Smart Backup Power Management and Real Time Monitoring System

Quazi Mohammed Monawar Mahtab
Department of Electrical & Computer Engineering
North South University
Dhaka, Bangladesh
quazi.mahtab@northsouth.edu

Anika Tabassum Chowdhury
Department of Electrical & Computer Engineering
North South University
Dhaka, Bangladesh
anika.chowdhury08@northsouth.edu

Ishrat Noor
Department of Electrical & Computer Engineering
North South University
Dhaka, Bangladesh
ishrat.noor@northsouth.edu

***Abstract*—This project presents a DC-based backup power management system using an ESP32 microcontroller and the Blynk mobile application. The system monitors the real-time power consumption of individual loads and allows remote control via Wi-Fi. During power outages, it prioritizes essential loads, ensuring efficient use of limited backup energy. This approach enhances load visibility, energy management, and intelligent operation in small-scale backup systems.**

Keywords—ESP32, Wi-Fi, DC Power, Power Monitoring System, Relay Switch, MOSFET, Resistors, Solar Panel, OLED Display, Energy, Arduino IDE

I. Introduction

The rapid advancement of embedded systems has made it easier to monitor and control electrical loads in both residential and industrial settings. Backup power systems, like traditional Instant Power Supply (IPS) devices, are commonly used to ensure power continuity during outages. However, conventional IPS systems typically supply power to all connected loads without considering their individual energy consumption or prioritizing essential devices. As a result, backup energy is often used inefficiently, and users lack insight into which loads are consuming the most power.

To address these challenges, this project uses an ESP32 microcontroller paired with the Blynk mobile application to manage a DC-based circuit with a solar-charged battery as a backup power source. The system allows for real-time monitoring of individual loads, remote control through Wi-Fi, and intelligent load prioritization. During power outages, critical low-power loads are kept operational, ensuring efficient use of the available backup energy while preventing system overload. By integrating real-time monitoring with smart load management, this system enhances energy efficiency and gives users greater control over their connected devices.

II. Technology Review

A. ESP32

The ESP32 is a highly integrated system-on-chip (SoC) with built-in Wi-Fi and Bluetooth capabilities, widely used in embedded systems and IoT projects. It features a dual-core processor, making it capable of handling multiple tasks simultaneously, and offers a range of peripherals, including ADCs, DACs, SPI, UART, and I2C interfaces. The ESP32's low-power consumption modes and high processing power make it suitable for applications such as sensor monitoring, data logging, and communication. Furthermore, the ESP32 supports various development environments, such as Arduino IDE and Espressif's native ESP-IDF, which makes it accessible to both beginner and advanced developers. Its compact size, coupled with its versatility and extensive support from the community, has made the ESP32 a popular choice in modern IoT applications.

B. Relay

The SRD-05VDC-SL-C is a popular and widely used electromagnetic relay, known for its reliability and low power consumption. This relay operates with a 5V DC coil voltage and is capable of switching AC or DC loads, making it suitable for controlling devices like motors, lights, and other electrical equipment. The relay features a single-pole, double-throw (SPDT) switch configuration, which allows for both normally open (NO) and normally closed (NC) contact positions. The SRD-05VDC-SL-C is favored for its ease of use in microcontroller-based projects, such as those involving the ESP32, due to its low control voltage and simple drive circuitry. It is a cost-effective and efficient solution for automated switching applications, providing isolation between low and high voltage circuits.

III. Methodology

A. System Design

The overall system architecture is illustrated in the block diagram (Fig. 1). The design consists of four main components: the primary power supply, a solar-charged backup battery, the ESP32 microcontroller, and the connected loads. The ESP32 acts as the central controller, handling voltage and current measurement, decision-making for load prioritization, and communication with the Blynk mobile application. An OLED display is also integrated to provide local, real-time system feedback.

