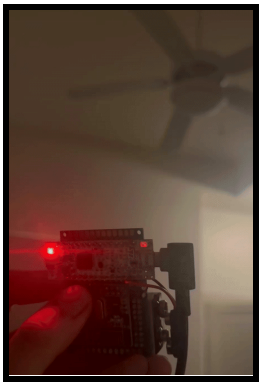
Project Lux

Kyle Chong

September 2025



1. Introduction

1.1 - Overview

*Project Lux is a wearable, low-power **watch-based transmitter** that detects **finger snaps** to wirelessly **turn on home lights**. A wall-powered "middleman" (ESP32 + nRF24L01+) receives the packet and switches Philips Hue bulbs through the local bridge API.*

The project has 4 major goals:
1) accurately detect when I snap my fingers, 2) successfully fire a wireless transmission upon trigger, 3) continuously function at low power, and 4) maintain a sleek, "invisible" appearance.

1.2 - Roadmap

[✓] Wireless Communication Protocol
- Designed and implemented a robust 2.4GHz protocol for the E01-ML01S transceiver.
- Built and validated the receiver module, achieving a 100% packet success rate with proper IC and actuator decoupling.

[✓] Hardware Design and Integration
- Designed and fabricated a custom PCB to integrate the MCU, sensor, and power management circuitry in a wearable form factor.
- Assembled a fully functional prototype, packaging the custom hardware and battery into a compact watch enclosure.

[In Progress] Snap-Detection Firmware
- Developing accurate snap-detecting algorithm using the onboard piezo sensor (Currently at 20% accuracy, looking for >90%)
- Currently evaluating performance trade-offs between the piezo sensor and a MEMS microphone + IMU solution for future revisions

[Future] Low-Power Optimization
- Implement interrupt-driven, ultra-low-power sleep modes to achieve a target battery life of 3+ months

Notes:

-nRF24L01+ and E01-ML01S may be used interchangeably in this document. The nRF24L01 is the standard version that was used for testing and cruder receiver modules, while the E01-ML01S is a slimmer version of the same part that lives on the transmitter watch.
-The project is currently in active development, and this file will be updated regularly.

2. Transmitter Module

2.1 - System Architecture

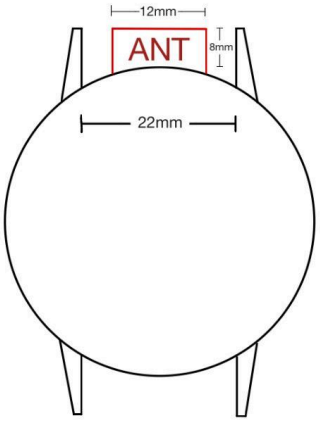


Figure 2.1
(Top View)

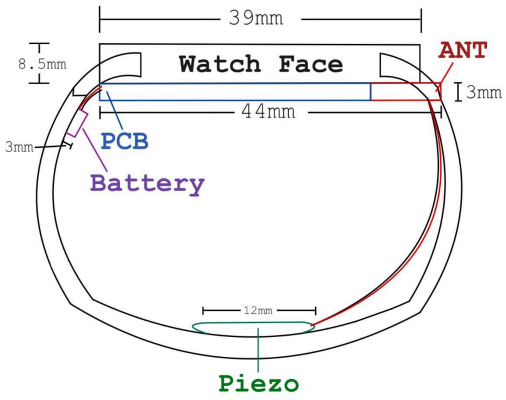


Figure 2.2
(Side View)

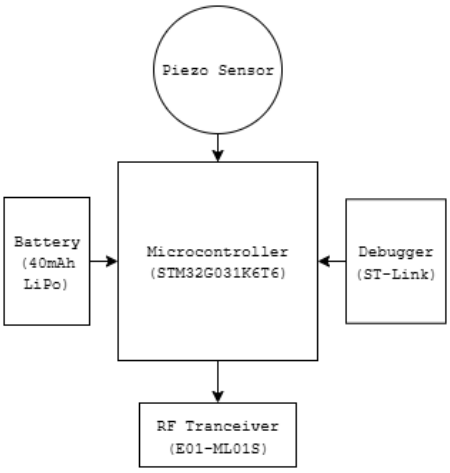


Figure 2.3
(Block Diagram)

When integrating the hardware into the watch, the primary considerations were the size and location of the PCB. In order to maintain a natural, invisible look and feel, the PCB was reduced down to a 3mm thickness. Additionally, the wireless antenna needed to be kept away from metal, so the PCB housing was designed to overhang off the edge of the watch, protruding through the watch strap via a hole cut by an oblong leather punch.

