trigger the KEY ABSENT state. We will be able to define any issues with friction, sticking, or mechanical misalignment with this test.

TEST EXAMPLE #4:

We will repeatedly place and remove keys from each hook while observing their corresponding LEDs. When the ESP32 registers the KEY PRESENT signal, then the LED needs to illuminate. When the key is absent, the LED must turn off. If each light for each hook behaves accordingly, then the test passes.

TEST EXAMPLE #5:

To confirm that our notification timing is within spec, we will program the system so that a user fulfills the reminder condition if their keys are still on the hook after their time is up. We will measure the time between when a reminder condition occurs and when the phone notification is requested to ensure that this all occurs within 1 second, verifying that our hook's sole purpose is properly completed. We will measure the notification delay as: $t_{delay} = t_{request} - t_{reminder}$.

VERIFYING THE ALERT SUBSYSTEM

TEST EXAMPLE #1:

To confirm that the audio playback of the subsystem properly works, a music clip will be stored locally, and the ESP32 will be triggered to play the recording. If the correct audio file plays through the speaker with relative clarity, then the test passes. We will also perform the same test with the Wi-Fi disconnected to test that this feature properly works without the use of Wi-Fi after the audio has already been uploaded.

TEST EXAMPLE #2:

To verify that our audio startup time is within our requirements, we will time when the Central Control Subsystem enters the LEAVENOW state and when the audio recording begins playing. Our delay calculations are as follows:

$t_{audio,delay} = t_{audio,start} - t_{LEAVENOW}$, with $t_{audio,delay} \leq 100ms$ to meet our requirements. In order to be precise enough for this experiment, oscilloscopes or logic analyzers are needed.

TEST EXAMPLE #3:

To verify that the RGB Display and audio alerts are activated by the same LEAVENOW state transition, we will measure the start of the visual alert to the start of audio playback. The synchronization error will be:

$$e_{sync} = |t_{visual} - t_{audio}|$$

We will perform this calculation amongst multiple LEAVENOW scenarios to make sure that our synchronization is accurate consistently rather than just for one trial.

TEST EXAMPLE #4:

Speaker volume will be tested to ensure that the alert is loud enough to be noticeable but not excessively loud. We will position a sound-level measurer 1 meter from the device in a quiet indoor room. The music alert will then be played, and the sound pressure must satisfy:

$$65dBA \leq SPL_{1m} \leq 85dBA$$

4. Costs

Figure 12: Parts Costs

Part	Manufacturer	Retail Cost (\$)	Bulk Purchase Cost (\$)	Actual Cost (\$)
64×64 RGB LED Matrix (×2)	Waveshare	79.98		79.98
ESP32-S3-WROOM-1-N8	Espressif Systems	5.66	5.66	5.66
AP2112K-3.3 Voltage Regulator	Diodes Incorporated	0.28	0.28	0.28
5 V, 10 A Power Supply	Adafruit	29.95		29.95
Microswitches (×6)	TBD	6.00*		6.00*
Indicator LEDs + Resistors (×6)	TBD	3.00*		3.00*
Springs + Hook Hardware (×6)	TBD	12.00*		12.00*
I ² S Audio Amplifier	TBD	6.00*		6.00*
Speaker	TBD	8.00*		8.00*
Custom PCB	Custom / PCB Manufacturer	10.00*		10.00*
3D-Printed Hook Assemblies	Custom	10.00*		10.00*
Enclosure / Wood	Custom	20.00*		20.00*
Connectors, Wiring & Passives	Various	10.00*		10.00*
Total		200.87*		200.87*

4.1 Labor

If three computer engineers were paid to create this project, how much would it cost? Using the equation provided in the assignment instructions: $(\$/\text{hour}) \times 2.5 \times \text{hours to complete} = \text{TOTAL}$ and the average salary of a UIUC new computer engineering graduate of \$119,064 or \$57.24 /hr, the cost becomes $C = (57.24)(2.5)(150) = 21,465$ per person or \$64,395 total.

5. Ethical considerations

One ethical limitation of our design is privacy. Since the device syncs with personal calendar information, it could reveal whether users are home. The ACM Code of Ethics emphasizes the importance of protecting user privacy and personal information, and our system will comply with these requirements by ensuring that it only accesses information necessary to display the next event and calculate departure time. Calendar information will not be stored longer than necessary, and users of the device must purposefully connect their calendars to the system.

In addition to ensuring that calendar information is protected, the system will only collect information that is absolutely necessary to perform operations and calculations. The key-sensing subsystem only determines whether a set of keys is present or absent and does not track any further information such as the user's physical location. Calendar access will be similarly discrete by using the minimal information to determine future events and departure times. These design decisions not only make the design process more concise but reduce the amount of personal information collected by the system, thus limiting the consequences that come with unauthorized access to user data.

