Implementing a Remote Switch Using the E220 LoRa Module

William Lucid, July 22, 2024

Introduction:

Goal: Conserve battery power for a Wyse Cam3 video camera that has a current draw 180 mA at 5 volts. Camera viewing is web based and has a very low demand to view camera; less than five views per day. Using a remote switch controlled by Async Web Server request will prevent battery draining power twenty-four hours a day. Currently a 10,000-mA battery bank only lasts a day.

Overview of the Project:

E220 Remote Switch project will reduce battery consumption by utilizing a countdown timer; started upon web request, turned off on countdown timer expiration. When web request is severed the E220-900T30D Sender module will send a wake-on-radio (WOR) message. Receiver module wakens from the Sleep Mode. Sleep Mode is an ultra-low current state in the microamp range. ESP32 will wake from Deep Sleep and turn on battery power; at the same time the E220 module receives the WOR message. When countdown timer has expired; E220 module will send a message, switching battery power off and putting the ESP32 into deep sleep and the receiving E220 module into sleep mode. Cycle repeats on new web request.

Brief description of the E220-Remote-Switch project:

E220-Remote-Switch project utilizes two, Ebyte E220-900T30D RF modules and two, ESP32 microcontrollers. Current project status; INA226 Battery Monitor and KY002S MOSFET switch have not been implemented in this update.

Demonstration mode:

1. ESP32 Receiver; push receiver reset button; this puts the ESP32 into deep sleep.
2. Open browser to "http://10.0.0.27/relay"; this will create a web request for turning on battery power. and start a countdown timer to turn off battery power, then put the ESP32 receiver into deep sleep.
3. First message is the WOR message to wake the E220 Receiver module and the Deep Sleeping ESP32. Second message from E220 Sender is required to turn on battery power and start the countdown timer. Both messages are sent when server request arrives from a click on web link to view video camera.

Video: E220-Remote-Switch Demo with sleep current monitoring

Three advantages of using the Ebyte, E220-900T30D are increased distance 10 km estimated, at power of 30 dbm and Sleep current of 5 uA. E220-900T30D third feature is the ability to send a WOR message to wake up the receiving transceiver allowing second message; to turn on battery power.

Transmit current of 620 mA is almost instantaneous at 30dbm to send; up to 200 bytes, before dropping current. Receiving current; for a message, 17.2 mA. Current values are from Ebytes E220-900T30D User Manual. Measured Standby current 11.8 mA. E220 module is always drawing current; utilizing E220 Sleep Mode, Sleep current ranges from .54 uA to 97.33 uA the complete cycle of messages, measured with digital multimeter set for microamps. Occasional very brief, on order of a few milliseconds; meter over loads occurring due to message being transmitted or received.

Components Used:

E220 LoRa module; two E220-900T30D radio, transceivers used as Sender and Receiver. Suggest labeling Sender and Receiver. Ebyte E220-900T30D Module

ESP32 microcontroller; two ESP32 Devkit v1 microcontrollers; one for web server; one for receiving messages and switching battery power on. Doit ESP32 Devkit v1

MOSFET bi-stable, switch KY002 for battery switch; switch can handle 3-27 Volts at 1.5 Amps. 3-27V 1.5A Single Button Bistable Switch Module Flip Flop Latch Falling Edge Trigger Switch KY002

INA226 Battery Monitor 36 Volt, 20 Amp, I²C INA226 Battery Monitor

No low power optimization was attempted due ESP32 Devkit v1 being a development board. E220 module current consumption was lowered by using sleep mode.

Sender and Receiver E220 module connections:

Setup connections for Sender and Receiver require no wiring changes after the initial connections. Sender and Receiver mode of operation will use E220 library commands coded into ESP32 Sender and Receiver sketches.

