WIRELESS HEALTH MONITORING OF BUILDINGS

by

Danial Choo (1005177)
Devon Yap (1005215)
Clement Loh (1004988)

Project
Singapore University of Technology and Design
30.201 Wireless Communications and Internet of Things (IoT)
Senior Lecturer Dr. Kwan Wei Lek

19th April 2024

Content page

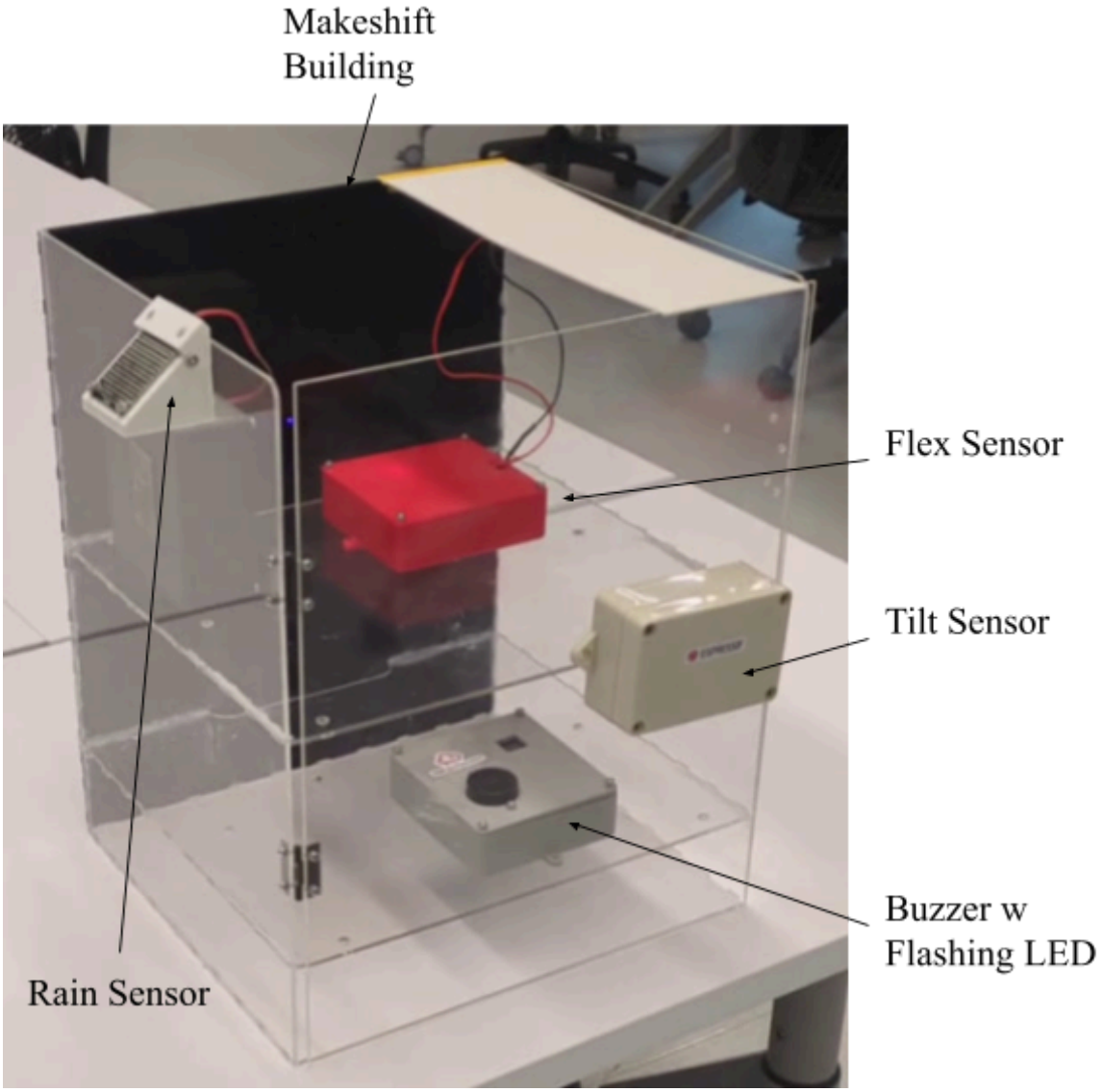
1. Project Description

This project aims to achieve wireless monitoring of the structural health of buildings. It does this by using multiple sensors, connected on a common network that senses different parameters which affects the health of buildings.

For our project, we are using four nodes; (1) Buzzer with Flashing LED that buzzes and flashes RGB LED once Flex Sensor hits a certain threshold, (2) Tilt Sensor that detects tilt in acceleration of gravitational force of the x,y,z axes of structures, (3) Rain Sensor that monitors the presence of water on the roof of a building and increases sampling rate for Flex Sensor upon hitting threshold set, (4) Flex Sensor that measures the flex of the structures of the building.

These sensors are housed independently such that it can be installed remotely, eliminating the need to connect power and data lines to the nodes.

Collectively, the data gathered from these sensors provides a holistic understanding of the buildings in real time. The remote updates over WiFi allows the building maintenance team to react instantaneously to anomalies in the structure, ensuring the building is at its best condition and promoting safety for the occupants in buildings.


Makeshift
Building
Flex Sensor
Tilt Sensor
Buzzer w
Flashing LED
Rain Sensor

2. Video Demonstration

- https://www.youtube.com/watch?v=zDYJlO5wrcY

3. Software required

3.1. For the code

- C
- Python3
- ESP-IDF with ESP RainMaker installed
- ESP RainMaker Mobile Application
- *Optional (Recommended)*: Visual Studio Code
- *Optional (Debugging)*: Arduino IDE
- *Optional (Debugging)*: Logic 2, Saleae

3.2. Repositories

- https://github.com/danialchoo79/rainmaker_accelerometer/tree/master
- https://github.com/ClementLohCK/esp-rainmaker/tree/master

4. Components required

4.1. Sensors

- 1 x ADXL 345 Accelerometer
- 1 x HW-028 Rain Sensor
- 1 x Buzzer
- 1 x Flex Sensor

4.2. Electrical

- 3 x ESP32-C6 Boards
- 1 x ESP32-S3 Board
- 4 Mini Bread Boards
- Assortment of Jumper Cables
- Assortment of Resistors (For LED, Voltage Divider Circuit)

4.3. Power Supply

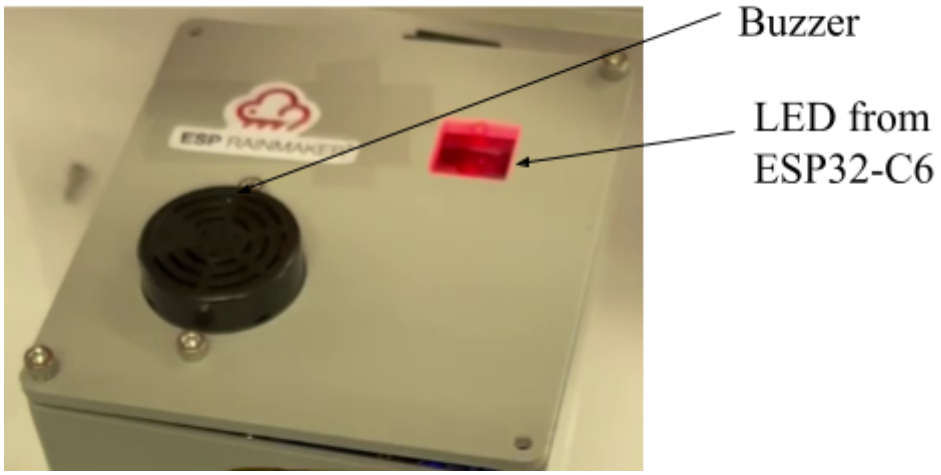
- 4 x 9V Battery
- 4 x Buck Converter (Step down Voltage from 9V to 5V)
- 4 x Snap Connectors

4.4. Hardware

- 4 x Casings (Preferably Waterproof)
- Assortment of Screws for Casing
- 1 x Type C Cable capable of Data Transfer
- 1 x Router Capable of WiFi
- 1 x Smartphone Capable of WiFi and BLE Communication
- 1 x Multimeter for Debugging Electrical Issues
- *Optional* : 1 x Logic Analyser for Debugging I2C Communications
- *Optional*: 1 x 3D Printer with PET-G Filament (Manufacture High Fidelity Casings)
- 1 x Computer capable of using ESP-IDF
- Soldering Kit (for soldering Buck Converters with Snap Connector and Jumpers)
- 3M Tape (Normal and Heavy Duty VHB)
- Cable Tie (mounting sensors on cylindrical surfaces)
- Self - Tapping Screws (mounting on materials such as thick block of wood)

