ICLR 15

MICROPROCESSOR AND MICROCONTROLLERS LABORATORY

PROJECT REPORT

DEEP SLEEP MODE USING ESP32

PREPARED BY

Rohit Jalani (110121081)
Kevinsam Tiny (110121045)
Priyanshu (110121067)
Pavan Kumar Ravi (110121061)

GUIDED BY

Dr. V. Sridevi
Assistant Professor

Department of
Instrumentation and
Control Engineering

ABSTRACT

This project aims to demonstrate the ESP 32 in deep sleep mode. Deep sleep mode is crucial for conserving power in various applications, especially in scenarios where energy efficiency is required, like battery-powered or eco-friendly IoT devices. We have discussed the operations of deep sleep mode and deep sleep mode with timer wake-up. In the deep sleep mode, we will explore the mechanisms involved along with their significance in daily life applications. We can extend its benefits in reducing power consumption as well as improving the battery life of various electronic appliances. In the deep sleep mode wake-up using the timer, we overlook the inbuilt functionalities of the timer for scheduled wake-up events. This mode gives us an idea to execute event schedulings and power-efficient operations. To demonstrate the above modes, we will be using a seven-segment display to denote the active mode opertion. By examining both the fundamentals of deep sleep and the practical implementation of timer-based wake-up, this project equips developers with the knowledge and tools necessary to design and operate energy-efficient applications in smart agriculture, home automation or wearable technology, ultimately extending their operational life and reducing the need for frequent battery replacements.

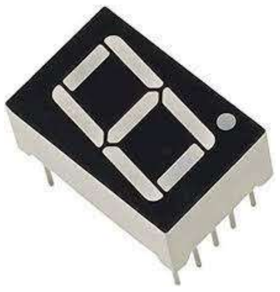
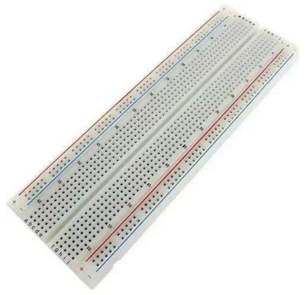
ACKNOWLEDGEMENTS

I wish to express my sincere thanks to my research guide, Dr. V. Sridevi for providing me with the expertise and knowledge for the implementation of the project work and for having faith in our aptitude through this entire period. I would also like to extend my gratitude to the Project Review Committee for ensuring a seamless progression of the project. Finally, I wish to thank the Head of Department, Dr. K. Dhanalakshmi for her support, without which the project would never see completion. We would also like to thank Lab Technician Sir for providing us with technical expertise.

PROBLEM STATEMENT

To implement deep sleep mode and wake up using timer in ESP32 development mode

COMPONENTS USED

	ESP32 Development Board- The ESP32 development board is a versatile and powerful microcontroller platform widely used in the field of electronics and IoT (Internet of Things) applications. It is designed to provide a comprehensive solution for various embedded projects, offering a range of features that make it suitable for both prototyping and production. At its core, the ESP32 development board is built around the ESP32 microcontroller chip, which is based on the Xtensa LX6 dual-core processor. This processor provides ample computing power, making it capable of handling complex tasks. Additionally, the board includes built-in Wi-Fi and Bluetooth connectivity, allowing for seamless communication with other devices and networks.
	**7-Segement Display** A 7-segment display is a common electronic component used for numeric and alphanumeric visual representation. It consists of seven individual LED segments arranged in a specific pattern, allowing it to display numbers from 0 to 9 and sometimes additional characters like letters or symbols. Each segment can be independently controlled, making it useful for showcasing numerical information in digital devices such as clocks, calculators, and digital meters.
	**Breadboard** A breadboard is a fundamental prototyping tool in electronics. It's a rectangular board with numerous holes for connecting electronic components without soldering. Components like resistors, LEDs, and wires can be easily plugged into the holes, allowing for quick and temporary circuit experimentation.
	Jumper wires are small, insulated wires used to make electrical connections between components on a breadboard or other prototype circuit.

