Smart Ball Milling Machine

Aaron Dahl, Chase Szafranski, Flavio Ortiz, Korey Menefee

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — A practical and applicable project, the Smart Ball Milling Machine is the overhaul and upgrade of an existing ball milling machine controller for university lab research equipment. This project employs a myriad of upgrades to the current controller including multiple sensory capabilities (current draw detection, ambient humidity and temperature detection, and motor temperature detection), a new and more refined user interface, remote operation through the Home Assistant App, and time scheduling and retention. These goals were first accomplished by reverse engineering the original ball milling machine controller to accurately distribute power and interface with the existing mill and motor. The Smart Ball Milling Machine uses the sensors to read and accumulate data to give alerts both on the hardware to make logic decisions such as stopping the motor if it overheats, but also uploading the statistical data to Home Assistant, so that the owner of the mill will have recordings of the data collected both for research and diagnostic purposes. This project was sponsored by and performed on behalf of the Blair Research group from the University of Central Florida.**

Index Terms -- Relays, Sensors, I2C, SPI, MCU, Faraday Cage, PCB

I. Introduction

Desired and requisitioned to make lab research more convenient and less tasking for the individual, the Smart Ball Milling Machine implements a series of desired improvements and quality of life enhancements for the controller of a SPEX Sample Prep ball milling machine. A ball milling machine is a device – often for chemical laboratories – that shakes and grinds substances and materials in a metal container with metal spheres by vigorously shaking them. The device currently/was only capable of performing this primary function for a selectable amount of time, and still had issues and drawbacks including manually stopping the machine in the middle of a milling cycle, forcing the user to 'babysit' the equipment for hours on end – at times up to half the day. Thus, the forefront and primary goal of this endeavor was to remedy this shortcoming.

To accomplish this foremost purpose two major functions needed to be implemented to the new 'smart' controller: First, the controller would need to be remotely operated so that the operator wouldn't be held hostage by the milling machine until the device was finished. Secondly, the new controller would need to be not only accurate, but flexible in how it would handle ever changing cycle times and intelligent to react to stimuli such as the temperature of the motor. To bring these about a real time clock was designed and programmed with algorithms to implement state save the progress for the mill while working in conjunction with the Home Assistant app via the microcontroller's (ESP32 WROOM UE) Wi-Fi capabilities to be monitored and operated remotely, measuring, showing, and reacting to the temperature of the motor with a simple thermistor epoxied to it.

With the new ball milling controller already being designed with the aforementioned purpose, it was convenient to consider additional improvements also. These improvements came in the form of a suite of additional sensors. A current sensor was added to measure power consumed to gauge the cost of research attributed to the ball milling machine; an ambient humidity and temperature sensor was added to give contextual background to any mishap in case of a temperature or humidity induced reaction to the chemicals/materials being milled; and an accelerometer was also added to the new mill controller record the shaking of the table/canisters to track deviation and degradation of the of this function.

II. System Overview

It is easiest to understand how the whole system fits together using a block diagram. Fig. 1 shows the major hardware components, along with their wired or wireless communication paths, and their source of power.

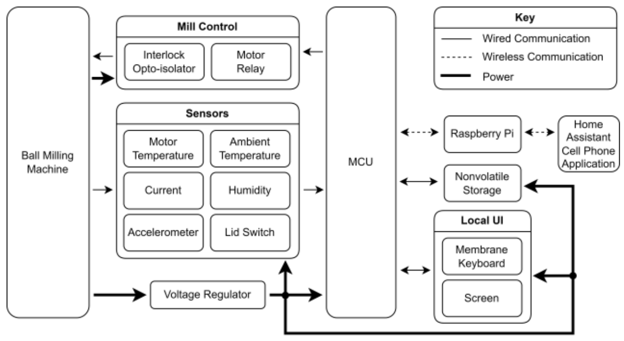

Fig. 1: Hardware Block Diagram

At a high level, the microcontroller uses inputs from the local UI, commands from a Home Assistant Raspberry Pi server, and readings from the sensors to determine how to set inputs to the opto-isolator that drives the interlock and the relay that drives the motor. Power for this system is supplied through an existing power supply board in the ball milling machine. This board provides us with a 12V rail and a 5V rail. We use a switching voltage regulator to convert the 12V source into our main 3.3V power source that drives our microcontroller, display, and other integrated circuits. We use the 5V source to switch our on-board relay.