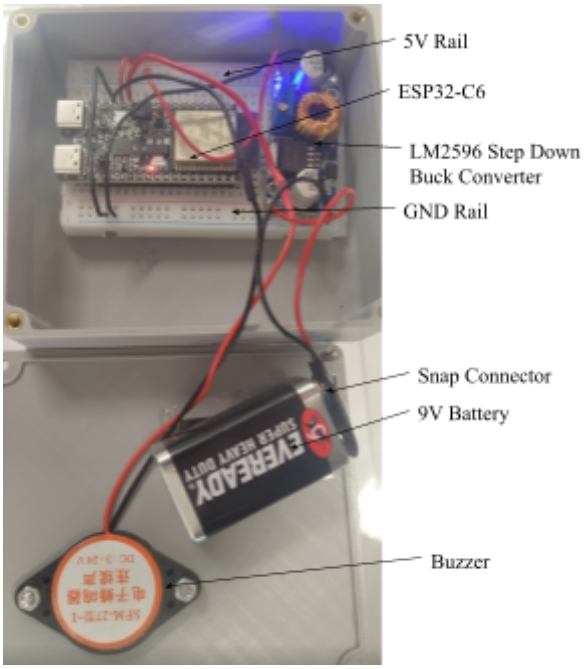
5. Node (1) - Buzzer with Flashing LED

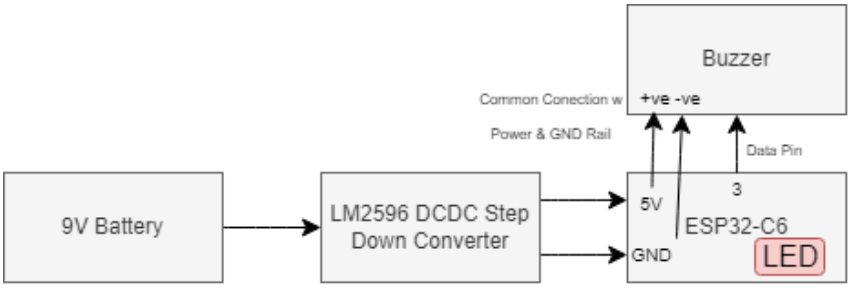
The buzzer accepts an input power of between 3V-24V, producing a monotonous sound with varying volumes according to the voltage supplied to pin 3. We also used the inbuilt RGB LED to flash red as a warning indicator. This buzzer can be turned off remotely using the ESP RainMaker phone application.

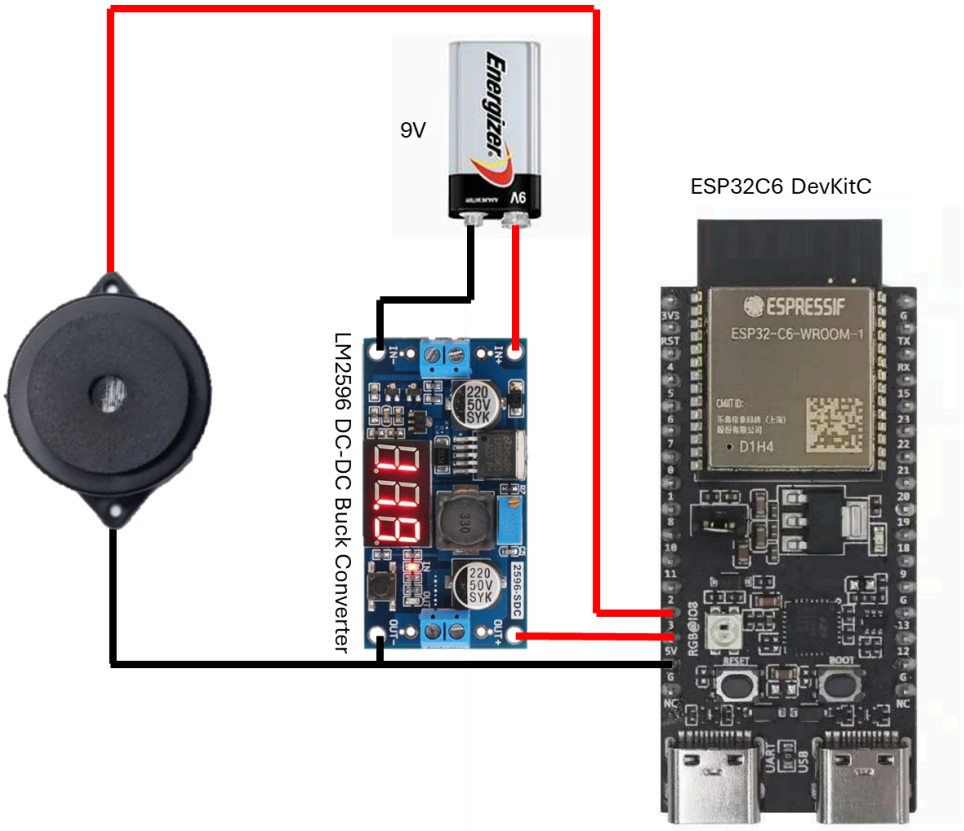


5.1. Hardware

The sensor uses a 3D printed PET-G plastic casing. In it, the ESP32C6 is connected to the Buzzer PWR and GND lines respectively. It is powered through a 9V Battery that will be stepped down to 5V into the 5V pin of the ESP32C6. Within the ESP32C6, there is an internal DC-DC Step-Down Converter that converts 5V to 3.3V. This 3.3V pin is used to power the Buzzer as its input voltage is from 3-24V. These connections are made through a breadboard that is taped to the casing.


5V Rail
ESP32-C6
LM2596 Step Down
Buck Converter
GND Rail
Snap Connector
9V Battery
Buzzer


Buzzer
Common Conection w
+ve -ve
Power & GND Rail
Data Pin
9V Battery
LM2596 DCDC Step
Down Converter
5V
3
ESP32-C6
GND
LED


9V
ESP32C6 DevKitC
LM2596 DC-DC Buck Converter
ESPRESSIF
ESP32-C6-WROOM-1

5.2. Firmware

Let's start from `app_main()`. All of the functions below are for initialisation.

```
/* Initialize Application specific hardware drivers and
 * set initial state.
 */
esp_rmaker_console_init();
app_driver_init();
app_driver_set_state(DEFAULT_POWER);

/* Initialize NVS. */
esp_err_t err = nvs_flash_init();
if (err == ESP_ERR_NVS_NO_FREE_PAGES || err == ESP_ERR_NVS_NEW_VERSION_FOUND) {
    ESP_ERROR_CHECK(nvs_flash_erase());
    err = nvs_flash_init();
}
ESP_ERROR_CHECK( err );

/* Initialize Wi-Fi. Note that, this should be called before esp_rmaker_node_init()
 */
app_wifi_init();

/* Register an event handler to catch RainMaker events */
ESP_ERROR_CHECK(esp_event_handler_register(RMAKER_EVENT, ESP_EVENT_ANY_ID, &event_handler, NULL));
ESP_ERROR_CHECK(esp_event_handler_register(RMAKER_COMMON_EVENT, ESP_EVENT_ANY_ID, &event_handler, NULL));
ESP_ERROR_CHECK(esp_event_handler_register(APP_WIFI_EVENT, ESP_EVENT_ANY_ID, &event_handler, NULL));
ESP_ERROR_CHECK(esp_event_handler_register(RMAKER_OTA_EVENT, ESP_EVENT_ANY_ID, &event_handler, NULL));

/* Initialize the ESP RainMaker Agent.
 * Note that this should be called after app_wifi_init() but before app_wifi_start()
 * */
esp_rmaker_config_t rainmaker_cfg = {
    .enable_time_sync = false,
};
```