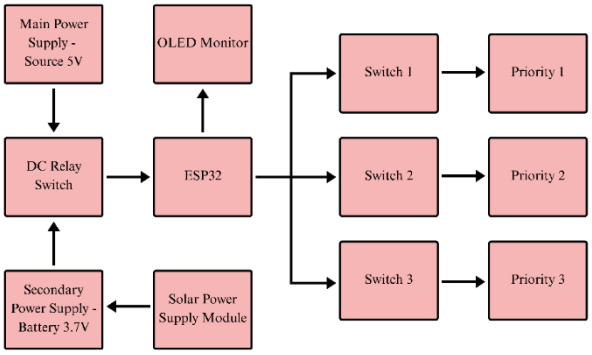

Figure 1. Flowchart of the Concept.

B. *Circuit Schematic*

The detailed circuit schematic was designed in Proteus, as shown in Fig. 2. The ESP32 interfaces with relays that control three loads of different priority levels. Current and voltage sensors are connected to monitor consumption, while the OLED module provides visual updates. The power supply is configured to automatically switch from the main supply to the solar-charged battery in the event of an outage.

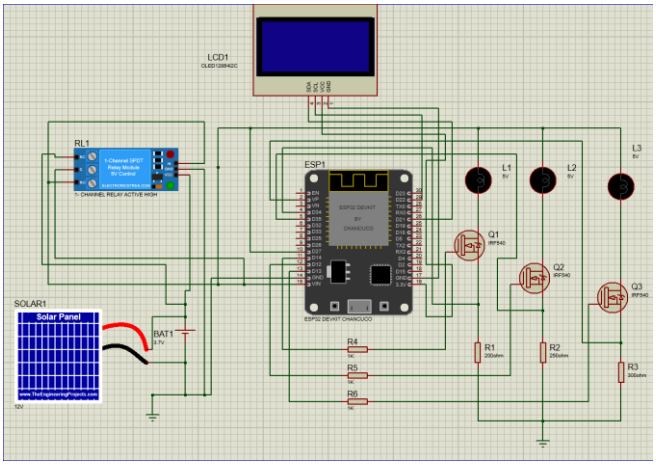

Figure 2. Circuit Schematic Designed in Proteus.

C. *Hardware Implementation*

The hardware prototype was assembled on a breadboard, as shown in Fig. 3. The ESP32 module is connected to relays, LEDs, and the OLED display. Three different colored LEDs are used to represent loads of varying priority levels. Red is the first load; Blue is the second load and Green is the third load. During testing, the system successfully switched between power sources, monitored current consumption, and allowed remote control via the Blynk mobile app. The OLED shows the voltage and power across the shunt resistors and Current is shown in the serial monitor in Arduino IDE.

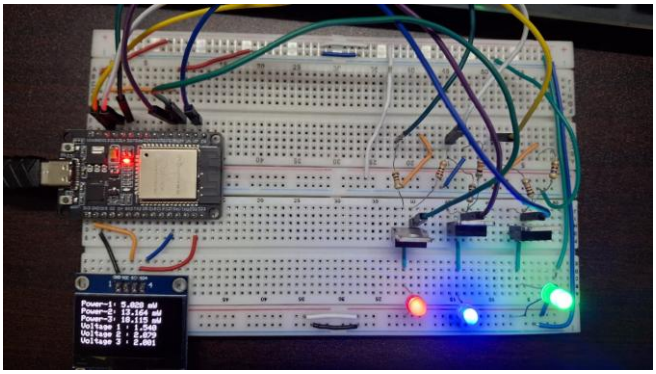

Figure 3. Hardware Implementation of the Circuit without Relay, biased via Main Line

D. *Hardware Implementation with Solar-Charged Battery*

The circuit was further tested by biasing the system with a solar-charged battery, as illustrated in Fig. 4. In this configuration, the solar-charged battery served as the primary backup source, supplying power directly to the connected loads. To demonstrate prioritized load operation under limited energy availability, only the first load (red LED) was kept active, while the second (blue LED) and third (green LED) remained off. This validated the system's ability to ensure continuity of essential loads while conserving backup energy. The OLED display provided real-time measurements of voltage and power across the shunt resistor, while current values were observed through the Arduino IDE serial monitor. The setup confirmed the effectiveness of solar energy integration into the backup management system, ensuring reliable operation during outages.

