

# Design Review 3 - SIT

Group 11 - IR Remote Control

# Meeting Agenda

1. Review of original problem statement and requirements
2. Review of system design and subsystem organization
3. Review of progress to date as it relates to original goals:
  - a. Board design and details
    - i. IR Actuator subsystem
    - ii. Power subsystem
  - b. Lower level software
    - i. Non-volatile memory and user data organization
    - ii. Sending and receiving various signals using IRremote.h package
  - c. Website design and details
    - i. HTML design
    - ii. Final interface

# Original Problem Statement:

The original problem statement was a simple observation based on struggles most had with typical remotes:

1. Missing remote
2. Dead batteries
3. Insufficient signal
4. Multiple users cannot be in control at the same time
5. No customization / hotkeys
6. Can only control one type of device with each remote (no universality)

# Original Requirements

This was copied straight from the proposal submitted last semester. We will be candid where progress is and what is realistic moving forward, but there was no fibbing in this report.

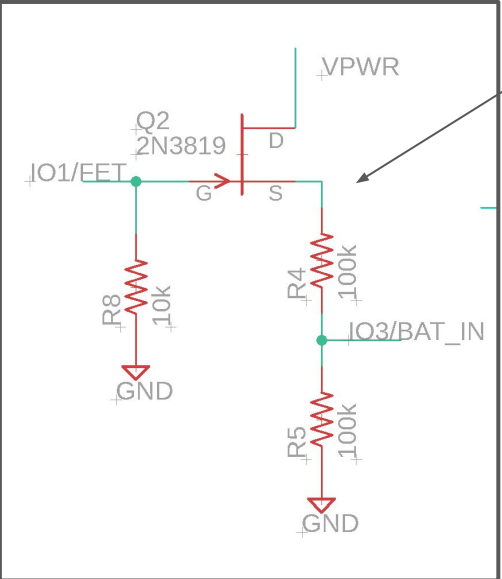
1. Functional communication between ESP32 and various TVs
  - a. Addresses hardware actuator and universality with multiple TVs
2. Functional communication between user phone / laptop and ESP32
  - a. Address lower level coding actuation and functionality of website from bottom-up
3. Multiple users at once
  - a. Intrinsic with this type of site
4. Attractive and User Friendly Interface
  - a. Website not only needs to look good, but it needs those special “hot-keys”

# Review of What We've Done: Hardware

- Started with basic receiver and IRED circuits connected to kit board.
- Slowly learned how to implement correct packages to use the hardware with ESP32 and IRremote.h
- Enhanced the circuits and drivers to increase effective range
- Tested subsystem for power measurements and written software to monitor
- Implemented everything together into final PCB design