2.2 - PCB Design

The PCB flashes code via an ST-Link v3 miniE, along with a TC2070 adapter, which is placed onto the TC2070 footprint (Figure 2.3, top left corner). Although I would recommend outsourcing assembly (as the smallest footprints are 0603), the E01-ML01S should be soldered by hand, as it hangs over the edge of the PCB, which can cause issues in assembly.

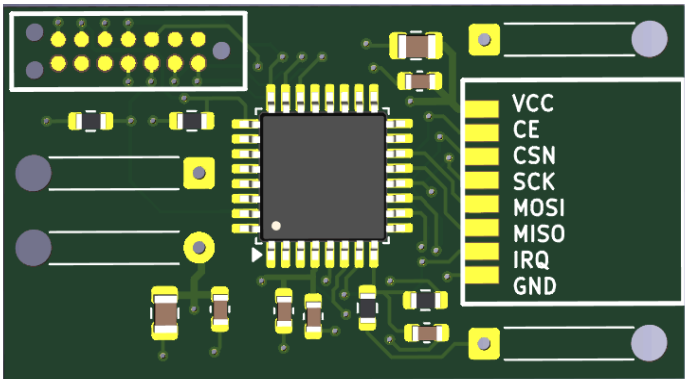


Figure 2.4

*Gerbers can be found in Github

2.3 - PCB Housing

I used an SLA printer for the PCB housing, as my nozzles for alternate printers inconsistently printed the .4mm walls that my housing used. Again, the wireless antenna must hang over the edge of the watch, therefore, a hole should be made in the watch strap. I used oval leather punches from amazon, which are linked in Github.

*.stl files can be found in Github

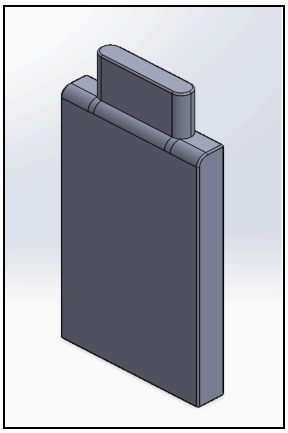


Figure 2.5

2.4 - Bill of Materials

Function	**Part Name**	**Quantity**	**Footprint**
MCU	STM32G031K6T6	x1	QFP
RF Transceiver	E01-ML01S	x1	N/A
Battery	LiPo (40mAh)	x1	N/A
Piezo Sensor		x1	N/A
Programmer	ST-Link v3 MiniE	x1	N/A
R - Limit	10k	x1	0603
R - Bleed	1M	x1	0603
C - Decoupling	4.7nF	x1	0805
C - Decoupling	100nF	x3	0603
C - Decoupling	1uF	x1	0603
C - Decoupling	22uF	x1	0603

3. Receiver Module (V2)

3.1 - Smart Bulb Receiver (“Middleman”)

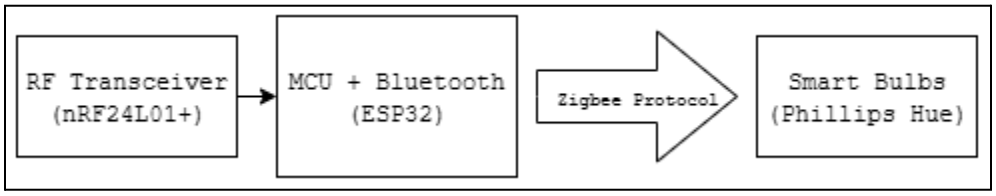


Figure 3.1

The mechanism I used to activate my lights is the Philips Hue smart bulb, triggered via an ESP32 using Arduino code. This required the Philips Hue bridge in order to talk to the lights. It should be possible to talk to the bulbs directly if they have bluetooth, but it will be far more difficult and require

As the ESP32 requires a heavier power draw than my watch transmitter can use, I use the ESP32 + nRF24L01+ as a middleman. The system is plugged into the wall at all times via USB. When the nRF receives a signal from the transmitter, the ESP is triggered, toggling my smart bulbs on and off.