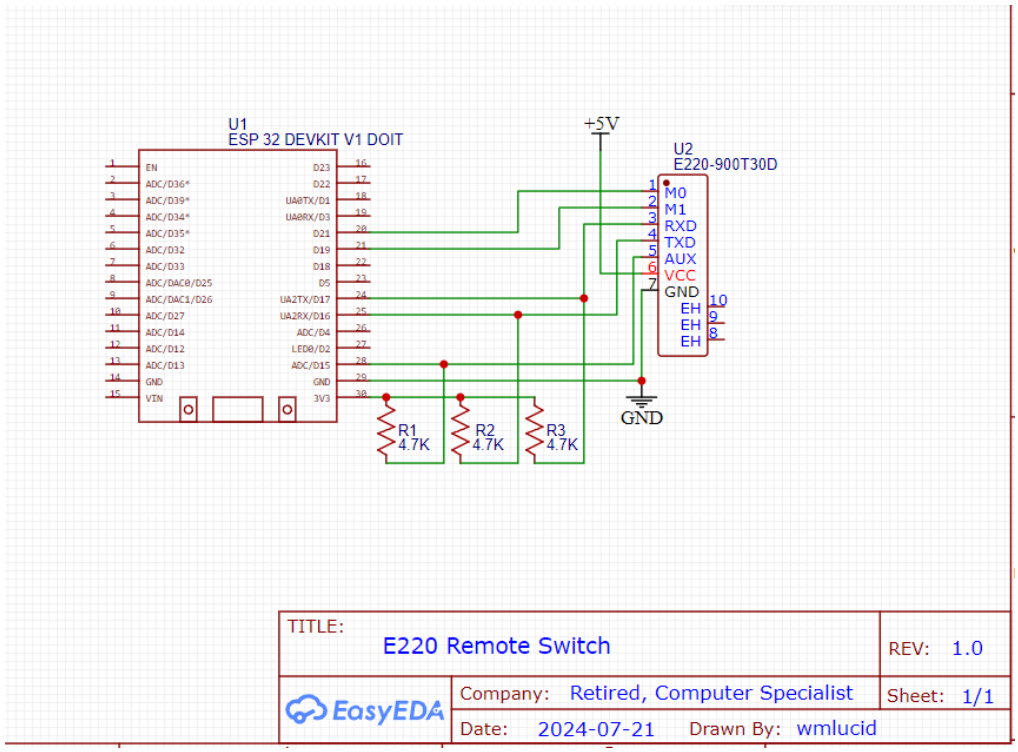
M0 = ESP32, GPIO21
M1 = ESP32, GPIO19
RX = ESP32, 17 TXD2
TX = ESP32, 16 RXD2
AUX = ESP32, GPIO15 (RTC GPIO pin impotant)
Vcc = 3.3 Volts or 5 Volts (only Vcc of E220 Module)
GND = GND

KY002S Bi-stable MOSFET switch will be included in future update; KY002S is emulated in this sketch with serial print statements.

INA226 Battery Monitor will be a future update; will monitor voltage, current, alarm at a set voltage, and log to a file.

Detailed schematic showing connections between ESP32, LoRa E220 modules:

E220-Remote-Switch --Sender and Receiver Schematic



Note: E220 Module AUX pin is connectd to GPIO15; which is a RTC GPIO pin, required to wake ESP32

Sender --required libraries for ESP32 and LoRa E220 module:

#include <Arduino.h> *

#include "WiFi.h" *

#include <WiFiUdp.h> *

#include <HTTPClient.h> *

#include <time.h> *

#include "LoRa_E220.h" Ebyte LoRa E220 Library

#include <AsyncTCP.h> AsyncTCP Library

#include "ESPAsyncWebServer.h" ESPAsyncWebServer Library

#include <Ticker.h> *

#import "index7.h" HTML web page stored in memory –Do not remove; except to replace with new “index7.h”.

Libraries with “*” are included in Arduino IDE, Board manager 2.0.17. Board manager 2.0.17 was used to compile sketches for this project.

Receiver --required libraries for ESP32 and LoRa 220 module:

#include "Arduino.h" *

```
#include "LoRa_E220.h"
#include <WiFi.h> *
#include <time.h> *
#include <FS.h> *
#include <LittleFS.h> *
#include "esp_sleep.h" *
#include "driver/gpio.h" *
#include "esp_system.h" *
#include <INA226_WE.h>   INA226_WE Library
#include <Wire.h> *
```

Libraries with "*" included in Arduino IDE, Board manager 2.0.17. Board manager 2.0.17 was used to compile sketches in this project.

Explanation of the process for sending structure Message message using the LoRa E220 module:

Project uses the E220 Library's containers to send structured messages; there are two structures one for time using NTP time from the Sender, since Receiver will have no Internet. Other structure Message uses integer variable data to switch the battery and the actual formatted timestamp; timestamping will be used in Battery Monitor log fille. Value of variable data; determines battery switch position on or off.

Container code for receiving structure Message message

```
  ResponseStructContainer rsc = e220ttl.receiveMessage(sizeof(Message));
  if (rsc.status.code == 1) {  // Check if the status is SUCCESS
    Message message = *(Message*)rsc.data;
    //Serial.println(message.switchState);  // This prints to monitor
    //Serial.println(message.dateTime);     // This prints to monitor
    rsc.close();
```

Sending of the WOR message is sent when the server request arrives. WOR was moved to a function to prevent it from sending the WOR message at sketch boot up and to be able to send two messages on one web user click to view camera.