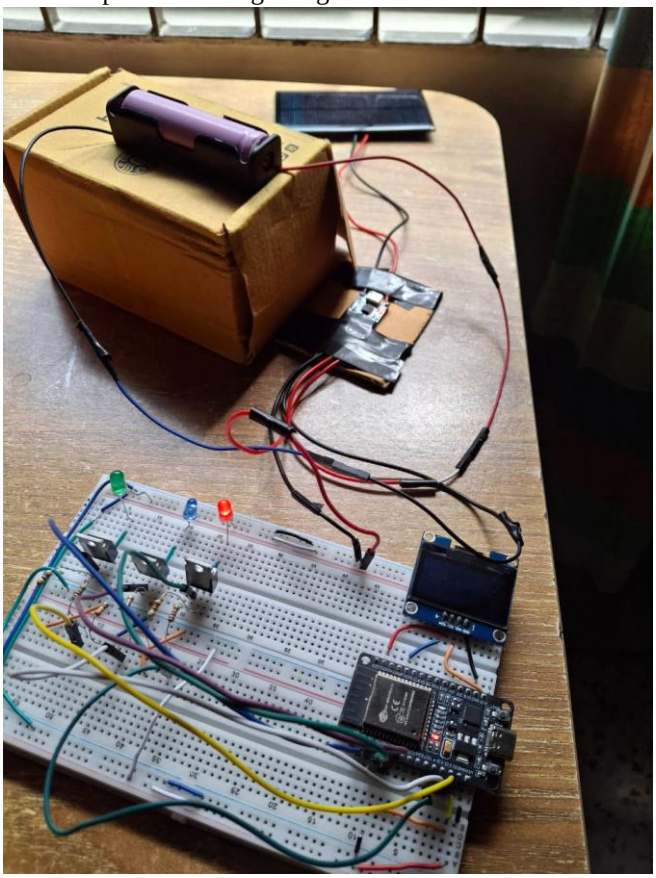

Figure 4. Hardware Implementation of the Circuit biased via Solar Charged Battery.

E. *Hardware Implementation with Relay Switching for Test for safety of the circuit*

A test setup was developed to evaluate the feasibility of relay-based power source switching, as illustrated in Fig. 5. A single-channel DC relay module was employed to alternate between two independent 5 V supplies: one from the Arduino Uno and another from a secondary ESP32 board. The

objective was to investigate the performance of the relay when managing dual inputs and delivering stable output to the loads.

During testing, the relay successfully switched between the two sources, demonstrating the validity of the switching concept. However, the output voltage exhibited instability, fluctuating significantly during transitions. As a result, the effective output remained low and insufficient to drive all loads. Only the first load, represented by the red LED, remained illuminated, while the other LEDs failed to activate reliably. This experiment emphasized that while relay switching is operationally viable, a regulated and stable DC supply is crucial for consistent system performance.

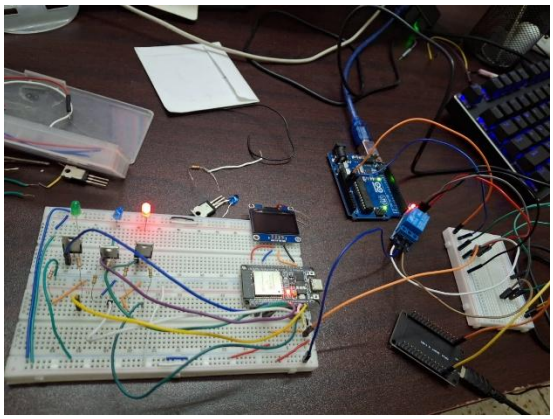

Figure 5. Hardware Implementation of the Circuit with Relay Switching using Dual 5 V Supplies (Arduino and ESP32).

F. Software Implementation

1. Environment Setup

The software was developed using the Arduino IDE with the ESP32 board support package. Essential libraries were installed: WiFi.h for Wi-Fi connectivity, BlynkSimpleEsp32.h for Blynk integration, and Adafruit_SH110X.h along with Adafruit_GFX.h for controlling the OLED display.