III. System Components

A. Microcontroller

The Espressif ESP32-WROOM-32E microcontroller is the brain of the system. It drives the local user interface, communicates with a Home Assistant server, reads data from sensors, and interacts with the milling machine itself. We chose the ESP32 for these tasks because it has built-in WiFi capabilities — which are useful for communicating with Home Assistant, a large number of GPIO pins that allow us to support our peripherals, and a 240MHz dual core processor that ensures a responsive system. Additionally, the ESP32 has strong support for the Arduino IDE, which provides us with access to extensive library support for MQTT, graphics, sensors, storage devices, and other peripherals.

B. Relay

The relay is the single most important part of the project's functionality. At the most basic level, this project is about building hardware and software to switch a relay on a user-defined schedule. Our on-board relay is used to switch a larger relay in the ball milling machine, which is used to drive the motor.

C. Sensor Suite

Given that the mill's primary function is to mix various substances or grind materials into a fine powder, it's imperative to closely monitor the surrounding environment. Hygroscopic materials are highly sensitive to moisture, making it crucial to safeguard against high humidity levels, especially in regions like Florida where such conditions prevail. Additionally, elevated temperatures can alter the properties of certain materials, further emphasizing the need for comprehensive environmental monitoring.

Our design incorporates a sensor suite to monitor both environmental and operational conditions within the mill. To track ambient conditions, we will utilize a dual humidity/temperature sensor. This will enable us to monitor and mitigate any adverse effects caused by fluctuations in temperature and humidity. Furthermore, a separate thermistor will be employed to monitor the heat generated by the AC motor, thereby preventing prolonged overheating and potential damage.

To ensure optimal operational performance, we will deploy a current sensor to monitor the AC motor's power consumption. This will allow us to detect any irregularities in motor function and ensure that the relays are operating as intended. Additionally, we are addressing a common issue encountered by such mills which is the result of belt breakage or loosening. As the belt connects the AC motor to the shaking assembly this can result in wasted mill time or poor performance. By incorporating an accelerometer we enable real-time monitoring of the mill's activity, ensuring that it is functioning properly.

D. FRAM Module

A key feature of the smart milling machine controller is that it is able to recover from power disruptions. This functionality requires using nonvolatile memory to store program state While the ESP32 module we selected does have 4MB of onboard flash, flash memory has low write endurance. We are frequently writing to the memory, and this would cause problems in the future. The flash would almost certainly survive until the end of the semester, but we have built a system that is meant for longer term use in Dr. Blair's lab. If we relied on flash memory, it would eventually burn out. Instead, we selected an FRAM module with a rated write endurance of at least 1012 cycles, which is orders of magnitude more than required to outlast the milling machine's lifetime when we are saving state data once per second [3].

E. Voltage Regulator

Our design requires a power source capable of delivering 3.3V and approximately 2A of current. This configuration ensures sufficient power to operate the ESP32, OLED screen, sensor suite, and accommodates transient spikes in current demand during intensive tasks like WiFi transmissions. Using Webench, we obtained a voltage regulator design proficient in converting 12V to 3.3V while supplying a 2A current.

We chose a design with a low output ripple voltage of 5mV, as this ensures enhanced stability of the input voltage to our design. This low ripple voltage is crucial for maintaining consistent power delivery to our components, minimizing voltage fluctuations, and promoting overall system reliability. The design is modular, employing a separate PCB for the voltage regulator circuit which is connected to the main PCB via header pins. This modular setup facilitates easy replacement of the power supply in case of failure or the need for an upgraded version. Furthermore, it promotes efficient heat dissipation by allowing airflow over both the top and bottom of the PCB, thereby minimizing heat transfer to the main PCB.