3.2 - Middleman Architecture

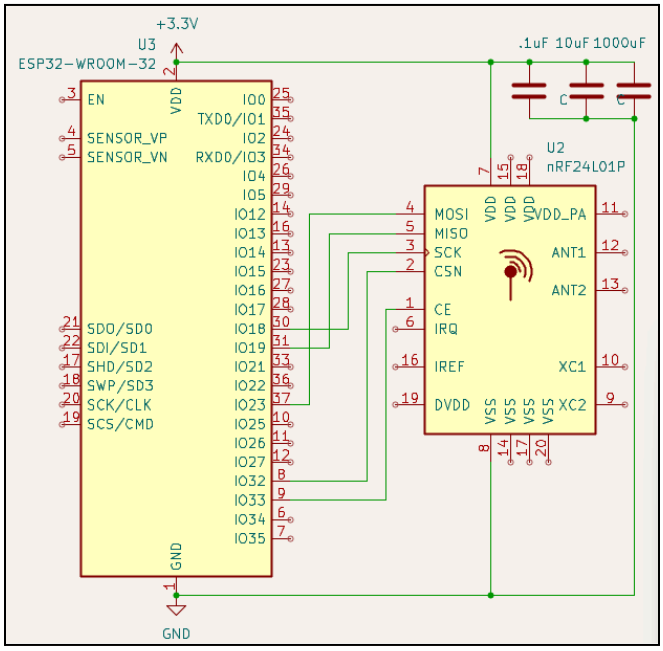


Figure 3.2
(Circuit Diagram)

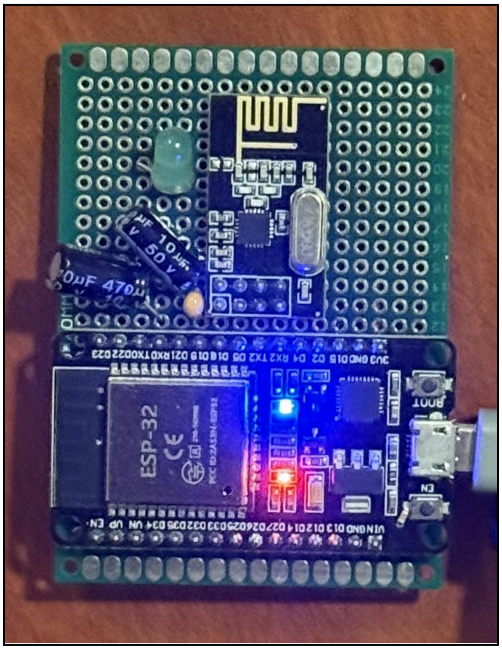


Figure 3.3
(Physical Module)

In addition to the VDD and VSS pins, (nRF MUST operate at 3.3V), there are 5 critical pins for nRF24 operation. MOSI, MISO, and SCK pinouts are usually noted on a given ESP32 diagram, while CSN and CE are user provided. My CSN pin is GPIO32, and my CE pin is GPIO33. Additionally, my nRF required 3 decoupling capacitors, at .1uF, 10uF, and 1000uF. I usually find the final bulk capacitor to not be necessary on the nRF24L01+, but on this system the nRF could not fire until the 1000uF was added. Connect all capacitors and nRF24 at 2 nodes, for power and ground respectively.