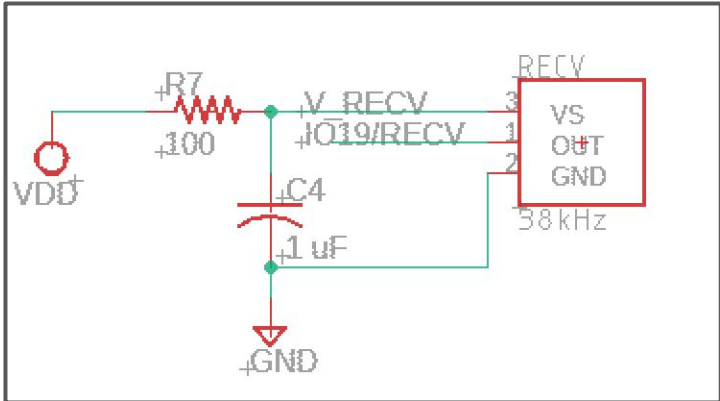
# Review of What We've Done: Hardware

Battery Read



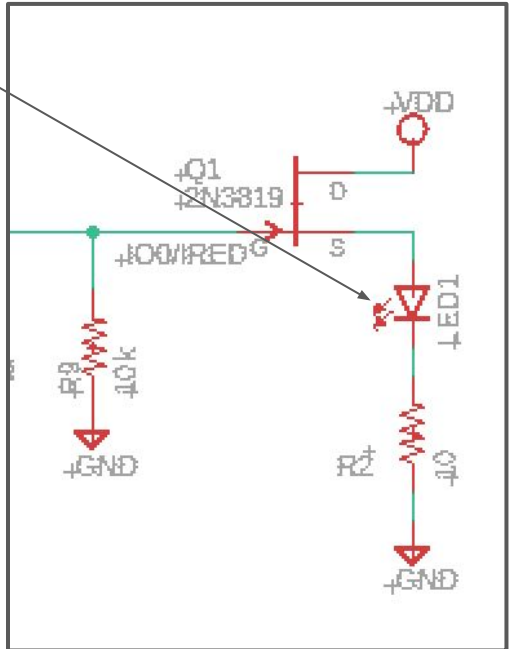
VON ~ 1/2  
VPWR

Receiver Read



85 mA!