2. Pin Configuration

The software defined ADC pins for current sensing through shunt resistors and GPIO pins for controlling the three loads and a relay. It also included ADC pins for solar and battery voltage sensing in the earlier version of the project.

3. Current and Voltage Measurement

Functions were implemented to read analog values from the ADC pins, converting raw ADC readings into voltage (V) and current (A). Three shunt resistors were used to measure the current across the loads.

4. Load Control

The system was configured to control Load1, Load2, Load3, and the relay. Two versions of control logic were implemented:

Automatic Control (older version): Loads were switched based on current thresholds.

Manual Control (final version): Loads were controlled directly via switches in the Blynk app.

5. Blynk App Integration

A Blynk project was created with switches (V10–V12) to control the loads, LEDs (V6–V8) to indicate load status, and value widgets (V3–V5) to display current readings. Additional value widgets (V1–V2) were used for solar and battery voltage monitoring in the earlier version. The functions BLYNK_WRITE() and Blynk.virtualWrite() facilitated communication between the ESP32 and Blynk.

6. OLED Display Integration

The OLED display shows real-time current readings for all three loads and the system status. The Adafruit_SH110X and Adafruit_GFX libraries were utilized to display text and information on the OLED.

7. Debugging and Fixes

Issues such as board selection and Blynk LED status were fixed. Serial debugging was added to monitor load current values, load ON/OFF states, and solar/battery voltages. A delay (2000) was included for smoother updates without lag.

8. Final Software Features

The final system enables real-time current monitoring, manual load control through Blynk, and ON/OFF status indication via both the Blynk app and OLED display. Remote monitoring and control are facilitated through Wi-Fi, providing a comprehensive solution for load management.

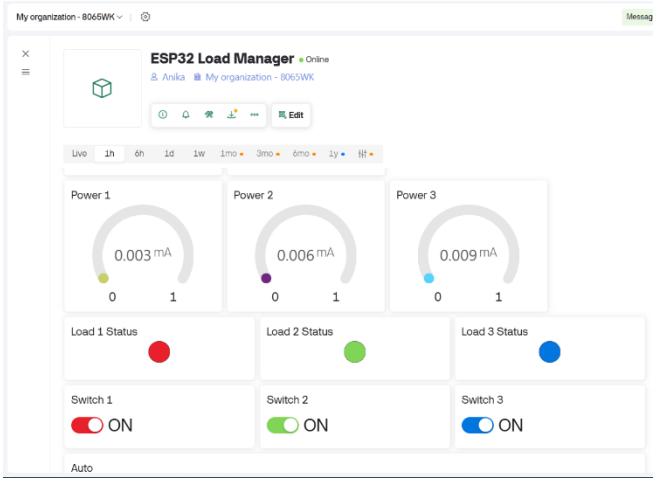

Fig 6. Monitoring and Controlling through Blynk Server.

IV. Results and Discussions

The serial monitor displays the voltage and current measurements, which are then used in the power calculation included in the code. All three LEDs operate correctly and their corresponding values are shown in the Blynk app. When a switch is turned ON from Blynk, it responds as expected, allowing the loads to be controlled and monitored directly through the app.

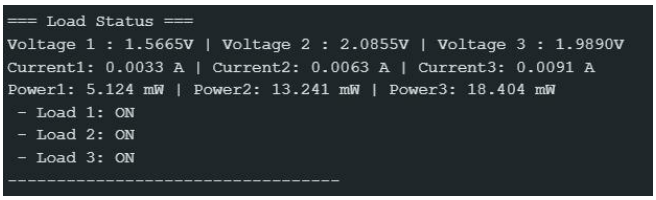

FIG 8. SERIAL MONITOR OUTPUT.

The output from Blynk performed well, with the USB cable serving as the main power source. However, due to wiring and power issues, the relay did not switch reliably between the solar and battery inputs. Additional work is needed to improve solar power and charging monitoring.

REFERENCES

[1] Muhammad Abdullah, Hammad Muhammad, and Muhammad Sarmad, "SMART PRIORITY BASED POWER LOAD SWITCHING USING PROGRAMABLE LOGIC CONTROLLER AUTOMATION," ResearchGate, 2022. [Online]. Available: https://www.researchgate.net/publication/366291235_SMART_PRIORITY_BASED_POWER_LOAD_SWITCHING_USING_PROGRAMABLE_LOGIC_CONTROLLER_AUTOMATION.