3.3 - Bill of Materials

Function	**Part Name**	**Quantity**
MCU	ESP32-WROOM	x1
RF Transceiver	nRF24L01+	x1
C - Decoupling	.1uF	x1
C - Decoupling	10uF	x1
C - Decoupling	1000uF	x1

4. Receiver Module (V1)

4.1 - Servo Receiver (Alternate mechanical trigger)

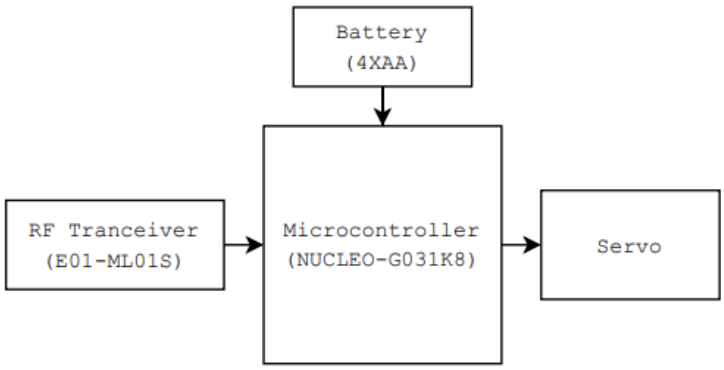


Figure 3.4
(Block Diagram)

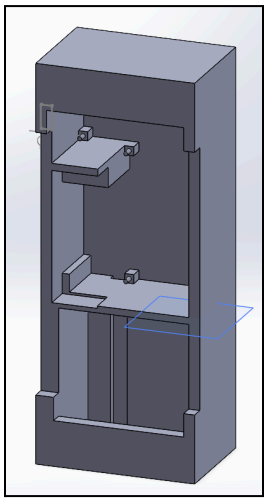


Figure 3.5
(CAD Model)

My initial plan for the receiver module was a servo that would fire upon transmission, physically toggling my light switch. This prototype was fully functioning, but I ran into issues with consistency in toggling the light switch. While this would not have been a

terribly difficult issue to fix, I found the usage of smart bulbs to be a more elegant, streamlined solution, with no bulky module over my light switch. However, this is a cheaper, DIY alternative to the ESP + Smart Bulb combo, and files will be linked in the github. Mechanical fine-tuning will likely be required for 100% accuracy, but it should be an intuitive problem to work on.