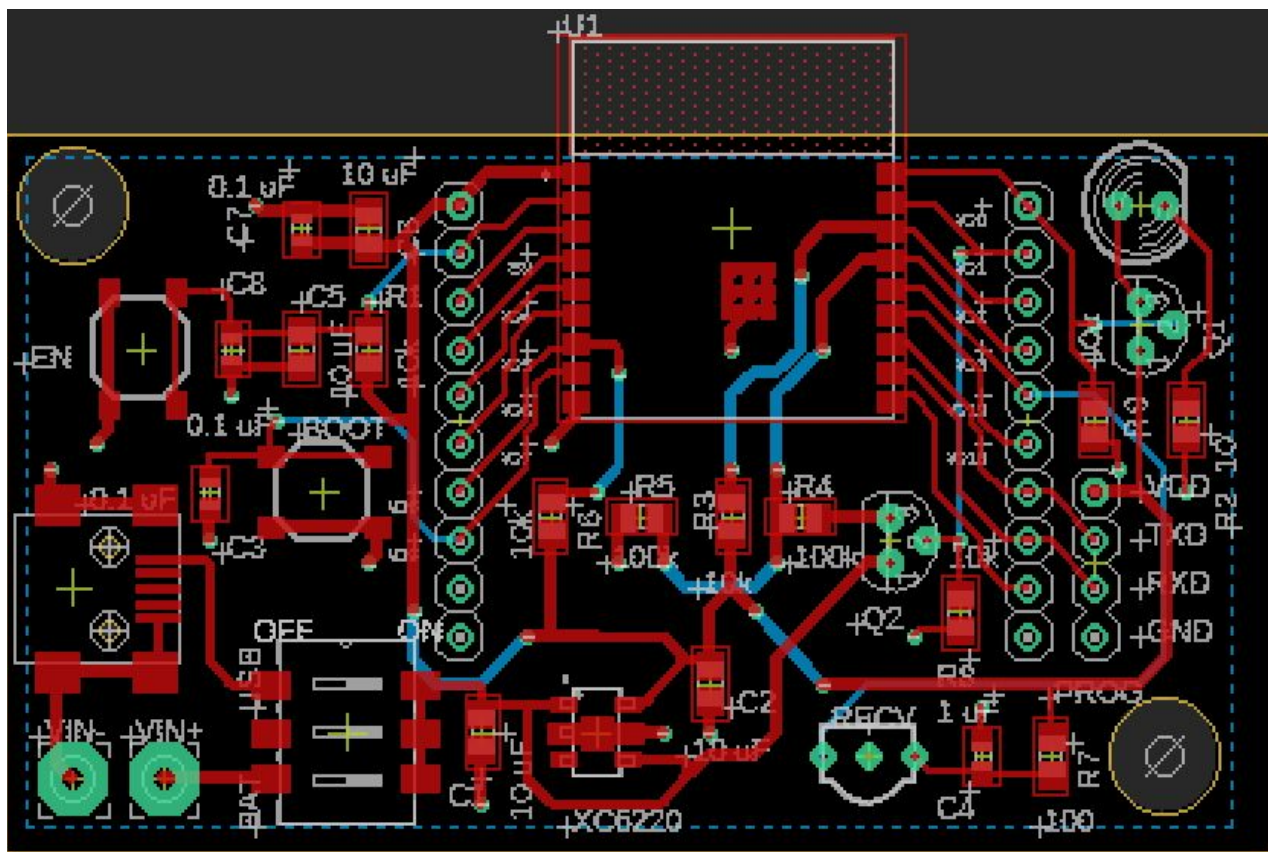
IR Actuate



# Pin Assignments

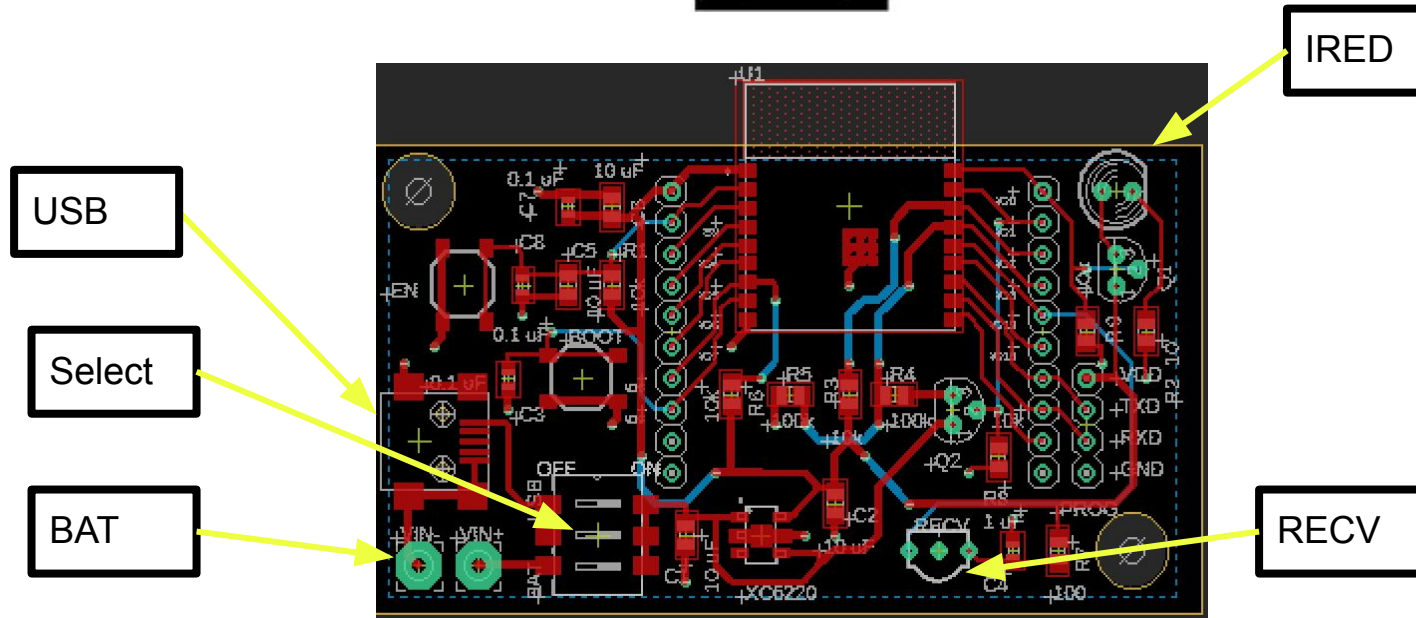
Pin #	Purpose
IO1	Actuates N-MOSFET to allow current flow for reading the battery.
IO19	Digital input pin for the receiver to send data to.
IO3	Analog read pin for the battery (HAS ADC!!!)
IO0	Send pin for the IR signal, releases N-MOSFET gate to allow current through the IRED
IO2, IO8, IO9	Strapping pins, same as last semester.

# Review of What We've Done: Hardware





# Review of What We've Done: Hardware



# Review of What We've Done: Hardware

Some things to note:









1. Option for user to battery versus USB-C power address batteries dying as a problem. Monitoring of batteries provides supplemental user-friendly power concerns.
2. IRED points towards TV, receiver towards audience.
3. In the final phase of this project, a 3D printed casing will be provided to enclose PCB while allowing antenna, receiver, IRED, batteries, and switch to be accessed. It will essentially be a mountable plastic prism with different openings.