TERMINOLOGIES

ESP32: The ESP32 is a low-cost, low-power system on a chip (SoC) series created by Espressif Systems. It boasts integrated Wi-Fi and dual-mode Bluetooth. The ESP32 is designed for mobile devices, wearable electronics, and Internet of Things (IoT) applications because of its rich features and reliability. It contains a Tensilica Xtensa LX6 microprocessor, which can be dual-core or single-core, and operates at up to 240 MHz. With its sleep and deep sleep modes, it's well-suited for battery-powered applications. It has a wide range of peripherals including capacitive touch, ADCs, DACs, I2C, SPI, UART, and much more. The chip also supports hardware encryption and image processing. Its development is supported by a free SDK, and it has gained massive popularity in the hobbyist market for its capabilities and extensive community support.

Deep Sleep: Deep sleep mode in ESP32 is a power-saving feature designed to minimize energy consumption during periods of inactivity. When the ESP32 enters deep sleep, it shuts down most of its components, including the CPU, peripherals, and RF (radio frequency) circuitry. This significantly reduces power consumption, making it suitable for battery-powered applications. During deep sleep, the ESP32 retains its state by storing relevant data in the RTC (Real-Time Clock) memory. This memory persists even when the main power is cut off. To wake the ESP32 from deep sleep, a trigger source is required. Common triggers include timers, external signals, or specific GPIO (General Purpose Input/Output) pins. The duration of deep sleep is typically configurable, allowing developers to balance power savings with the need for periodic wake-ups. The longer the deep sleep duration, the greater the power savings, but it also means a longer time to resume normal operation. Deep sleep is particularly advantageous in scenarios where devices spend a significant amount of time idle, periodically waking up to perform tasks and then returning to a low-power state, extending the overall battery life of ESP32-based applications.

Light Sleep: Deep sleep mode is designed for ultra-low-power consumption but comes with the trade-off of slower wake-up times. If faster wake-up times are a priority, you may consider using other low-power modes that are less power-efficient but offer quicker transitions to active states. One such alternative is the "Light Sleep" mode. In Light Sleep mode, only the CPU is paused while the rest of the system, including peripherals and memory, continues to operate. This allows for faster wake-up times compared to deep sleep. However, Light Sleep doesn't achieve the same level of power savings as deep sleep.

Maximum delay

The maximum delay in deep sleep mode on the ESP32 is determined by the timer used to wake up the chip. The ESP32's deep sleep mode is often configured with a sleep time defined by a 32-bit unsigned integer. The maximum delay that can be set in deep sleep mode is determined by this 32-bit value.
The formula for calculating the maximum delay (in microseconds) for the ESP32 deep sleep mode is as follows:
Maximum Delay = $2^{32}-1$
This is because the sleep time is represented by a 32-bit unsigned integer, and the maximum value that can be represented by a 32-bit unsigned integer is $2^{32}-1$. Therefore, the maximum delay we can set for deep sleep mode is $2^{32}-1$ microseconds, which is approximately 71.6 minutes.

Timer Module:

The ESP32 timer module is a built-in feature that provides precise timing and scheduling capabilities. It includes multiple hardware timers, each with specific functions. These timers enable tasks like generating accurate delays, measuring time intervals, and triggering events at specific times. They offer microsecond-level precision, support interrupts, and can be configured for various applications, making them a valuable tool for time-sensitive tasks in ESP32-based projects.

RTC

RTC Real-Time Clock (RTC) plays a crucial role. The RTC is a separate hardware component on the ESP32 chip that operates independently of the main processor, and it continues to run even when the rest of the chip is in deep sleep.

The basic relation between the RTC and ESP32 deep sleep mode:

- Wake-Up Source:
  The RTC serves as a wake-up source for the ESP32 during deep sleep. When the ESP32 enters deep sleep mode, it can be configured to wake up after a certain amount of time (sleep duration) using the RTC alarm. This allows the ESP32 to consume minimal power during the sleep period and then wake up when needed.

- RTC Alarm:
  The RTC alarm is used to set a specific time at which the ESP32 should wake up from deep sleep. This can be a precise time or a relative time from the moment the deep sleep mode is initiated. The ESP32 will enter deep sleep, and the RTC alarm will trigger a wake-up when the specified time elapses.

- RTC Memory:
  The ESP32 provides a small amount of RTC memory that can be used to store data that needs to persist across deep sleep cycles. This allows the ESP32 to retain certain information even when the main processor is in deep sleep. This RTC memory is separate from the main RAM.

- Low-Power Operation:
  While in deep sleep, most of the ESP32's components, including the main CPU and peripherals, are powered down to minimize power consumption. The RTC continues to operate with low power consumption, and it's responsible for managing the wake-up process.