F. Front Panel

The Front Panel Display serves a dual purpose: Naturally, its primary purpose is to implement the custom local UI and additional buttons needed to support the added functionality and features. Its second and less obvious purpose is to allow signal exit and entry. The ball milling machine is a motor and canister in an all-steel container making it a textbook example of a faraday cage. Replacing the existing front panel with a custom-made PCB display provides a port of entry and departure for Wi-Fi signal, thus enabling the Smart Ball Milling Machine's remote capabilities.

The front panel display is a simply designed and constructed PCB incorporating seven buttons (Up, Down, Start/Pause, Menu, Advance, Enter, and Back), a space for an OLED screen, holes to mount to the ball milling machine container, and an eight-pin header to attach the main PCB to while interfacing with it. The front panel itself was designed with the constraints of the existing mounting holes present on ball milling machine. The diameter of these holes and distance between them was carefully measured with a caliper and modeled in CAD before designing a PCB to match the same specifications.

G. Raspberry Pi

To carry out our projects goal to remotely interact with the mill, wireless communication from a device to the MCU is required. We chose to go with a Wi-Fi connection as opposed to Bluetooth because this allows use of IoT protocols and provides a more stable connection. To host this Wi-Fi connection, we chose to use a single on chip computer connected to a router because the computing requirements do not warrant the use of a more power computer such as a desktop. The Raspberry Pi 4 was chosen from here because it was the cheapest for the specs offered and featured easy setup of an operating system and good documentation.

IV. Hardware Detail

A. Real Time Clock

In addition to the hardware that has already been discussed, we are using a PCF8563 real time clock module to improve upon the accuracy of the ESP32's internal clock, and more importantly, to determine how long the system has been turned off upon a reboot. Measuring this time requires that the real time clock has a power source beyond what is supplied from the mill. The purpose of the down time measurement is to allow for different behavior upon reboot. If the power has been out for mere moments, we can simply resume the run. If it has been a longer amount of time (more than a few minutes), we would prefer to pause and allow the user to decide how to handle the situation. For this use case, we selected a supercapacitor as our backup power source, following the schematic in Fig. 2. We only need the power to last for a short amount of time, and we don't want to use a battery that will eventually need to be replaced.

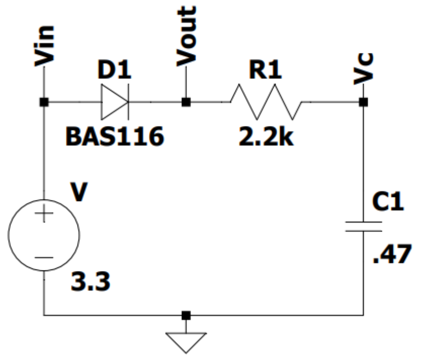

Fig. 2: Real Time Clock Backup Power

B. Preexisting Milling Machine Interface

Our new controller for the milling machine is able to use the existing interfaces that were designed for the original controller. This greatly simplifies the process of switching to our system. The milling machine controller has a sixteen pin connector and a six pin connector. Fortunately, many of these pins are either tied together or unused, which made it much easier to understand what all of the signals were.

The sixteen pin connector is the interface to the ball milling machine's power supply. It uses five pins for the 12V rail, seven pins for the ground signal, two pins for a 5V power source, one pin with the active low reset signal from the 5V linear regulator on the mill's power supply, and one pin that connects to an opto-isolator that controls the interlock. The fact that the opto-isolator is on the mill's power supply board, instead of on our board, means that our PCB does not need to handle enough current to drive the interlock, it only needs to send the control signal to enable it.

The six pin connector does not have any wires that are tied together, but two of the pins are unused. Two of the pins are either side of the mill's lid switch. We provide ground signal on one side of the switch and read the other side with a pull-up resistor. If the microcontroller reads a low signal, the lid is closed, so it is safe to engage the interlock and start the motor.