Name	Q	Description	Package	Link
ESP32-C3	1	Microprocessor	MODULE_ESP32-C3-W ROOM-02-H4	<a href="#">Link</a>
XC6220	1	Voltage Regulator	XC6220SOT89-5	<a href="#">Link</a>
USB-MINI-B	1	USB-C connector	USB-MINI-FCI10033527	<a href="#">Link</a>
SWITCH-PB4X4	2	Push button	PBSMT4X4M	Not needed
PINHD-1X4	1	4x1 Pin Header	1X04N	Not needed
PINHD-1X10	2	1X10 Pin Header	1X10	Not needed
2N3819	2	N-Type Transistor	TO92	<a href="#">Link</a>
TSHG6210	1	IRED	T-1 $\frac{3}{4}$	<a href="#">Link</a>
TSOP38238	1	Receiver	TSOP382	<a href="#">Link</a>
1-1825059-2	1	Switch	SWSO6_325X138P100	<a href="#">Link</a>
LampVPath - 3AA	1	AA Battery Holder	N/A	<a href="#">Link</a>

In Design

In Use

Available

Name	Version	Managed Folder	Description
led	5	 Eagle Pcb	LEDs
newSDlib			
rcl	11	 Eagle Pcb	Resistors, Capacitors, Induct...
 SparkFun-Sensors			
supply2	2	 Eagle Pcb	Supply Symbols
 Switches			
transistor-fet	5	 Eagle Pcb	Field Effect Transistors
wirepad	2	 Eagle Pcb	Single Pads

Browse...

Search for Library Name

# Final Hardware Notes

- A final “go to market” design is currently being developed.
  - No pin headers other than PROG
  - No EN or BOOT push buttons for strapping pins
  - There will still be a switch to allow users to pick if they want batteries or if they have a cable configuration to avoid the problem of changing batteries
- A tabulation of passives and their values / package sizes has been made. Order form will go out when PCB is ordered / finalized. Nothing too crazy on that front - just 0805s and 0603 for caps and resistors.
  - Only weird one is a couple resistors will be really big (~1 M) or somewhat small (~5 ohms).
- Pin Modes have been determined for the C3, shouldn't be a problem.
- Files will be attached in Canvas submission

# What We've Done So Far: Low Level Implementation

We have our PCB and subcircuits figured out, and using the kit-board, we've shown we can send signals. But we need to dig a little deeper.

1. Non-volatile memory in our ESP32 world (and how to be smart with it)
2. Structure for profiles and custom buttons
  - a. Various TV's / protocols
  - b. Multi-input buttons
  - c. Various changeable profiles
3. Low level interface for adding new buttons
  - a. Uses receiver, hence why it is in the PCB design
4. Low level interface for sending buttons across different TVs.

# Memory Inspired by Website

We wanted the following options in the user interface:

1. Different profiles with names (like Netflix accounts)
2. Each profile could customize their hot-keys / buttons
3. Each profile is specific to a type of TV (changeable)
4. Custom buttons can cross different IR remotes (not limited to your samsung TV, but maybe could turn on other IR devices)

These constraints, which are specific versions of the original requirement, imposed a great challenge on how to organize / store data.

# Hierarchical Structure

## Website

- Profile 1
  - Name
  - Default protocol for standard buttons (on, off, up, down, etc.) - automatically on the remote
  - Custom Button One
    - Name of button
    - Protocol of button
    - Series of data {0xabcd..., 0xabcd ...} (not limited to signal thing but can be recorded)
  - Custom Button Two (etc.)
  - ...
- Profile 2 (etc) ...



# Nitty Gritty of How it was Accomplished

Since it took up a lot of the development time, we thought it best to share how it was done briefly.

- Non-volatile memory in ESP32-C3 series (and other series) has documentation found [here](#). In essence, it is just a **space-limited hash table**.
- We generated c-structures for “button” and “profile”, where each profile can have up to 3 custom “button” structs. The structs hold metadata that allow for actuation.
- We essentially made our own API for writing and reading these abstract structs into C3 memory (see NVM.h) Example code [here](#).