Zooming into `app_driver_init()`, it allows the boot button on the ESP32C6 to toggle the buzzer and RGB LED using `push_btn_cb`, more on this to be covered later.

```
void app_driver_init()
{
    button_handle_t btn_handle = iot_button_create(BUTTON_GPIO, BUTTON_ACTIVE_LEVEL);
    if (btn_handle) {
        /* Register a callback for a button tap (short press) event */
        iot_button_set_evt_cb(btn_handle, BUTTON_CB_TAP, push_btn_cb, NULL);
        /* Register Wi-Fi reset and factory reset functionality on same button */
        app_reset_button_register(btn_handle, WIFI_RESET_BUTTON_TIMEOUT, FACTORY_RESET_BUTTON_TIMEOUT);
    }

    /* Configure power */
    gpio_config_t io_conf = {
        .mode = GPIO_MODE_OUTPUT,
        .pull_up_en = 1,
    };
    io_conf.pin_bit_mask = ((uint64_t)1 << OUTPUT_GPIO);
    /* Configure the GPIO */
    gpio_config(&io_conf);
    app_indicator_init();
}
```

```
static void push_btn_cb(void *arg)
{
    bool new_state = !g_power_state;
    app_driver_set_state(new_state);
```

Going back to `app_main()`, initialise a node named “ESP RainMaker Device” of type “Unknown-Alarm”. Back then, we did not know what effect this name causes hence we named it Unknown.

```
esp_rmaker_node_t *node = esp_rmaker_node_init(&rainmaker_cfg, "ESP RainMaker Device", "Unknown-Alarm");
if (!node) {
    ESP_LOGE(TAG, "Could not initialise node. Aborting!!!");
    vTaskDelay(5000/portTICK_PERIOD_MS);
    abort();
}
```

Create a device, making use of the standard “Switch” device offered by RainMaker, and assigning the callback function to this device. The callback function would be covered later.

```
/* Create a Switch device.
 * You can optionally use the helper API esp_rmaker_switch_device_create() to
 * avoid writing code for adding the name and power parameters.
 */
switch_device = esp_rmaker_device_create("Device-Alarm", ESP_RMAKER_DEVICE_SWITCH, NULL);

/* Add the write callback for the device. We aren't registering any read callback yet as
 * it is for future use.
 */
esp_rmaker_device_add_cb(switch_device, write_cb, NULL);
```

Create and add a parameter to set “Alarm” as the device name. Create a standard power parameter and add to the device. Assign the power parameter as the primary parameter so that it can be controlled from the dashboard.

```
 */
esp_rmaker_device_add_param(switch_device, esp_rmaker_name_param_create(ESP_RMAKER_DEF_NAME_PARAM, "Alarm"));

/* Add the standard power parameter (type: esp.param.power), which adds a boolean param
 * with a toggle switch ui-type.
 */
esp_rmaker_param_t *power_param = esp_rmaker_power_param_create(ESP_RMAKER_DEF_POWER_NAME, DEFAULT_POWER);
esp_rmaker_device_add_param(switch_device, power_param);

/* Assign the power parameter as the primary, so that it can be controlled from the
 * home screen of the phone apps.
 */
esp_rmaker_device_assign_primary_param(switch_device, power_param);

/* Add this switch device to the node */
esp_rmaker_node_add_device(node, switch_device);
```

The remaining code in `app_main()` is to enable features on the RainMaker app and initialise WiFi.

This is the event callback function such that this function runs when a device requests to write to its parameter from the cloud.

```
/* Callback to handle commands received from the RainMaker cloud */
static esp_err_t write_cb(const esp_rmaker_device_t *device, const esp_rmaker_param_t *param,
            const esp_rmaker_param_val_t val, void *priv_data, esp_rmaker_write_ctx_t *ctx)
{
    if (ctx) {
        ESP_LOGI(TAG, "Received write request via : %s", esp_rmaker_device_cb_src_to_str(ctx->src));
    }
    if (strcmp(esp_rmaker_param_get_name(param), ESP_RMAKER_DEF_POWER_NAME) == 0) {
        ESP_LOGI(TAG, "Received value = %s for %s - %s",
                val.val.b? "true" : "false", esp_rmaker_device_get_name(device),
                esp_rmaker_param_get_name(param));
        app_driver_set_state(val.val.b);
        esp_rmaker_param_update_and_report(param, val);
    }
    return ESP_OK;
}
```

The main action this function does is to toggle the power value for the buzzer using `app_driver_set_state` between `true` and `false`, then update and report the change.

```
if (strcmp(esp_rmaker_param_get_name(param), ESP_RMAKER_DEF_POWER_NAME) == 0) {
    ESP_LOGI(TAG, "Received value = %s for %s - %s",
            val.val.b? "true" : "false", esp_rmaker_device_get_name(device),
            esp_rmaker_param_get_name(param));
    app_driver_set_state(val.val.b);
    esp_rmaker_param_update_and_report(param, val);
}
return ESP_OK;
```

The state of the app_driver affects whether the buzzer is turned on or not. To do that, a software timer is created when `g_power_state` is `true`, and the software timer is deleted when `false`. The flag `flag_alarm_timer_created` is to ensure a maximum of 1 software timer is active at any point in time. This can be found inside app_driver.c.

```
int IRAM_ATTR app_driver_set_state(bool state) // Error: conflicting types for 'app_driver_set_state'; have 'int(int)
{
    g_power_state = state;
    if(g_power_state == true && flag_alarm_timer_created == false)
    {
        alarm_timer = xTimerCreate("app_alarm_update_tm", (REPORTING_PERIOD * 1000) / portTICK_PERIOD_MS,
                            pdTRUE, NULL, app_alarm_update);
        flag_alarm_timer_created = true;
        if (alarm_timer) {
            xTimerStart(alarm_timer, 0);
        }
    } else if (g_power_state == false && flag_alarm_timer_created == true)
    {
        int err2 = xTimerDelete(alarm_timer, 1); // xBlockTime set to 1 tick to prevent alarm_timer not deleted before
        set_power_state(false);
        if (err2 != 1) {
            ESP_LOGE(TAG, "Could not xTimerDelete, err: %i", err2);
            vTaskDelay(5000/portTICK_PERIOD_MS);
        }
        flag_alarm_timer_created = false;
    } else
    {
        set_power_state(false);
    }
    return ESP_OK;
}
```

The software timer created executes the function `app_alarm_update` every `REPORTING_PERIOD` set as 0.1s which frequently toggles the buzzer and RGB LED on and off using `set_power_state()` to create the warning sound. The buzzer is connected to `OUTPUT_GPIO`.

```
static void app_alarm_update(TimerHandle_t handle)
{
    if (alarm_state == true)
    {
        set_power_state(false);
        alarm_state = false;
    } else if (alarm_state == false)
    {
        set_power_state(true);
        alarm_state = true;
    } else {
        ESP_LOGI(TAG, "Weird alarm_state: %i, setting to false", alarm_state);
        set_power_state(false);
    }

}
```

```
static void set_power_state(bool target)
{
    gpio_set_level(OUTPUT_GPIO, target);
    app_indicator_set(target);
}
```