WOR function:

```
void sendWOR(){
 e220ttl.setMode(MODE_1_WOR_TRANSMITTER);
 delay(delayTime);
 // Send message
```

```
  ResponseStatus rs = e220ttl.sendFixedMessage(0, DESTINATION_ADDL, CHANNEL, "Hello, world? WOR!");
  e220ttl.setMode(MODE_0_NORMAL);
  delay(delayTime);
}
```

Problems and their solutions:

Biggest issue; intermittent connection between breadboard and E220 module. Resolved by using female to male, Dupont wires with module to breadboard connections.

Another issue was the sending WOR message; then needing to send second message on the same web request. Solution was to include sending of second message in Async Web Server request.

Server request solution code:

```
  server.on("/relay", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, PSTR("text/html"), HTML7, processor7);
    sendWOR();  //Sends WOR message; moved the location for sending WOR message to sendWOR function.
    data = 1;  //Sets data to 1 used to turn on battery switch
    needAnotherCountdown = 1;  //Limits ticker once timer to be “reset” for multiple uses; instead of a single use.
    countdownTrigger();  //Starts countdown timer.
  });
```

SendWOR function:

```
  void sendWOR(){
  e220ttl.setMode(MODE_1_WOR_TRANSMITTER);
  delay(delayTime);
  // Send message
  ResponseStatus rs = e220ttl.sendFixedMessage(0, DESTINATION_ADDL, CHANNEL, "Hello, world? WOR!");
  e220ttl.setMode(MODE_0_NORMAL);
  delay(delayTime);
}
```

Function countdownTrigger():

```
void countdownTrigger() {
  // Perform countdown actions here
  Serial.println("\nCountdown timer triggered!\n");
  //getDateTime();
```

```
  // Schedule the next countdown if needed
  if (needAnotherCountdown == 1) {
    onceTick.once(60, ISRcamera);
    data = 1;
    switchOne(data);
    needAnotherCountdown = 0;
  }
}
```

Function switchOne() sends structure Message message; includes value of data and the formatted timestamp.

```
void switchOne(int data) {
  if (data == 1) {
    Serial.println("\nWaked up from external GPIO!");
    Serial.println("Wake and start listening!\n");
    delay(500);
    Serial.println("\nESP32 waking from Deep Sleep");
    Serial.println("Battery Switch is ON\n");
  }

  if (data == 2) {
    Serial.println("\nBattery power switched OFF");
    Serial.println("ESP32 going to Deep Sleep\n");
  }
  Serial.println("Hi, I'm going to send message!");
  e220ttl.setMode(MODE_1_WOR_TRANSMITTER);
  delay(delayTime);
   get_time();

  Message message;
   //initialize struct members
  message.switchState = data;

  // Initialize the dateTime
```

```
  String dateTimeStr = get_time();
  if (!dateTimeStr.isEmpty()) {
strncpy(message.dateTime, dateTimeStr.c_str(), MAX_dateTime_LENGTH - 1);
message.dateTime[MAX_dateTime_LENGTH - 1] = '\0'; // Ensure null-termination
}
Serial.print("switchState:  "); Serial.println(message.switchState);
Serial.print("dateTime:  "); Serial.println(message.dateTime);

// Send message
ResponseStatus rs = e220ttl.sendFixedMessage(0, DESTINATION_ADDL, CHANNEL, &message,
sizeof(Message));
// Check If there is some problem of succesfully send
Serial.println(rs.getResponseDescription());
}
int sendMessage(int data){
Serial.println("Hi, I'm going to send message!");
e220ttl.setMode(MODE_1_WOR_TRANSMITTER);
delay(delayTime);
get_time();
Message message;
//Initialize struct members
message.switchState = data;
// Initialize the dateTime
String dateTimeStr = get_time();
if (!dateTimeStr.isEmpty()) {
strncpy(message.dateTime, dateTimeStr.c_str(), MAX_dateTime_LENGTH - 1);
message.dateTime[MAX_dateTime_LENGTH - 1] = '\0'; // Ensure null-termination
}

//Serial.print("switchState:  "); Serial.println(message.switchState);
//Serial.print("dateTime:  "); Serial.println(message.dateTime);

  // Send message
```

```
  ResponseStatus rs = e220ttl.sendFixedMessage(0, DESTINATION_ADDL, CHANNEL, &message,
sizeof(Message));

  // Check If there is some problem of successfully send

Serial.println(rs.getResponseDescription());

}
```

List of applications for DIY Makers:

The E220-Remote-Switch, utilizing the LoRa E220 module and an ESP32, offers a robust solution for DIY makers looking to create remote-controlled projects. Here are some applications specifically tailored for DIY enthusiasts:

1. Home Automation Projects

Smart Lighting: Create a system to control home lighting remotely, enabling users to turn lights on or off from a smartphone or web interface.