# Nitty Gritty Cont.

```
1  /**
2   * NVM.h
3   * A student developed API that write to the C3's NVM for a specific case
4   * The case is the structs "button" and "profile" defined below
5   * They are metadata for a larger website project.
6   */
7  #include <stdio.h>
8  #include <inttypes.h>
9  #include "freertos/FreeRTOS.h"
10 #include "freertos/task.h"
11 #include "esp_system.h"
12 #include "nvs_flash.h"
13 #include "nvs.h"
14 #include "Arduino.h"
15 #include <malloc.h>
16
17 struct button
18 {
19     char *name;
20     char *protocol;
21     uint64_t data[5];
22 };
23
24 struct profile
25 {
26     int number;
27     char *name;
28     char *protocol;
29     button buttons[3];
30 };
31
```

```
31
32 > uint64_t flash_read_u64(const char *key) ...
73
74 > char *flash_read_str(const char *key) ...
118
119 > void flash_write_u64(const char *key, uint64_t value) ...
155
156 > void flash_write_str(const char *key, const char *value) ...
192
193 > button flash_read_button(int p_num, int b_num) ...
321
322 > void flash_write_button(int p_num, int b_num, button b) ...
449
450 > profile flash_read_profile(int p_num) ...
486
487 > void flash_write_profile(profile p) ...
520
521 > void button_reset(int p_num, int b_num) ...
531
532 > void profile_reset(int p_num) ...
549
550 > void flash_reset() ...
556
557 > void print_button(button b) ...
566
567 > void print_profile(profile p) ...
```

# How to use the structures?

Now we have these cool structures that can organize and hold our metadata and we have a non-volatile API that stores them in the C3's big ole hash table. How do we use them?

- Develop a function that records IR input from the receiver, interprets stream of data as single “button” in the button struct, save that to someone’s profile in memory.
- Develop a function that take a specific button struct, loads it from memory, and actuates its bit stream into a send signal.
- Any text, button click, or personalization gets immediately saved and written to NVM using these functions.

# WiFi from boot up?

When the ESP32 first moves to a new location, it obviously will not have the WiFi credentials stored. Just like a smart TV, we need to enter those. How do we do it?

- We treat the ESP32 as its own network
- Connect our phones to that
- Enter WiFi information which gets stored in memory
- Then reboot on the router's network. See [images](#) below:



# How does that work?

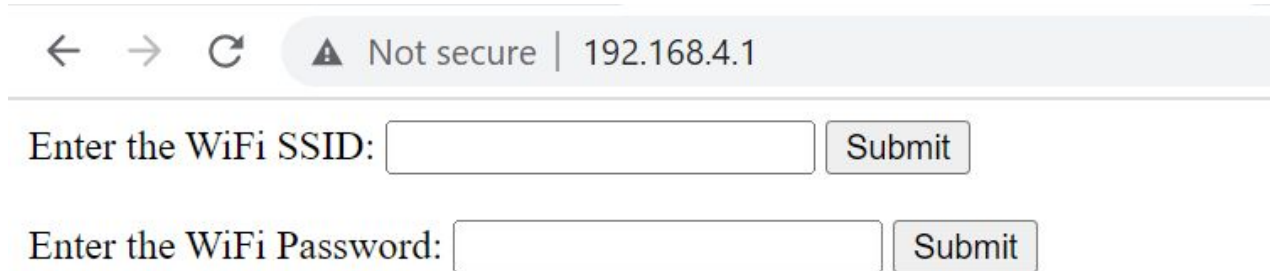
The following code snippet sets up the ESP32's personal network.

```
WiFi.softAP(ap_ssid, ap_password);  
IPAddress IP = WiFi.softAPIP();  
Serial.print("AP IP address: ");  
Serial.println(IP);
```

We get to pick the `ap_ssid` and `ap_password`. In this example, I picked “ESP32” and “password” respectively. We now connect with our laptop WiFi and enter the corresponding password to enter the network.