[2] Thirupathaiah M, "Design and Implementation of Solar Based DC Grid using Arduino UNO," ResearchGate, 2021. [Online]. Available: https://www.researchgate.net/publication/351323207_Design_and_Implementation_of_Solar_Based_Dc_Grid_using_Arduino_Uno.

[3] Abhishek Muthyala, "Solar Powered Battery Charging System by Using Arduino: Experimental design," ResearchGate, 2023. [Online]. Available: https://www.researchgate.net/publication/379551479_Solar_Powered_Battery_Charging_System_by_Using_Arduino_Experimental_design#:~:text=The%20implementation%20of%20this%20system,connected%20to%20an%20output%20pin.

[4] Erik Wahyu Pratama, "Electrical Analysis Using ESP-32 Module In Realtime," ResearchGate, 2022. [Online]. Available: https://www.researchgate.net/publication/367134207_Electrical_Analysis_Using_ESP-32_Module_In_Realtime.

[5] Espressif Systems, "ESP32-WROOM-32 Datasheet," Espressif Systems, 2025. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf.

APPENDIX

A. *Code in Arduino IDE for ESP32*

```
#define BLYNK_TEMPLATE_ID "TMPL62rI9RA98"
#define BLYNK_TEMPLATE_NAME "ESP32 Load Manager"
#define BLYNK_AUTH_TOKEN "_OLvWf_e3fLhp4OcDdVfvmkWFt8qhSSg"

#include <WiFi.h>
#include <BlynkSimpleEsp32.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SH110X.h>

// === WiFi Credentials ===
char ssid[] = "Your Wifi";
char pass[] = "Your Wifi Password";

// === Pin Definitions ===
const int SHUNT1_PIN = 34;
const int SHUNT2_PIN = 35;
const int SHUNT3_PIN = 36;

const int LOAD1_PIN = 14;   // Load 1
const int LOAD2_PIN = 12;   // Load 2
const int LOAD3_PIN = 13;   // Load 3

const int RELAY_PIN = 33;   // Solar/Battery relay
const int BATTERY_VOLTAGE_PIN = 32; // Not used now

// === Constants ===
const float R1 = 470.0;
const float R2 = 330.0;
const float R3 = 220.0;

// OLED initialization
Adafruit_SH1106G display(128, 64, &Wire, -1);

// === Blynk Button Handlers (with instant LED sync) ===
BLYNK_WRITE(V10) {
  int state = param.asInt();
  digitalWrite(LOAD1_PIN, state);
  Blynk.virtualWrite(V6, state);  // Update Load 1 LED immediately
}
BLYNK_WRITE(V11) {
  int state = param.asInt();
  digitalWrite(LOAD2_PIN, state);
  Blynk.virtualWrite(V7, state);  // Update Load 2 LED immediately
}
BLYNK_WRITE(V12) {
  int state = param.asInt();
  digitalWrite(LOAD3_PIN, state);
  Blynk.virtualWrite(V8, state);  // Update Load 3 LED immediately
}

// === Helper Functions ===
float readCurrent(int pin, float shuntResistor) {
  float raw = analogRead(pin);
  float voltage = (raw / 4095.0) * 5.0;
  return voltage / shuntResistor;
}

  float read_voltage(int analogPin) {
  int raw = analogRead(analogPin);
  float voltage = (raw / 4095.0) * 5.0;
  return voltage;
}

float power_value(float cur, float res) {
  return cur * cur * res * 1000.00; // mW
}

void setup() {
  Serial.begin(115200);