Automated Curtains/Blinds: Design a project to open and close curtains or blinds based on time of day or light levels.

Garage Door Opener: Build a remote-controlled garage door opener for convenience and security.

2. Garden and Outdoor Projects

Automated Irrigation System: Develop a remote-controlled irrigation system to water plants at scheduled times or based on soil moisture levels.

Garden Lighting: Control outdoor lighting systems to enhance garden aesthetics and security.

3. Security Systems

Remote Door Lock: Implement a system to lock or unlock doors remotely for home security or access control.

Surveillance Cameras: Create a project to control the activation and angle of surveillance cameras.

4. Environmental Monitoring

Weather Station: Build a remote weather station to monitor and control sensors for temperature, humidity, and atmospheric pressure.

Air Quality Monitor: Develop a system to measure and report air quality parameters, with remote alerts for poor conditions.

5. Pet Care

Automatic Pet Feeder: Create a remote-controlled feeder to dispense food at specific times or on demand.

Pet Door: Build a remote-controlled pet door to allow or restrict pet access based on schedule or user command.

6. DIY Robotics

Robot Control: Design a remote-control system for robots, enabling navigation and operation from a distance.

Drone Projects: Implement remote control for DIY drones, managing flight patterns and actions.

7. Energy Management

Smart Plugs: Create remote-controlled smart plugs to manage power consumption of connected devices.

Thermostat Control: Develop a remote-controlled thermostat to regulate home heating and cooling systems.

8. Automotive Projects

Car Alarm System: Build a remote-controlled car alarm and tracking system for enhanced vehicle security.

Remote Start: Create a system to remotely start and stop the car engine.

9. Entertainment and Media

Remote-Controlled Audio System: Design a system to control home audio setups, switching between sources and adjusting volume remotely.

Projector Screen Control: Implement a remote-controlled system to raise and lower projector screens.

10. Custom IoT Projects

Multi-Sensor Networks: Create a network of sensors that communicate via the E220 LoRa module, with remote monitoring and control capabilities.

Data Logging: Develop a remote data logging system to collect and analyze data from various sensors.

Mailbox alert for arriving postal mail.

These applications highlight the versatility and potential of the E220-Remote-Switch for DIY makers, enabling them to create innovative and functional projects that enhance everyday life. --ChatGPT

Documentation and Resources:

William Lucid's Project code

E220-Remote-Switch project code

ESP32 Devkit v1 Pinout

High Resolution ESP32 Devkit v1 Pinout snd specs

INA226 and KY002S Connections

Connections for KY002S Switch and INA226 Battery Monitor

INA226 Module Details and Power Down

KY002S Bi-Stable MOSFET Switch --Details

ChatGPT

OpenAI's; ChatGPT was a team member assisting with technical issues, code snippets, examples, and optimizing code with my prompting, we were able to find information that was scarce or unavailable on the Internet and we were able to complete coding for this project.

Modes of Operation --ChatGPT

LoRa E220 Module --Modes of Operation

Renzo Mischianti's E220 Articles

Ebyte LoRa E220 device for Arduino, esp32 or esp8266: settings and basic usage

Ebyte LoRa E220 device for Arduino, esp32 or esp8266: library

Ebyte LoRa E220 device for Arduino, esp32 or esp8266: configuration

Ebyte LoRa E220 device for Arduino, esp32 or esp8266: fixed transmission, broadcast, monitor, and RSSI

Ebyte LoRa E220 device for Arduino, esp32 or esp8266: power-saving and sending structured data

Ebyte LoRa E220 device for Arduino, esp32 or esp8266: WOR microcontroller and Arduino shield

Ebyte LoRa E220 device for Arduino, esp32 or esp8266: WOR microcontroller and WeMos D1 shield

Ebyte LoRa E220 device for Arduino, esp32 or esp8266: WOR microcontroller and esp32 dev v1 shield

Github Ebyte E220 library

Mischianti Arduino LoRa shield (Open source)

Mischianti WeMos LoRa shield (Open source)

Mischianti ESP32 DOIT DEV KIT v1 shield (Open source)

Support Forum

Wolfgang Ewald's

Using LoRa with the EByte E220, E22 and E32 series

Appreciation:

I would like to express my gratitude to ChatGPT, an AI developed by OpenAI, for providing valuable assistance and guidance throughout the development of my E220 LoRa module project. The insights and support received were instrumental in overcoming technical challenges and optimizing the project's performance.

Thank you! Renzo Mischianti for your LoRa E220 Library (xReef), E220 Ebyte E220 articles, LoRa Ebyte module Community support, and the Ebyte E220 Support page.

Wolfgang Ewald thank you! Learned much from your article "Using LoRa with the EByte E220, E22 and E32 series". Thanks for sharing your hands on experiences.