# How does that work?

- Once we're on the ESP32's access point network, we've built a small site to allow users to enter their ssid and password for their own home WiFi.
- Once both are entered, the ESP32 will stop hosting the site, and the user should reset the device with the ESP32 now working on the right network.
- It saves this information to nonvolatile memory. (picture of site)



The image shows a browser address bar with navigation icons (back, forward, refresh) and a warning icon. The address bar text is "Not secure | 192.168.4.1". Below the address bar are two form fields. The first field is labeled "Enter the WiFi SSID:" and has a "Submit" button to its right. The second field is labeled "Enter the WiFi Password:" and also has a "Submit" button to its right.

← → ↻ ⚠ Not secure | 192.168.4.1

Enter the WiFi SSID:

Enter the WiFi Password:

# How does that work:

FLOW CHART:

NEW WIFI LOCATION?

YES - Loaded WiFi credentials are wrong, connection stalls, ESP32 opens its own access point and waits for user input.

Upon info being entered, ssid and password are written to memory, ESP32 can reboot.

NO - Loaded WiFi credentials are right, WiFi and site load normally on reboot.

# Constant IPs for this demo device:

Every ESP32 will have a constant IP that it prints to the serial monitor. This will be written beforehand with instructions for that user.

ACCESS POINT IP: <http://192.168.4.1/>

WEBSITE IP: <http://192.168.10.147/>



# Website Functionality

1. Ability to boot on to WiFi after restart + ability to boot when in new WiFi
  - a. See previous slides
2. Profile selection that saves user data in NVM
  - a. Changes in protocol
  - b. Changes in custom button names
  - c. Changes in custom button functionality
  - d. Changes in user name
3. Protocol selection that allows users to change the type of TV their remote defaults to.
  - a. NEC will not work with a SAMSUNG TV

# Website Functionality

## 4. Custom Buttons to actuate multistep clicks

- a. Due to timeout issues, button clicks have to be pretty quick as of right now
- b. Naturally stops recording after 2 seconds of non-user input
- c. Due to development constraints, the number of compound buttons limited to 5
- d. Each user gets 3 for a total of 9 custom buttons on this implementation.

## 5. Stored memory even on reset

- a. Press the “EN” button or unplug the thing, and you’ll see when it resets all the data is there.

## 6. Multiple users

- a. No user hierarchy
- b. Both can be on different profiles sending different things.

## 7. NOTE!!!

- a. We understand there are currently no “ENTER” or “UP, DOWN, LEFT, RIGHT” buttons. This was an oversight.
- b. Adding buttons is relatively easy now that we know what we’re done. It should not be a problem and the infrastructure is there.

### ESP32 Web Server - TV Remote

**Collin** Dad Kids

Change Profile Name:  Submit

SAMSUNG

NEC

Battery

ON OFF VOL UP VOL DOWN

1 2 3  
4 5 6  
7 8 9  
0

Up3 - \_\_\_ - \_\_\_

Custom Button 1:  Submit  
Custom Button 2:  Submit  
Custom Button 3:  Submit

# Original Goals Line Up

Generally speaking, we met and / or exceeded the original goals:

- We can easily “talk” to a TV using our website in a robust manor, conquering points (1-2).
- Multiple TVs and multiple users (3).
- UI (4)
  - a. Custom buttons that allow for functionality across different protocols and devices on a single profile.
  - b. Generally good looking HTML (for electrical engineers)
  - c. Intuitive usage.

# Where can we improve

1. Limitation of functionality - Due to **storage** constraints in memory, **time** / manpower constraint, and lack of types of **TVs to test on**, functionality has the following limitations:
  - a. Recorded custom buttons max out at 5 inputs.
  - b. Saved profiles are set to 3 total profile.
  - c. Protocols are set to just 2.
2. No hierarchy - Multiple users can control at once, but there is no single user in “control” of the remote.
  - a. Not necessarily a bad thing - this was a minor aspect of our original pitch and in retrospect may have been an unnecessary feature.
3. UI Limitations
  - a. Make the text boxes a little prettier
  - b. Make the protocol thing a drop down menu. Maybe the same for profiles.

# Moving forward: what to do in home stretch?

1. Order parts and build board
2. Build physical encasing structure based on dimensions
3. Attempt to implement better UI (drop downs, textbox)
4. Add move and enter buttons (embarrassing)
5. Documentation.