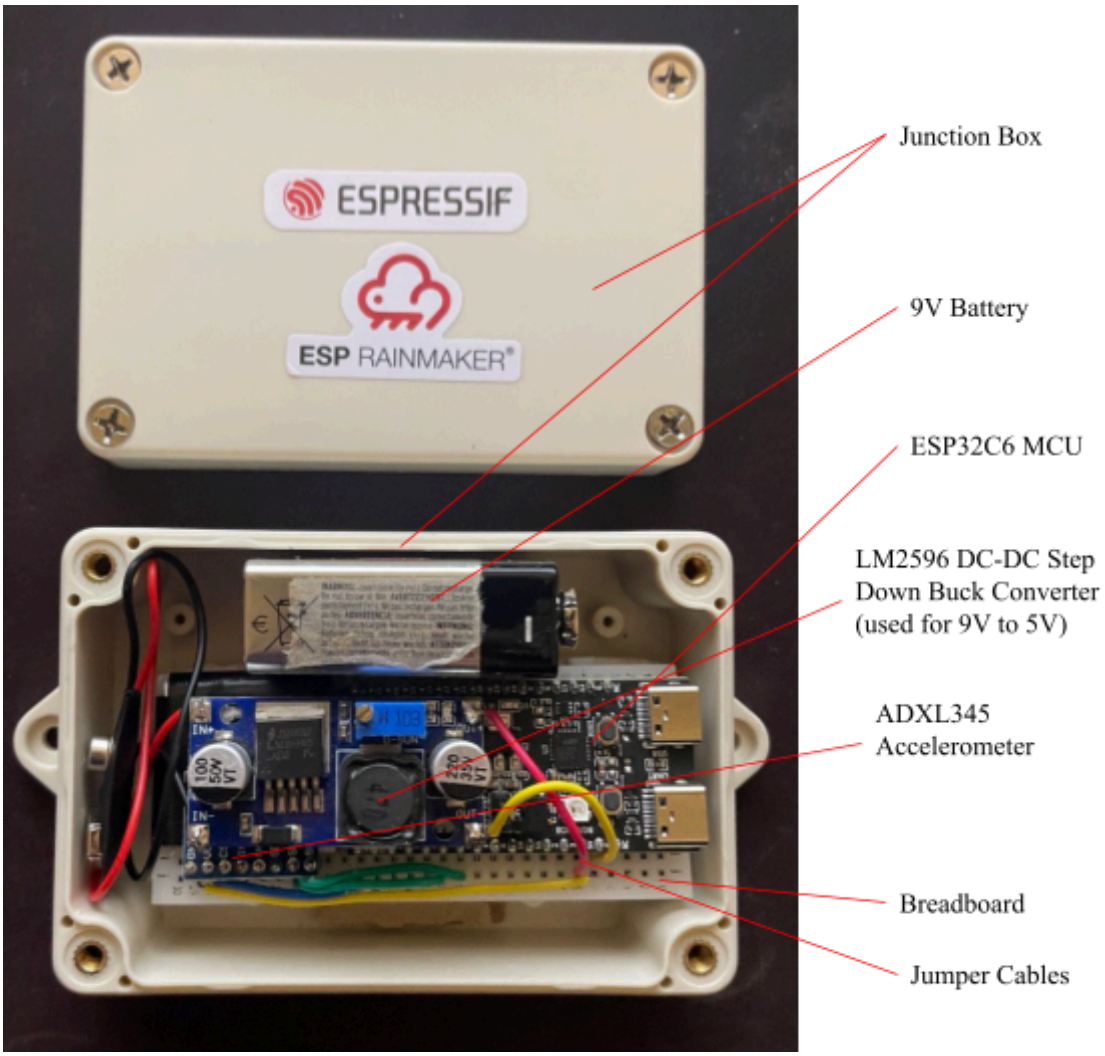
```
static void app_indicator_set(bool state)
{
    if (state) {
        ws2812_led_set_rgb(DEFAULT_RED, DEFAULT_GREEN, DEFAULT_BLUE);
    } else {
        ws2812_led_clear();
    }
}
```

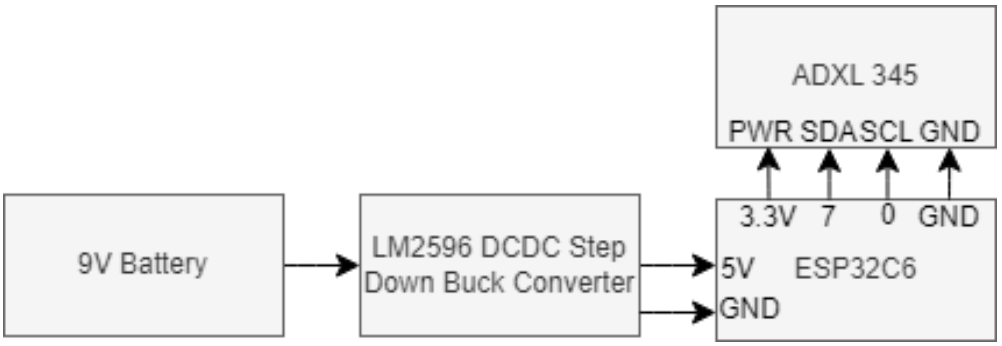
6. Node (2) - Tilt Sensor

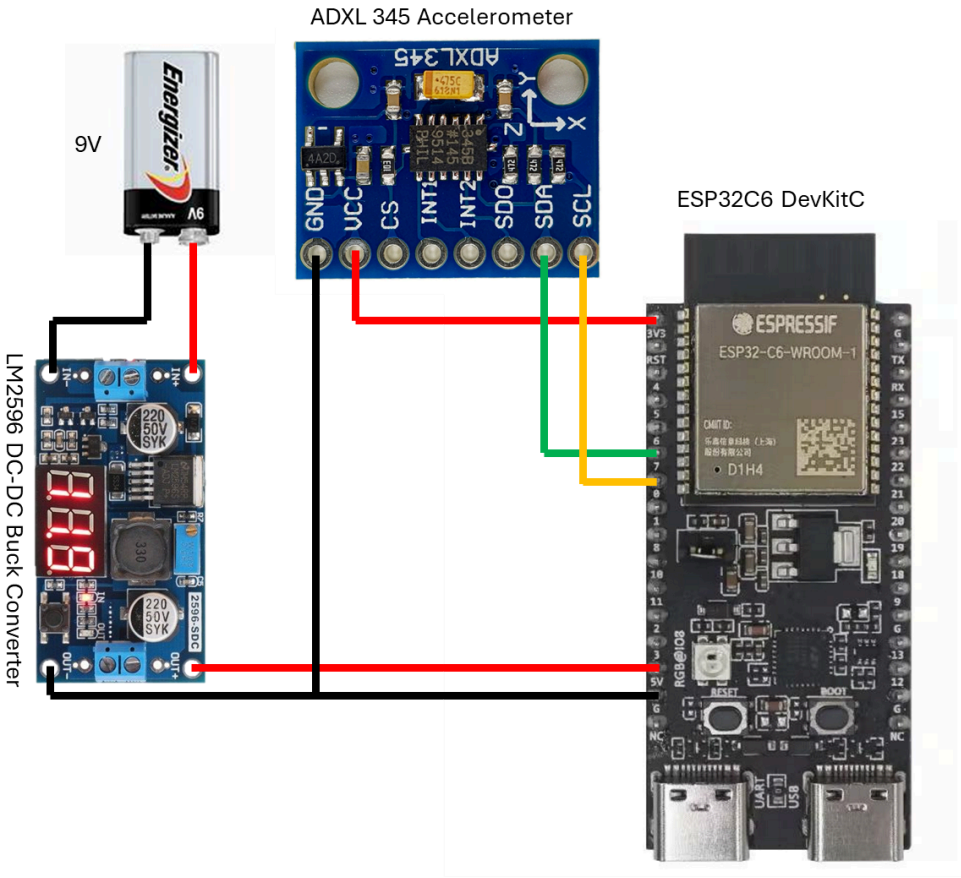
The mounted Tilt Sensor provides information regarding the tilt of a component. On the default setting, it measures up to +/- 2g gravitational force in the x,y,z axes. Additionally, it is able to detect acceleration within the ranges of +/- 1g, 2g, 4g and 8g. We have opted to maintain the +/- 2g sensitivity as it provides finer granularity. Our assumption is that structures do not move as much, and hence we aim to detect slight movements at higher granularity. Using the 2g setting instead of the 1g setting reduces error due to normal fluctuations.

6.1. Hardware

The sensor uses a ABS hard plastic casing. In it, the ESP32C6 is connected to the ADXL345 sensor using I2C Serial Clock (SCL), Serial Data (SDA), PWR (3.3V) and GND lines respectively. It is powered through a 9V Battery that will be stepped down to 5V into the 5V pin of the ESP32C6. Within the ESP32C6, there is an internal DC-DC Step-Down Converter that converts 5V to 3.3V. This 3.3V pin is used to power the ADXL 345 accelerometer as its input voltage is from 3-5V. These connections are made through a breadboard that is taped to the casing.