The other two pins are a 115 VAC line, and the input to the relay that drives the milling machine's motor. We use an on-board relay as a single pole single throw switch between the 115 VAC line and the mill's relay. We placed our on-board relay physically close to the six pin connector so that the high voltage bus has short traces. Because this source is only used to switch another relay the current going through our 115 VAC traces is tiny, only around 20mA [4].

C. Front Panel Display

We wanted an OLED screen due to their high contrast ratio which would allow the screen to be easily viewed from a distance and under various lighting conditions. We chose the NHD-2.7-12864WDY3-M display by New Haven Displays. This screen offers a resolution of 128x64 pixels, a 10,000:1 contrast ratio, and a 100,000-hour lifespan making it an ideal choice for design. While, It offers multiple connectivity options we chose to utilize its SPI interface due to it requiring the lowest number of pins to communicate with the microcontroller. The display will connect to our main PCB via a molex connector providing a secure and reliable connection.

D. Board Design

We developed our system across four printed circuit boards. Our control board contains our ESP32 and most of our peripherals. This board is connected to the front panel, which is a separate board to allow for simple physical connection to the milling machine, and because the control board is already quite crowded. This board has a cutout for our OLED screen, and seven membrane keyboard buttons. Our third board is simply for our voltage regulator. We placed this on a separate board to make it easier to troubleshoot if the regulator was problematic. Finally, we have a dedicated board for flashing and debugging the chip. This allows us to keep the micro-usb header off of our main board, which removes a physical failure point. This board has the micro-usb header, usb-uart bridge, and a linear voltage regulator that can optionally be used to power the system while flashing a new program.

V. Software Detail

A. Main Loop

Our main loop is set up in a sequential fashion, as shown in Fig. 3. On each iteration for each part of the program, we poll a timer or flag and if there is something new to do, that subsystem runs. This setup allows us to quickly handle inputs to any part of the program.

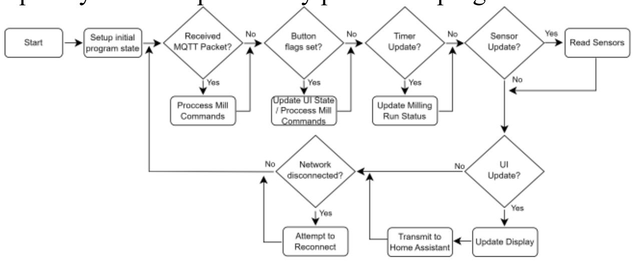

Fig. 3: Main Program Loop

B. MQTT

MQTT (Message Queuing Telemetry Transport) is a transport layer communication protocol similar to HTTP. It is built for IoT devices and so is lightweight and made for easy development of IoT networks through its features. These include the topic architecture, QoS (Quality of Service), and LWT (Last Will and Testament). Each of these concepts are used in our project and will be explained in their own section. The network is hosted by a broker and manages all clients on the network. A broker also manages topics. A broker is a router and for our project, is the Raspberry Pi 4. A client is an IoT device connected to the broker. In our project, the MCU controlling the mill is a client. A connection between a broker and client is done through TCP/IP so every connected client has an IP address. This means clients can be connected over the internet or over a LAN. These connections are bi-directional meaning packets can be sent from the client to the broker and vice versa. A packet contains a header and sometimes a payload. Packets consisting of only a header are for managing the overhead such as verifying connections. When a payload is attached to a header, this usually indicates the payload is a message and so a topic must be specified in the header.

i. MQTT Topics

A topic is a channel where messages can be sent from broker to client, or from client to broker. For a client to receive a message on a topic, it must be subscribed to that topic. The broker saves which clients are subscribed to which topics so it can route packets appropriately. To send a message on a topic, a client or the broker must publish a payload to a topic. Payload are sent as strings. Topics are also strings and have layers. Layer one is the highest scope and the very last layer is the lowest scope. Layers are separated by the '/' character. This optimizes development as connections become visibly organized and allows for the ability to subscribe to many topics at once by only subscribing to higher scopes and replacing the next lower scope with a '*' character. In our project, we have 3 topic layers. The highest layer is Mill_1 which represents the ball milling machine we are working on. Layer 2 separates layer 3 topics into 3 parts so that the broker and client can subscribe to all the subtopics in each part. Layer 3 is for handling received messages.