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);

  if(!display.begin(0x3C)) {
    Serial.println(F("SH1106 allocation failed"));
    for(;;);
  }

  display.clearDisplay();
  display.display();

  display.setTextSize(1);
  display.setTextColor(SH110X_WHITE);
  display.setCursor(5, 5);
  display.println(F("Smart BackUP Power \nManagement and \nMonitoring \nSystem"));
  display.display();

  pinMode(LOAD1_PIN, OUTPUT);
  pinMode(LOAD2_PIN, OUTPUT);
  pinMode(LOAD3_PIN, OUTPUT);
  pinMode(RELAY_PIN, OUTPUT);

  // Start all loads OFF
  digitalWrite(LOAD1_PIN, LOW);
  digitalWrite(LOAD2_PIN, LOW);
  digitalWrite(LOAD3_PIN, LOW);
  digitalWrite(RELAY_PIN, LOW);
}
```

```
void loop() {
  Blynk.run();

  //Read Voltage
  float v1 = read_voltage(SHUNT1_PIN);
  float v2 = read_voltage(SHUNT2_PIN);
  float v3 = read_voltage(SHUNT3_PIN);

  // Read currents
  float current1 = readCurrent(SHUNT1_PIN, R1);
  float current2 = readCurrent(SHUNT2_PIN, R2);
  float current3 = readCurrent(SHUNT3_PIN, R3);

  // Calculate power
  float pow1 = power_value(current1, R1);
  float pow2 = power_value(current2, R2);
  float pow3 = power_value(current3, R3);

  // === OLED Display ===
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SH110X_WHITE);

  display.setCursor(0, 0);
  display.printf("Power-1: %.3f mW", pow1);

  display.setCursor(0, 10);
  display.printf("Power-2: %.3f mW", pow2);

  display.setCursor(0, 20);
  display.printf("Power-3: %.3f mW", pow3);

  display.setCursor(0,30);
  display.printf("Voltage 1 : %0.3f\n", v1);
  display.setCursor(0,40);
  display.printf("Voltage 2 : %0.3f\n", v2);
  display.setCursor(0,50);
  display.printf("Voltage 3 : %0.3f\n", v3);


  display.display();

  // === Serial Debug ===
  Serial.println("=== Load Status ===");
  Serial.printf("Voltage 1 : %.4fV | Voltage 2 : %.4fV | Voltage 3 : %.4fV\n", v1, v2, v3);
  Serial.printf("Current1: %.4f A | Current2: %.4f A | Current3: %.4f A\n", current1, current2, current3);
  Serial.printf("Power1: %.3f mW | Power2: %.3f mW | Power3: %.3f mW\n", pow1, pow2, pow3);
  Serial.print(" - Load 1: "); Serial.println(digitalRead(LOAD1_PIN) ? "ON" : "OFF");
  Serial.print(" - Load 2: "); Serial.println(digitalRead(LOAD2_PIN) ? "ON" : "OFF");
  Serial.print(" - Load 3: "); Serial.println(digitalRead(LOAD3_PIN) ? "ON" : "OFF");
  Serial.println("-----------------------------------");

  // === Send data to Blynk ===
  Blynk.virtualWrite(V3, current1);
  Blynk.virtualWrite(V4, current2);
  Blynk.virtualWrite(V5, current3);

  // Keep LEDs synced
  Blynk.virtualWrite(V6, digitalRead(LOAD1_PIN));
  Blynk.virtualWrite(V7, digitalRead(LOAD2_PIN));
  Blynk.virtualWrite(V8, digitalRead(LOAD3_PIN));

  delay(2000);
}.
```

B. ESP32 Pinout Diagram

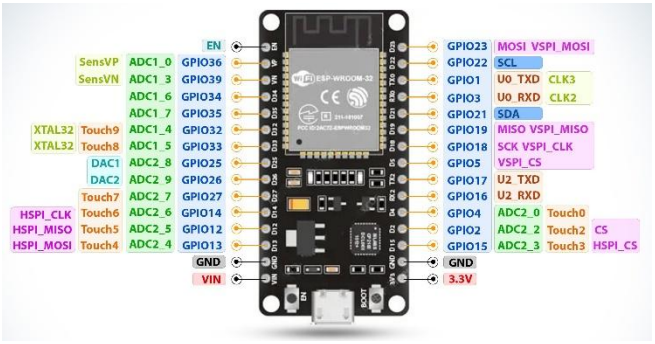

Figure 9. ESP32 Pinout Diagram