ESPRESSIF
ESP RAINMAKER
Junction Box
9V Battery
ESP32C6 MCU
LM2596 DC-DC Step Down Buck Converter (used for 9V to 5V)
ADXL345 Accelerometer
Breadboard
Jumper Cables


ADXL 345
PWR SDA SCL GND
3.3V 7 0 GND
9V Battery
LM2596 DCDC Step Down Buck Converter
5V
GND
ESP32C6



6.2. Firmware

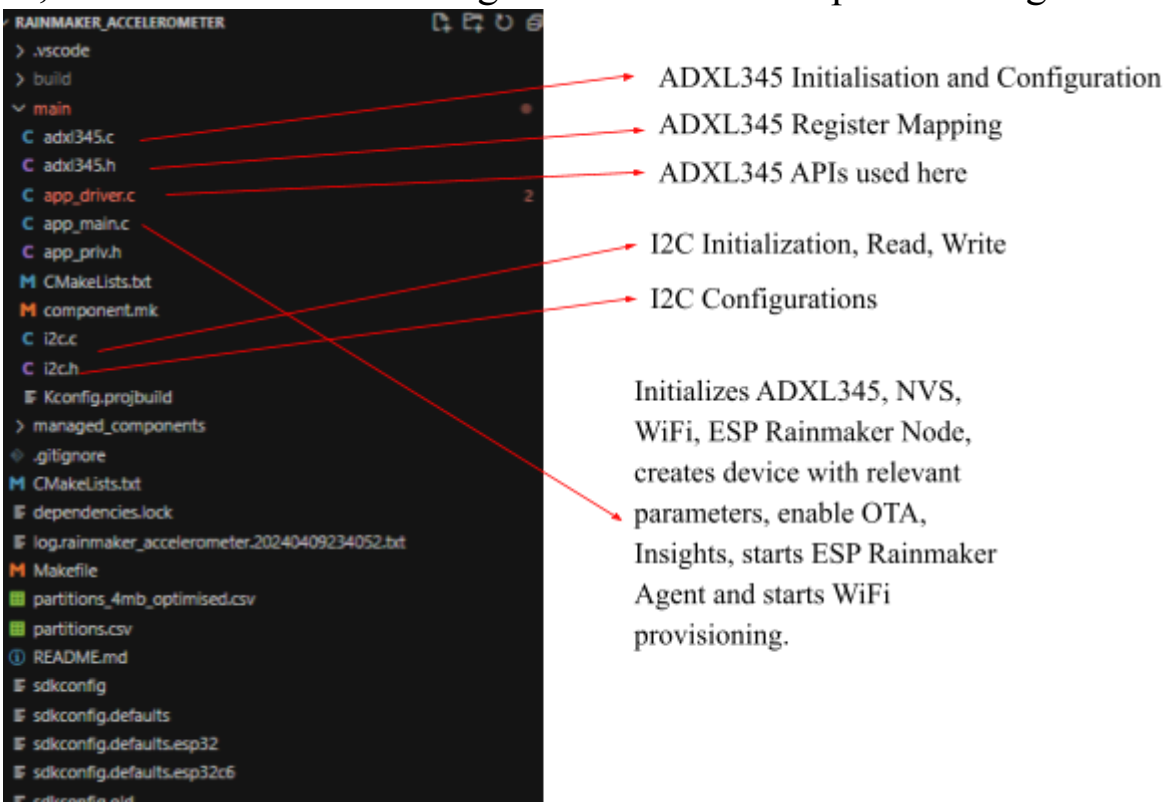
For the firmware, it has 3 main components. These components are modularized.

Firstly, for the ADXL345, it has adxl345.h header file that does the Register Mapping and adxl345.c file that intitialises and configures the ADXL345 and has 3 functions to read the x, y, z axes of the ADXL345 in +/- 2g.

Secondly, for I2C Communications, it has a i2c.h header file that configures the controller that controls the synchronisation (Clock) and frequency is set to Fast Mode at 400kHz and i2c.c file that has functions for initialising, writing and reading using I2C.

Thirdly, for the Integration, it has an app_driver.c file that uses the ADXL read APIs and applies a No-Turn Single Point Calibration Scheme where 10 data points are collected and average is computed.

This is the offset for the noisy values in ADXL345. It is then multiplied by a scaling factor for +/- 2g, 10 bit resolution of typically 3.9g mg/LSB. As it is in mg, it is converted to g. Then, there are ESP RainMaker APIs for reporting. Here, the sensor is also initialised by setting up the SDA and SCL pins, and I2C Controller. It also has an app_main.c that initialises ADXL345, NVS, WiFi, ESP RainMaker Node, creates a device with relevant parameters, enables OTA, Insights, starts ESP RainMaker Agent and starts WiFi provisioning.

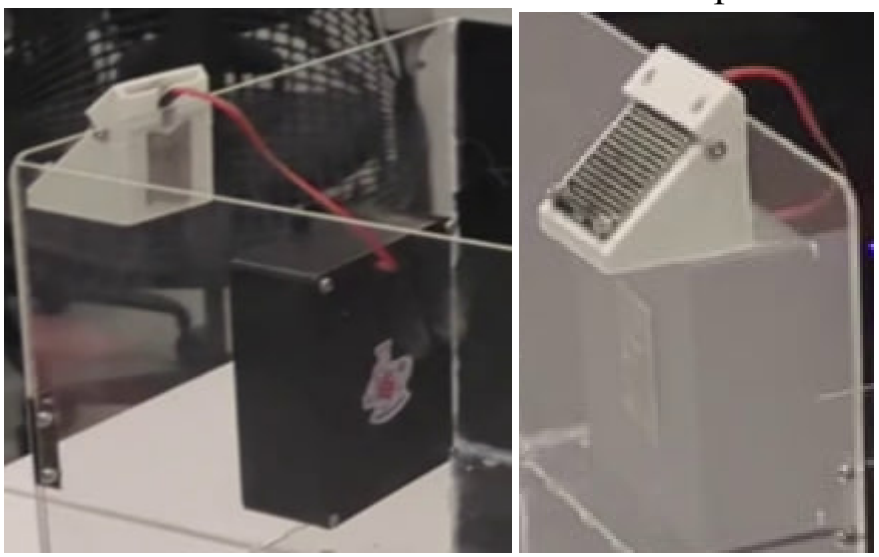


6.3. Data Logging

In the ESP RainMaker Dashboard, there are accelerometer devices for x, y, z axes. The data is sent wirelessly every `REPORTING_PERIOD` seconds based on the variable set in app_priv.h. Data logging for the three sensors can be seen in [ESP RainMaker Integration].

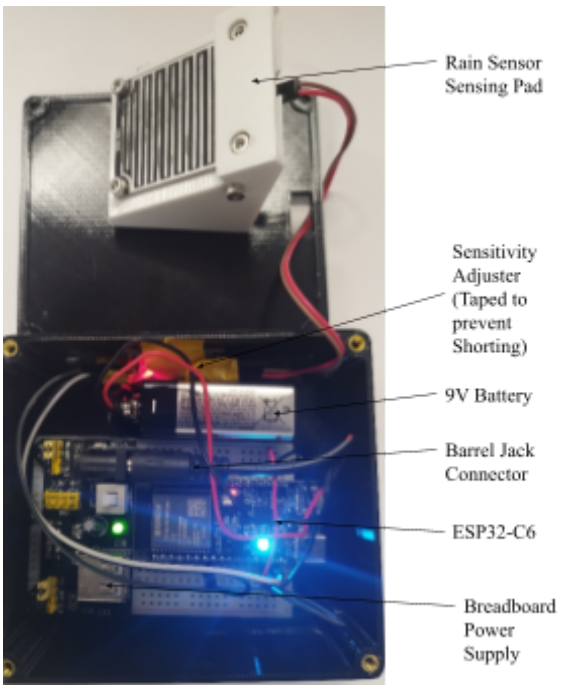
7. Node (3) - Rain Sensor

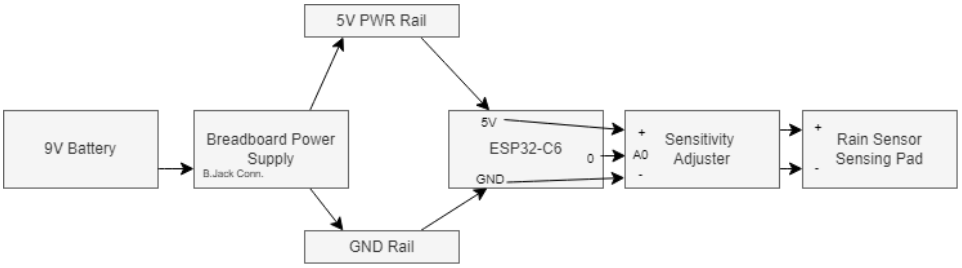
The HW-028 rain sensor provides a varying resistance depending on how much water or rain is found on the plate.