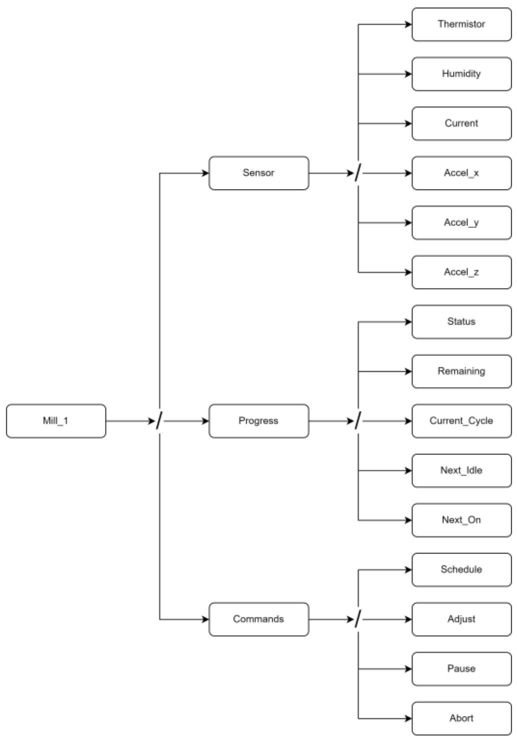

Fig. 4: MQTT Topic Flow Chart

As an example, temperature readings will be taken of the ball mill's motor and published on the topic 'Mill_1/Sensors/Thermistor'. These readings are sent from the MCU to the Raspberry Pi broker. Some topics will have information traveling from the broker (Raspberry Pi) to the MCU and some vice versa. There is one topic which is both listened to and subscribed to by both the MCU and the Broker and this is the topic 'Mill_1/Progress/Status'.

ii. MQTT QoS

Quality of Service describes how messages are delivered. There are 3 qualities of services: QoS 0, QoS 1, and QoS 2. QoS 0 is the default and there is no assurance that a message will be delivered to the client. This setting minimizes bandwidth for messages that are not important such as sensor readings that refresh frequently. QoS 1 guarantees a message will be received at least once by verifying with the client it has received the message. It does not guarantee it will not be received more than once. QoS 2 guarantees a message will be received exactly one time through more verification overhead. This is important to set if a duplicate message can break functionality. The following table shows every topic from Fig. 4 and their associated QoS as well as which direction messages are going. MCU→ Pi4

Topic	QoS	Direction
Mill_1/Sensors/Thermistor	0	Pi4←MCU
Mill_1/Sensors/Humidity	0	Pi4←MCU
Mill_1/Sensors/Current	0	Pi4←MCU
Mill_1/Sensors/Accel_x	0	Pi4←MCU
Mill_1/Sensors/Accel_y	0	Pi4←MCU
Mill_1/Sensors/Accel_z	0	Pi4←MCU
Mill_1/Progress/Status	1	Pi4→MCU Pi4←MCU
Mill_1/Progress/Remaining	1	Pi4→MCU
Mill_1/Progress/Current_Cycle	1	Pi4→MCU
Mill_1/Progress/Next_Idle	1	Pi4→MCU
Mill_1/Progress/Next_On	1	Pi4→MCU
Mill_1/Commands/Schedule	1	Pi4←MCU
Mill_1/Commands/Adjust	1	Pi4←MCU
Mill_1/Commands/Pause	1	Pi4←MCU
Mill_1/Commands/Abort	1	Pi4←MCU

Table 1: Topic QoS and Direction

iii. MQTT LWT

Last Will and Testament is a message with a corresponding topic that is set during an initial connection. This message is saved by the broker and is published to the saved topic when the broker loses connection to the client. Our LWT message is "Disconnected" and the LWT topic is "Mill_1/Progress/Status". This feature is useful in the final product because seeing a status of disconnected allows Dr. Blair to know that there is either a power outage or a Wi-Fi outage in real time and remotely as opposed to finding out later. This helps with planning visits to the lab and coordinating with his research assistants.