The IEEE Code of Ethics upholds the standard of protecting public safety and honestly communicating with the limitations of a product or service. To align with these ideals, we will heavily consider safety and reliability throughout our construction and testing process to avoid incorrect calendar synchronization, sensor readings, or departure calculations that could provide users with faulty information and directions. Throughout development, we will constantly test the calendar, key sensors, alerts, and departure calculations before our demonstration. Additionally, to comply with safety regulations and ECE 445 laboratory requirements, we will avoid electrical risks such as short circuits, overheating, and incorrect voltages by utilizing regulated low-voltage power supplies and properly rated components.

There are a variety of engineering standards and guidelines relevant to our design. The IEEE and ACM ethical guidelines are relevant to the project because they provide instructions on how to regulate user information handling, maintaining reliability of the system, and communication of any system limitations. Because we are using the ESP32 for Wi-Fi communication, the design also makes use of the IEEE 802.11 wireless networking standard that the microcontroller will follow. Our project will strictly follow manufacture-specified operating voltages, currents, and temperature limits for any electrical components used. Our PCB design and assembly will be done using accepted practices for conductor spacing, trace

sizing, grounding, and installation. This is especially important to pay attention to because our RGB LED matrices draw significantly more current than its neighboring control components.

Our major electrical concerns in terms of safety include drawing too much current, shorting circuits, overheating, supplying improper voltages, and reversing the polarity of our connections. To bypass these hazards, we will use an externally regulated 5 Volt DC power source so that hazardous voltage is not present on the PCB. Our 5 Volt supply will be distributed to our high current devices, and a 3.3 Volt regulator will be present to feed current into the ESP32. Before any connections are made, all power rails will be double checked with a multimeter. Any current draw and component temperature will be monitored throughout testing. Wiring and PCB traces will also be sized correctly for their load, and all electrical connections that may become dangerous or exposed will be insulated during our final demonstration.

Our mechanical concerns include the wall-mounted design and the spring-loaded key-hook mechanisms. We need to ensure that the enclosure is secured to the wall in a way that supports the display, electronics within, and keys. The hook development will be assessed to prevent any sharp edges, loose hardware, or moving components from being exposed, which could cause injury. The completed enclosure will prevent users from encountering the PCB and internal wiring. Mechanical testing will be thoroughly done to verify that no components easily loosen or deform.

The intended societal and economic impact of our product is to assist households in organizing their schedules, remind them of important personal belongings, and reduce departure delays. We do recognize, however, that the key-sensing feature of our project could be misused to track when individuals are home, so no information from this feature will be stored permanently. Environmentally, we hope to limit the use of unnecessary electrical components and power consumption. Overall, throughout the entirety of our project, we aim to prioritize privacy, safety, reliability, and responsible use.

References

- [1] Espressif Systems, ESP32-S3 Series Datasheet, ver. 2.2, Mar. 2026. Available at: <https://www.espressif.com/en/support/documents/technical-documents>. Accessed October 2, 2026.
- [2] Espressif Systems, ESP32-S3 Hardware Design Guidelines, ver. 1.8, July 2025. Available at: <https://documentation.espressif.com/esp-hardware-design-guidelines/en/latest/esp32s3/index.html>. Accessed October 1, 2026.
- [3] Diodes Incorporated, AP2112 600mA CMOS LDO Regulator with Enable, datasheet, DS39724, rev. 2-2, June 2017. Available at: <https://www.diodes.com/datasheet/download/AP2112.pdf>. Accessed September 30, 2026.
- [4] Waveshare, RGB-Matrix-Px-64x64 Documentation, web page. Available at: <https://docs.waveshare.com/RGB-Matrix-Px-64x64>. Accessed October 2, 2026.
- [5] Google, Google Calendar API: Events, Google for Developers, web page. Available at: <https://developers.google.com/workspace/calendar/api/v3/reference/events>. Accessed September 29, 2026.
- [6] Google, Google Calendar API: Authorization and API Scopes, Google for Developers, web page. Available at: <https://developers.google.com/workspace/calendar/api/auth>. Accessed October 1, 2026.
- [7] Google, Routes API, Google Maps Platform Documentation, web page. Available at: <https://developers.google.com/maps/documentation/routes>. Accessed October 1, 2026.

- [8] Analog Devices, MAX98357A/MAX98357B: Tiny, Low-Cost, PCM Class D Amplifier with Class AB Performance, datasheet, rev. 16, July 2019. Available at: <https://www.analog.com/en/products/max98357a.html>. Accessed October 2, 2026.
- [9] M. Reid, ESP32-HUB75-MatrixPanel-DMA: HUB75 RGB LED Matrix Panel Library Utilizing ESP32 DMA, GitHub repository. Available at: <https://github.com/mrcodetastic/ESP32-HUB75-MatrixPanel-DMA>. Accessed September 29, 2026.
- [10] Association for Computing Machinery, ACM Code of Ethics and Professional Conduct, New York, NY: ACM, 2018. Available at: <https://www.acm.org/code-of-ethics>. Accessed September 20, 2026.