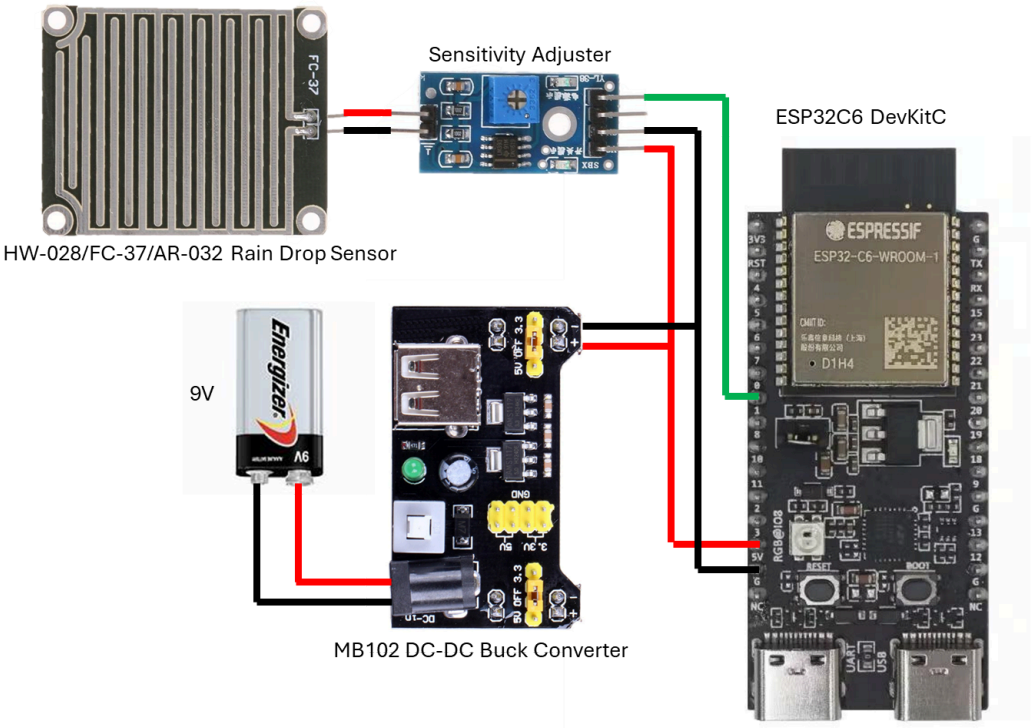


7.1. Hardware

The sensor uses a 3D printed PET-G plastic casing. In it, the ESP32C6 is connected to the Rain Sensor's Sensitivity Adjuster 5V Power and GND lines respectively. It is powered through a 9V Battery that will be stepped down to 5V into the 5V pin of the ESP32C6, which also powers the Rain Sensor as its input voltage is 5V DC. It uses GPIO Pin 0 which is Channel 0 of the ADC unit for Data Transfer from A0 of the Sensitivity Adjuster. These connections are made through a breadboard that is taped to the casing.


Rain Sensor
Sensing Pad
Sensitivity
Adjuster
(Taped to
prevent
Shorting)
9V Battery
Barrel Jack
Connector
ESP32-C6
Breadboard
Power
Supply


5V PWR Rail
9V Battery
Breadboard Power Supply
B.Jack Conn.
5V
ESP32-C6
GND
0
+
A0
-
Sensitivity Adjuster
+
-
Rain Sensor Sensing Pad
GND Rail


Sensitivity Adjuster
ESP32C6 DevKitC
HW-028/FC-37/AR-032 Rain Drop Sensor
9V
MB102 DC-DC Buck Converter

7.2. Firmware

Since most parts are explained in the first sensor, let's focus on the important parts. `app_sensor_update` is the function executed by the software timer `sensor_timer` every 5 seconds to read the sensor data, update the hue of the RGB LED and update and report the changed value to RainMaker.

```
static void app_sensor_update(TimerHandle_t handle)
{
    g_precipitation = DEFAULT_PRECIPITATION;
    g_precipitation = get_sensor_reading();
    ESP_LOGI(TAG, "Sensor reading: %f", g_precipitation);
    // for cycling at 0.5 intervals
    // static float delta = 0.5;
    // g_precipitation += delta;
    // if (g_precipitation > 99) {
    //     delta = -0.5;
    // } else if (g_precipitation < 1) {
    //     delta = 0.5;
    // }

    // Leaving this section to easily find out current state of precipitation
    g_hue = (100 - g_precipitation) * 2;
    ESP_LOGI(TAG, "g_hue: %d", g_hue);
    ws2812_led_set_hsv(g_hue, g_saturation, g_value);
    esp_rmaker_param_update_and_report(
            esp_rmaker_device_get_param_by_type(rain_sensor_device, ESP_RMAKER_PARAM_TEMPERATURE),
            esp_rmaker_float(g_precipitation));
}
```

`get_sensor_reading()` does most of the work, by reading the analog data from the pin in one shot. The `raw_value` is mapped to 0%-100% based on the 12-bit analog data received from the ADC. Since a fully wet sensor gives a lower reading due to the increased resistance, the sensor value is inverted to show maximum precipitation percent when the sensor is fully wet.

```
int map_range(int x, int in_min, int in_max, int out_min, int out_max)
{
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

int get_sensor_reading(void)
{
    // Read values
    int raw_value = -1;
    adc_oneshot_read(adc_handle, ADC_PIN, &raw_value);
    int sensor_val = map_range(raw_value, ADC_RAIN_RAW_MIN, ADC_RAIN_RAW_MAX, 0, SENSOR_RANGE); // can't invert out_min and out_max by s

    int inverted_sensor_val = 100 - sensor_val;

    // Comment out the line below to silence it
    ESP_LOGI(TAG, "Raw ADC value: %d, mapped sensor value: %d, inverted sensor value: %d", raw_value, sensor_val, inverted_sensor_val);

    return inverted_sensor_val;
}
```

A minor UI touch-up was using the standard Lightbulb logo for this Rain Sensor as it symbolises whether the weather is dark when raining.

```
rain_sensor_device = esp_rmaker_temp_sensor_device_create("Rain Sensor", ESP_RMAKER_DEVICE_LIGHTBULB, app_get_current_precipitation());
```

The parameter name could not be different from “Temperature” as we used the standard `esp_rmaker_temp_sensor_device_create()` which automatically sets the name to be “Temperature”. Adding an additional name parameter afterwards does not overwrite the name.

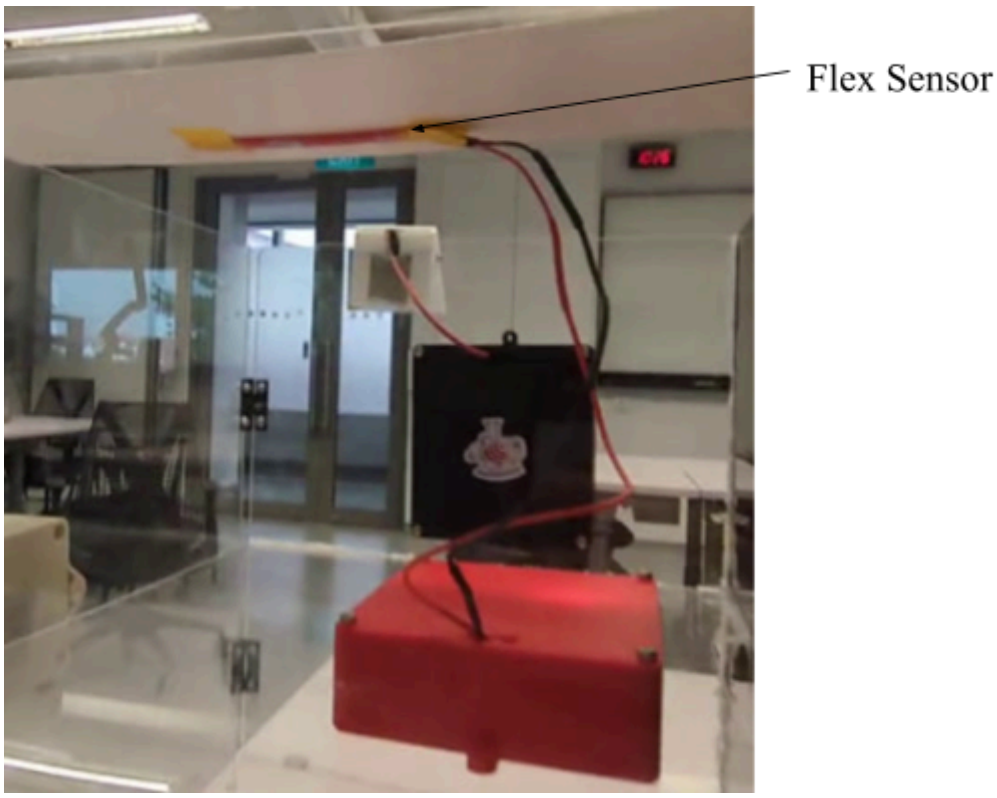
```
rain_sensor_device = esp_rmaker_temp_sensor_device_create("Rain Sensor", ESP_RMAKER_DEVICE_LIGHTBULB, app_get_current_precipitation()); // Changing 2nd arg into ESP_RMAKER_DEVICE_LIGHTBULB doesn't work for temp_sensor_device_create;

// esp_rmaker_device_add_param(rain_sensor_device, esp_rmaker_name_param_create(ESP_RMAKER_DEF_NAME_PARAM, "Rain Sensor")); // tried to add name of parameter but overwritten by temp_sensor_device_create
```

7.3. Data Logging

In the ESP RainMaker Dashboard, there is only one device for the precipitation data. The data is sent wirelessly every `REPORTING_PERIOD` seconds based on the variable set in app_priv.h. Data logging can be seen in ESP RainMaker Integration.