C. Remote UI

The remote UI is done using the Home Assistant (HA) OS running on the Raspberry Pi 4. HA works on YAML markup language and supports 'jinja'' for coding logic within YAML entries. Entities are core to HA and can be thought of as variables. As an example, an entity can be "lamp" and can have the property 'off' or 'on'. Entities are configured in the 'config.yaml' file. Dashboards are also core to HA and is a page a user will see and interact with and can contain anything that can be coded in YAML. HA intends for cards to be used to build dashboards and so is how our dashboard is built. Cards are containers for YAML and can come in various types but typically display text, entity properties, and user inputs. Automations is a feature of HA and is not necessary to building applications but is core to our project. An automation is a command that is triggered by an event. As an example, an automation could be triggered by an entities property changing, such as a button, and could result in an MQTT message being published on a topic. Additional features can be added to HA as integrations or add-ons and is explained in the following section.

i. HA Integrations and Add-ons

Integrations are programs running alongside HA but can be installed on our system within HA. Add-ons are libraries that can be downloaded and used within HA. The following shows all the integrations and add-ons used in our application.

Type	Name
Integration	MQTT
Integration	HACS
Add-on	Mosquitto Broker
Add-on	Studio Code Server
HACS Add-on	mini-graph-card
HACS Add-on	button-card
HACS Add-on	card-mod

Table 2: Integrations and Add-ons

For the integrations, HACS stands for Home Assistant Community Store and allows add-ons created by others to be easily installed. For the official add-ons, 'Mosquitto Broker' allows HA to be used as an MQTT broker and 'Studio Code Server' allows internal files to be edited, namely the 'config.yaml' file. The HACS add-ons are used to help create the dashboard.

ii. HA Components

Our remote UI built on HA includes a main dashboard and an optional dashboard with identical cards but with a layout to better suit a phone screen. The main dashboard has 2 pages. The first page is for scheduling and interacting with the mill, and for viewing activity of the mill. The second page is for viewing sensor data and power consumption data displayed on graphs. These pages will be added onto Dr. Blair's pre-existing dashboard. There are 38 total entities, all of which are configured in the config.yaml file. The total for our application includes 9 'mqtt' entities, 6 'input_number' entities, 6 'input_button' entities, and 17 'sensor' entities. There are 11 total automations. The 'automations.yaml' file contain all the code for our automations.

iii. HA Functionality

The first card on the first page in the main dashboard is titled "Run Scheduler". This card allows the user to set up a schedule for the mill. Three numbers can be set and are the 'On Time', 'Idle Time', and 'Cycles'. They can be typed and/or incremented using small buttons and have a range of 0 to 9999 and 0 to 99 for the 'Cycles'. Each digit represents 1 minute. The mill will run for the set on time, then will stop running for the set idle time and will repeat this for the set cycles but will not run the idle time during the last cycle. The equation is: total mill time = (On – Idle)*Cycles – Idle. Below this card is the 'Start Run' button. If the mill status is 'Not Running', pressing this will send the schedule to the MCU and start the run. If the status is 'Running', a pop-up will allow the user to abort and start the schedule or adjust the current schedule. If the status is 'Disconnected' or any other status, pressing the start run button will have no effect. The second card in this page is the 'Progress' card and displays the 'Status', 'Total Remaining Time', 'Current Cycle', 'Next Idle', and 'Next on'. All of these entities are updated by the controller over MQTT topics. A few cosmetic features are used to make the user experience better. These include changing icon colors based on various conditions, displaying '--' when a schedule is not running or when the mill is disconnected, and displaying 'Now' instead of '0' when the mill is idle or on. Below this card are two buttons: 'Pause' and 'Abort'. Pressing the pause button does as it implies, and pressing the abort button will cause a popup to appear to confirm the decision if the mill status is 'Running'. The second page on the dashboard displays graph views of live temperature readings, humdity readings, current draw readings, acceleration readings, and power consumption readings.