4.2 - Servo Receiver Architecture

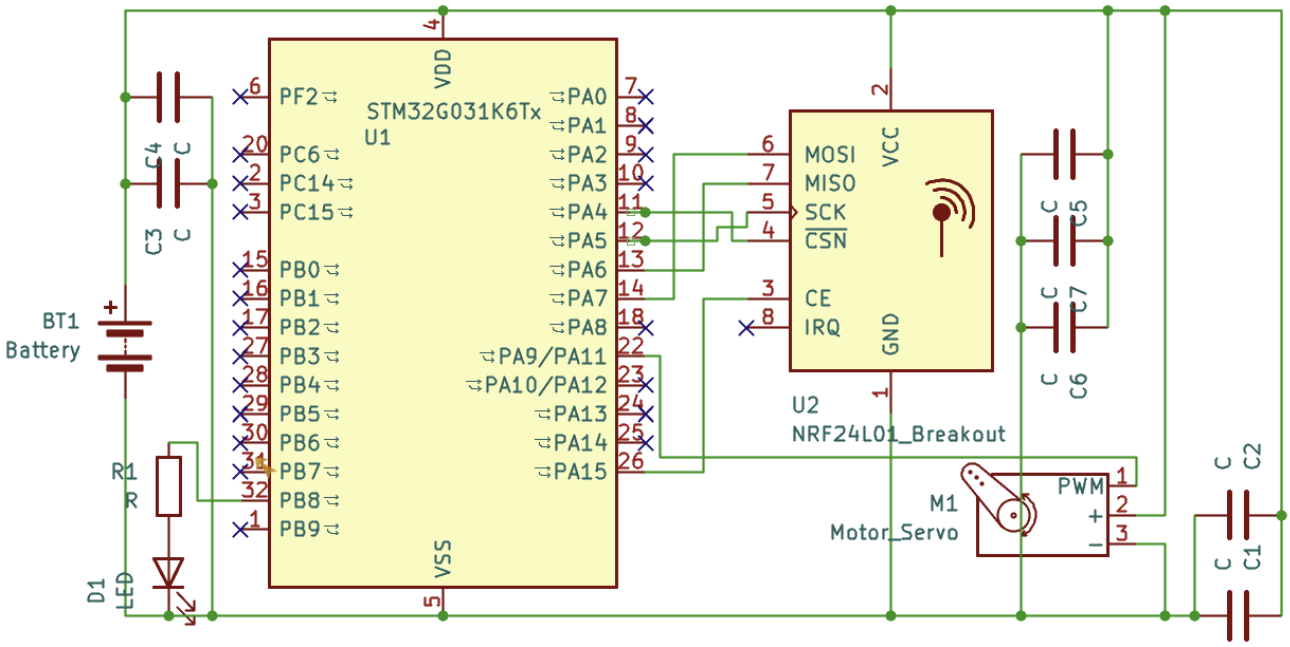


Figure 4.2

The wiring for this receiver module is more complex than the middleman module, as this one is designed to be powered by batteries rather than the outlet. This means that the servo, the batteries, and the nRF require decoupling, with the values listed below. However, if wired power is desired, you can simply remove the battery pack and plug in the Nucleo-G031 via USB.

4.3 - Bill of Materials

Function	**Part Name**	**Quantity**
MCU	STM Nucleo-G031K8	x1
RF Transceiver	nRF24L01+ (or E01-ML01S)	x1
Battery	AA (Alkaline/NiMH)	x4
Battery Holder (4xAA)	Generic	x1

Motor	HobbyPark Servo 20KG	x1
C - Decoupling	.1uF	x3
C - Decoupling	10uF	x2
C - Decoupling	1000uF	x2

5. Code

My project uses two IDEs. STM32CubeIDE is the primary one, used in almost every part of the project, specifically the nRF transmitter, nRF receiver, and sensor inputs. Arduino IDE is also used, only living on the ESP32 in order to speak to the smart bulbs. This section will contain basic overviews of the code as well as instructions for how to use it.

The full code files are on Github!

5.1 - STM32CubeIDE settings

Pinout Settings:

Pin Number	**Pin Name**	**Pin Type**	**Settings** (No pull-up/down unless specified)
PA0	NRF_IRQ	GPIO_EXTI0	EXTI Mode with Rising Edge
PA1	ADC1_IN1	ACD1_IN1	Analog Mode
PA2	USART2_TX	USART2_TX	Alternate Function Push Pull/Low
PA3	USART2_RX	USART2_RX	Alternate Function Push Pull/Low
PA4	NRF_CSN	GPIO_Output	High/Output Push Pull/Low
PA5	NRF_SCK	SPI1_SCK	Low/Alternate Function Push Pull/Low
PA6	SPI1_MISO	SPI1_MISO	Low/Alternate Function Push Pull/Low
PA7	SPI1_MOSI	SPI1_MOSI	Low/Alternate Function Push Pull/Low
PB0	LED_OUT	GPIO_Output	Low/Output Push Pull/Low
PA11	SERVO	TIM_1CH4	Alternate Function Push Pull/Low
PA13	SYS_SWDIO	SYS_SWDIO	
PA14	SYS_SWCLK	SYS_SWCLK	

PA15	NRF_CE	GPIO_Output	Low/Output Push Pull/Low
PB9	BUTTON	GPIO_EXTI9	Alternate Function Push Pull/Low

5.2 - nRF24L01+ Code (CubeIDE)

This module uses four files: `nrf_drivers.c`, `nrf_drivers.h`, `main_tx.c`, and `main_rx.c`. Add the driver pair to your project, then build either the transmitter or the receiver by selecting the corresponding entry point. Instead of commenting out `main.c`, use separate build configurations (TX/RX) or a single `main.c` that conditionally compiles the TX/RX app. The transmit-path initialization is adapted from MYaqoobEmbedded's nRF24L01 STM32 HAL driver (credited in the source headers per the license).

5.3 - Arduino Code

Make sure to adjust the Wi-FI SSID, Wi-Fi password, bridge IP, and username. Follow these steps in order to get your bridge IP and create your username.

[Get Started - Philips Hue Developer Program]