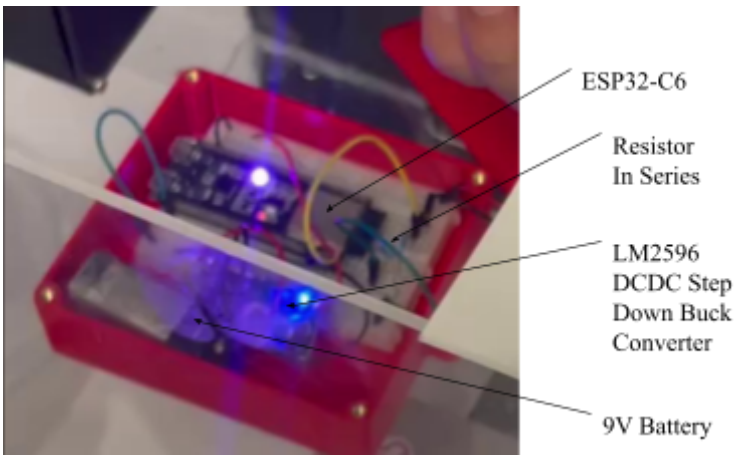
8. Node (4) - Flex Sensor

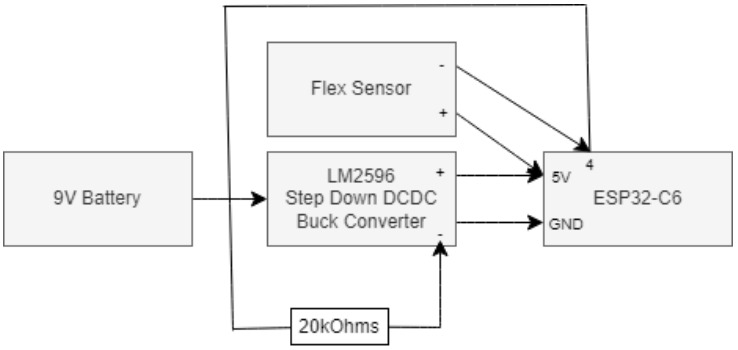
The Flex sensor provides a varying resistance depending on the amount of compression or tension on the surface.

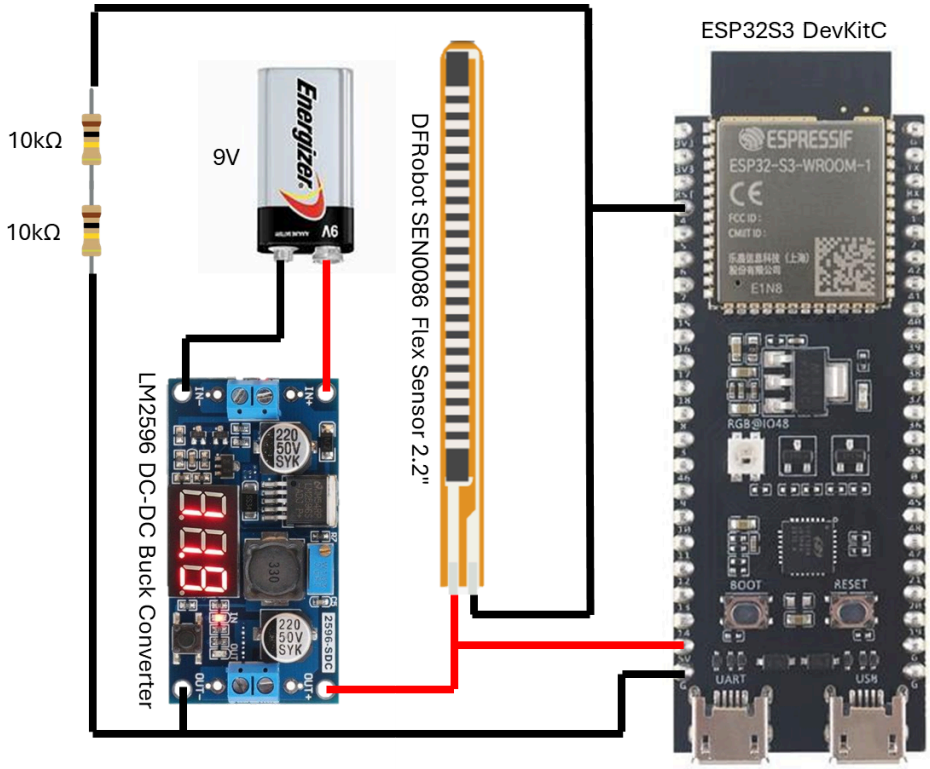


8.1. Hardware

The sensor uses a 3D printed PET-G plastic casing. In it, the ESP32C6 is connected to the 5V Power and GND lines via the rails respectively. It is powered through a 9V Battery that will be stepped down to 5V into the 5V pin of the ESP32C6. The Flex Sensor is powered by the 5V pin. It uses a GPIO Pin 0 which is Channel 0 of the ADC unit to read the analog data of the voltage divider circuit. These connections are made through a breadboard that is taped to the casing.







8.2. Firmware

Since most parts are explained in the first sensor, let's focus on the important parts. The callback function for the device changes the

sampling frequency according to the updated “Frequency” parameter value, ranging from 0 to 5. 0 for the default `REPORTING_PERIOD` of 32 seconds, 1 for 16 seconds, 2 for 8 seconds and so on.

```
/* Callback to handle commands received from the RainMaker cloud */
static esp_err_t write_cb(const esp_rmaker_device_t *device, const esp_rmaker_param_t *param,
            const esp_rmaker_param_val_t val, void *priv_data, esp_rmaker_write_ctx_t *ctx)
{
    if (ctx) {
        ESP_LOGI(TAG, "Parameter %s kena called back cause received write request via : %s", esp_rmaker_param_get_name(param), esp_rmaker_device_cb_src_to_str(ctx->src));
    }
    char *device_name = esp_rmaker_device_get_name(device);
    char *param_name = esp_rmaker_param_get_name(param);
    if (strcmp(param_name, "Frequency") == 0) {
        ESP_LOGI(TAG, "Received value = %d for %s - %s",
                val.val.i, device_name, param_name);
        app_driver_set_level(val.val.i);
    } else {
        ESP_LOGI(TAG, "Nothing changed");
    }
    esp_rmaker_param_update_and_report(param, val);
    return ESP_OK;
}
```

That is done using `app_driver_set_level()` by deleting the software timer `sensor_timer` with the previous reporting period, then creating another software timer also named `sensor_timer` with a different reporting period.

The reading of analog data is similar to Rain Sensor as explained earlier.