D. Local UI

The goal of the local UI is to enhance the mills basic functionality while introducing new advanced features. The original mill UI consists of a simple four-digit timer (two digits for minutes and two digits for seconds) allowing a maximum run time of 99 minutes 59 seconds and four buttons (Mins, Seconds, Start and Pause/Stop). We replaced these with OLED display directly mounted to a seven-button membrane keyboard. We programmed the graphics for our interface using the u8g2 open-source display library due to its simple setup and ease of use. Another benefit of this library is that it is supported by Lopaka graphics editor. This editor allowed us to easily create the graphics for our display via its interface, which allows you to draw simple shapes or insert strings of text on a simulated screen. The program then automatically generates the corresponding lines of code in C++.

The logic for the UI was coded in a combination of C and C++ programing languages and implemented through the use of case switch statements. The default or zero case corresponds to the home screen which allows the user to interact with the mill in a traditional manner such as setting the run time, starting, pausing and stopping a run. In addition to the basic features the user now has the ability to set the number of run cycles as well, eliminating the need to constantly monitor and restart the mill after cool down periods. There is also an indicator in the top left corner of the screen that allows the user to verify whether they are connected to the home assistant app or not.

Numbered cases correspond to the advanced menu options such Mill settings, Run Log, Network Status, and Sensor suite. Beginning with the mill settings the user can adjust the cool down time between runs, turn on and off the auto error handling feature that pauses a run upon detecting irregular conditions from the sensor suite. The run log displays all current settings, including total time and total time left taking into account runtime, idle time and number of cycles. The network status allows the user to view the network SSID the Esp32's IP address and the receive signal strength indicator. This is mainly to aid trouble shooting of network connectivity issues. The Sensor option allows users select and view all sensor readings in real time.

Button behavior was programmed in separate functions utilizing Booleans representing each button state. This was done so that we could have two operating modes, one for prototyping and one for the main design with minimal changes to the logic. For prototyping on a breadboard, we use interrupts to set the Booleans to true indicating a button press. However, our actual design requires the use of a port expander that will communicate via the I2C bus to register button presses.

E. Timing Subsystem

Accurate timing is an important part of this system. The mill needs to run for the scheduled amount of time in order to make sure Dr Blair's research results are easy to reproduce. The ESP32's internal clock is not rated for accuracy, so we decided to augment its precision using a real time clock. We are using a PCF8563, which requires an external oscillator We selected the Seiko SC-32S as the oscillator for our real time clock, with a rated accuracy of ±20ppm, which is more than sufficient for this purpose [5]. However, we do not want to read the real time clock every time we poll the time, because reading this chip uses I^2C, which takes longer than simply reading the ES32's internal timer.

In order to get the best of both worlds, we follow the logic in Fig. 5. We primarily rely on the ESP32's internal timer, but we schedule periodic synchronization with the real time clock module. This allows us to avoid most of the overhead associated with reading the real time clock, while still preventing the timer from drifting too far. In order to keep the program simple, we poll the ESP32's timer, rather than using hardware interrupts. We update our program's internal state every second, and perform the RTC_Synch routine on a user defined period. When the second counter reaches the period defined by the current milling machine settings, we reset the counter, update the state, and start a new timer for the next cycle.

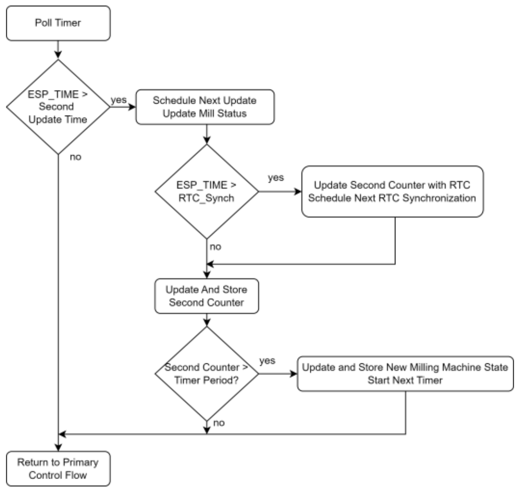
Poll Timer
ESP_TIME > Second Update Time
yes
no
Schedule Next Update Update Mill Status
ESP_TIME > RTC_Synch
yes
no
Update Second Counter with RTC Schedule Next RTC Synchronization
Update And Store Second Counter
Second Counter > Timer Period?
yes
no
Update and Store New Milling Machine State Start Next Timer
Return to Primary Control Flow