For the RainMaker UI, we have to create a parameter to indicate the current sampling rate. The default speed parameter is used but named as Frequency.

```
/* Create own parameter */
esp_rmaker_param_t *frequency_param = esp_rmaker_speed_param_create("Frequency", DEFAULT_SPEED);
esp_rmaker_param_add_ui_type(frequency_param, ESP_RMAKER_UI_SLIDER);
esp_rmaker_device_add_param(temp_sensor_device, frequency_param);
// esp_rmaker_param_t *switch_param = esp_rmaker_param_create("Alert", NULL, esp_rmaker_bool(fal
// esp_rmaker_param_add_ui_type(switch_param, ESP_RMAKER_UI_TRIGGER); // Trigger needs to be abl
// esp_rmaker_device_add_param(temp_sensor_device, switch_param);
```

8.3. Data Logging

In the ESP RainMaker Dashboard, there is only one device for the curvature data. The data is sent wirelessly every `REPORTING_PERIOD` seconds based on the variable set in app_priv.h by default. Otherwise, it would update its reporting period

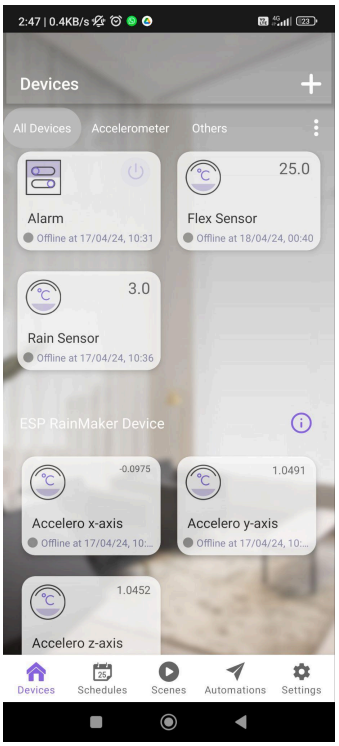
based on the precipitation data sent by Rain Sensor. Data logging can be seen in ESP RainMaker Integration.

9. ESP RainMaker Integration

This section covers how we use the features of the RainMaker phone application.

9.1. Dashboard

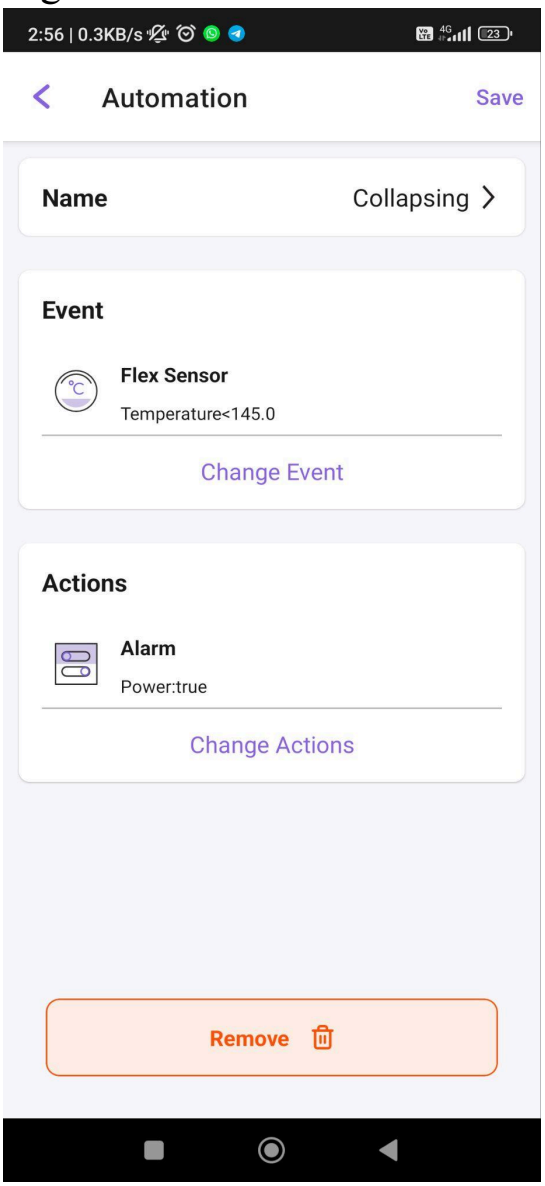
After provisioning all of the nodes using the same mobile phone, all of them would appear in the same dashboard. An operator just needs one phone to check on the structural integrity at places where the nodes are placed. The operator has the capability to turn off the Buzzer with Flashing LED alarm, this ensures that the operator would be informed about a potential failure in the building integrity, and conduct safety checks before the alarm is dismissed.



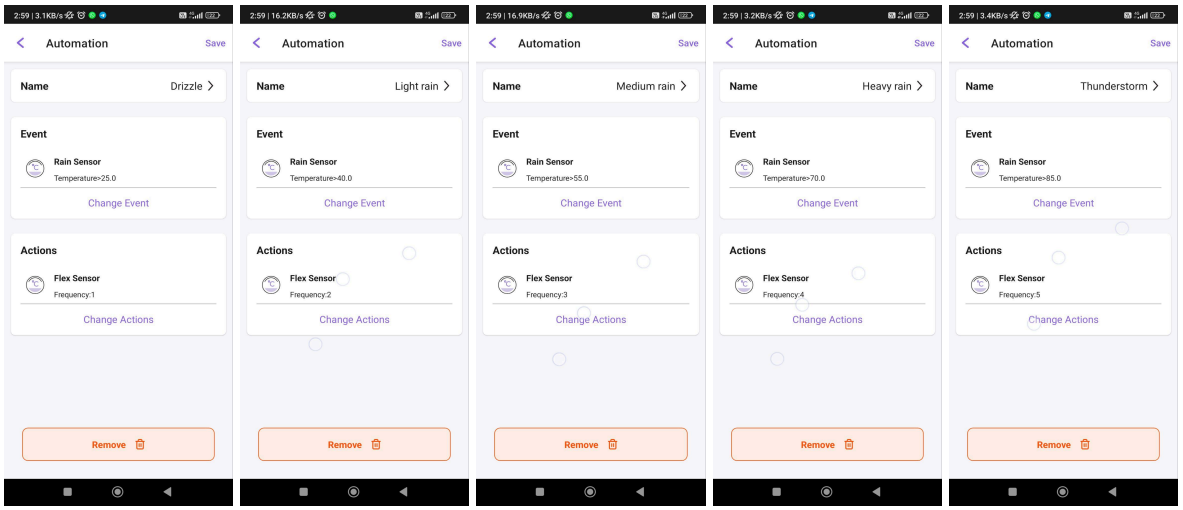
9.2. Automation

Different nodes are connected to each other using the automation feature in the app.

When the node with the Flex Sensor senses a change in the analog reading above the arbitrary threshold of 145.0, it would switch the parameter “Power” of Buzzer with Flashing LED to `true`. That would change `g_power_state` to `true` by a write request over the cloud and trigger the alarm, or deactivate it if `false`. An example that would cause a change is if the floor of a building starts to sink and create a ‘U’ shape. The arbitrary threshold set should be able to detect that small change.



When the node with the Rain Sensor detects a change in the precipitation percent, it changes the “Frequency” parameter of the node with the Flex Sensor so as to change its sampling rate depending on the 5 levels of wetness. This is to get more frequent data points during rainy days as structures are more prone to failure during such periods.



10. Breakdown of Roles

Tasks / Members	Danial	Devon	Clement
Hardware (Mechanical)	Used 1 High Fidelity Casing (ABS) [Junction Box]. Assembled final prototype.	Designed and 3D Printed 3 High Fidelity Casing (PET-G). Assembled final prototype.	Assembled final prototype.
Hardware (Electrical)	Designed Battery Power System for 4 Sensors.	Circuit Connection for Flex Sensor.	Circuit Connection for Rain Sensor and Buzzer with Flashing LED.

	Soldered all 3 Buck Converters with Snap Connectors and Jumpers. Circuit Connection for Tilt Sensor with I2C.		
Firmware (ESP32C6 and Rmaker)	Firmware for Tilt Sensor and ESP RainMaker Created a README format for consistency.	Ideated about logic for automation between nodes.	Ideated about logic for automation between nodes. Firmware for Rain Sensor and ESP RainMaker. Firmware for Buzzer with Flashing LED and ESP RainMaker. Firmware for Flex Sensor and ESP RainMaker. Automation among nodes.
Report	Created structure of report.	Beautified Wiring Diagrams (Physical	Did Firmware and Automation for Node (1), (3) and (4).

	Did the Components Required. Did Project Description, Tilt Sensor, Breakdown of Roles, Conclusion. Did Wiring Diagrams (drawio - high level understanding). .	Connections).	Did ESP RainMaker Integration.
Video	Did Video.	-	Gave feedback for Video.
Product Demo	-	Assisted in demonstration.	Presented prototype.

11. Conclusion

To sum it up, these sensors provide holistic understanding of the state of the buildings. These devices are designed with flexibility, modularity and scalability as considerations as it uses an independent power source and many devices can be connected to a single application (ESP RainMaker). Additionally, the Flex, Rain and Buzzer with LED nodes have interactions amongst one another, which provides the additional functionality of an intelligent sensor system. This achieves our objective to develop a remote health monitoring system of buildings to promote safer occupancy for dwellers as well as automating the monitoring process for building integrity technicians and engineers.