Fig. 5: Timer Subsystem

F. State Saving and Recovery

State saving is handled using a small ring buffer inside of our nonvolatile memory, as shown in Fig. 6. Every time the state is updated, the index is incremented and then the index % the ring buffer size is computed to handle wraparound. This gives us the next index. Older states are overwritten. When the system is rebooted, part of the initialization process is attempting to recover the previous state, in case there was a power disruption in the middle of a run. The most recent state is recovered by searching for the largest valid sequence number of the stored states. This is accomplished by first finding the smallest sequence number, then finding the largest state that occurs in sequence. Following this algorithm prevents the selection of a corrupted large sequence number.

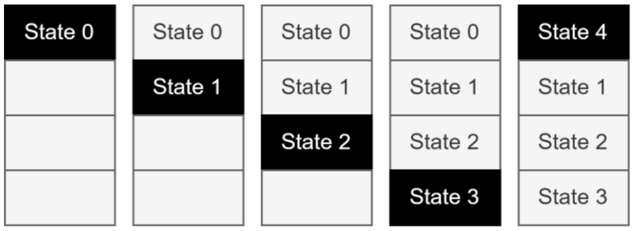

Fig. 6: State Saving Ring Buffer

Acknowledgment

We would like to thank our sponsor, Dr. Blair, for his generous support for this project in time, technical expertise, and funding. We additionally extend our thanks to Dr. Weeks for his constant presence in the senior design lab and his help with debugging our hardware, Mlinda Heng for lending a very steady hand in assembling our surface mount components on three different PCBs, Dr. Abichar for his embedded systems advice and his time as a reviewer, and Dr. Mike for agreeing to review our project despite his busy schedule. Finally, we would like to thank our coordinator, Dr. Chan, for his clear feedback on submissions throughout senior design.

References

[1] W. H. Cantrell, “Tuning analysis for the high-*Q* class-E power amplifier,” *IEEE Trans. Microwave Theory & Tech.,* vol. 48, no. 12, pp. 2397-2402, December 2000.
[2] W. H. Cantrell, and W. A. Davis, “Amplitude modulator utilizing a high-Q class-E DC-DC converter,” *2003 IEEE MTT-S Int. Microwave Symp. Dig.*, vol. 3, pp. 1721-1724, June 2003.
[3] Fujitsu Semiconductor, “Memory FRAM - MB85RC256V” Datasheet DS501-00017-3v0-E, 2013, https://www.fujitsu.com/us/Images/MB85RC256V-DS501-00017-3v0-E.pdf (accessed Oct. 27, 2023)
[4] Omron, “G7L Power Relay” Datasheet J055-E1-15, https://omronfs.omron.com/en_US/ecb/products/pdf/en-g7l.pdf (accessed Nov. 8, 2023)
[5] Seiko, “SMD Quartz Crystal Unit” Datasheet SC-32S, https://www.sii.co.jp/en/quartz/files/2013/03/SC-32S_Leaflet_e20151217.pdf (accessed Oct. 20, 2023)

Biography

Aaron Dahl is an Electrical Engineering student at UCF. After graduating, he plans to attend UF for a masters degree in computer engineering.

Chase Szafranski is an Electrical Engineering student at the University of Centrial Florida. After graduating, he plans to work for IMC Trading as a hardware engineer.

Korey Menefee is an Electrical Engineering student at the University of Centrial Florida. After graduating, he plans to work for Raytheon as a hardware engineer

Flavio R. Ortiz is a Electrical Engineering Student Veteran. Flavio’s career goals are to develop manned flight simulators specializing in dynamic integrated virtual environment for either the Navy or Air Force.