- Power Management:
- The RTC also plays a role in managing power domains during deep sleep, helping to control which parts of the chip remain powered and which can be safely shut down to conserve energy.

Operating Frequency:

ESP32 we used operates at 240 MHz frequency in active mode. The following code was used to check the operating frequency:

```
#include "esp_system.h"
#include "esp_log.h"
#include <esp_clk.h>

void setup() {
  // Setup code goes here
  Serial.begin(115200); // Initialize the serial communication

  // Your additional setup code, if any
}

void loop() {
  // Main loop code goes here
  printf("CPU Frequency: %d Hz\n", xt_clock_freq());

  // Your additional loop code, if any
  delay(1000); // Add a delay to make it easier to read the output
}
```

7-Segment Display: A 7-segment display is a simple electronic component commonly used to display numerical digits and some alphanumeric characters. It consists of seven individual LED (Light Emitting Diode) segments arranged in a specific pattern, typically resembling the digit "8." These segments can be individually controlled to illuminate and form different numbers from 0 to 9, as well as a few letters like A, B, C, etc., with each segment representing a specific portion of the character. This display is popular due to its simplicity and readability, making it suitable for various applications such as digital clocks, calculators, digital meters, and basic numerical readouts. To display a specific number or character, we need to activate the corresponding segments while keeping others off.
7-segment displays come in two main types: common anode and common cathode. In common anode displays, the anodes of all segments are connected together, and individual cathodes control each segment. In common cathode displays, the cathodes are common, and the anodes control each segment. Choosing between the two depends on the circuit's design and compatibility with the microcontroller or driver used.

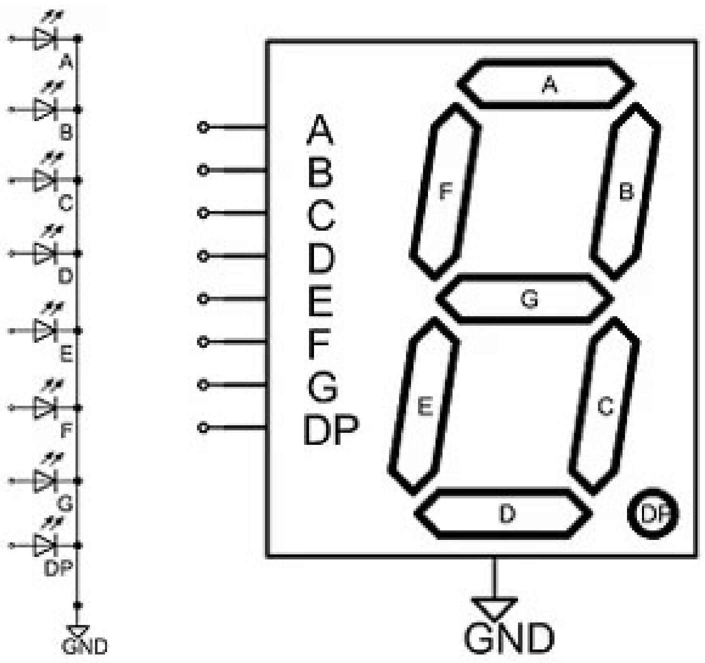


Fig: 7-Segment Display in Common anode configuration

PROPOSED IMPLEMENTATION

We used a 7-segment diaply as an indiactor. It remains turned off for 10 seconds of deep sleep mode of ESP32, and dispalys the counts from 0-9 for 10 seconds of active mode displaying that the ESP32 is in active mode for 10 seconds.

ASSOCIATED CODE

Following is the code used to control the ESP32 and the 7-segment display as described above:

```
#include "SevSeg.h" // Include the seven-segment library
SevSeg sevseg; // Create a seven-segment object
#define factor 1000000 // Define a constant factor for sleep time calculation
#define sleep_time 10 // Define the sleep time in seconds
RTC_DATA_ATTR int bootCount = 0; // Define a variable to store the boot count in RTC
memory

void setup() {
  Serial.begin(115200); // Initialize serial communication at a baud rate of 115200
  byte sevenSegments = 1; // Number of connected seven-segment displays (1 in this
case)
  byte CommonPins[] = {}; // Define the common pin(s) of the seven-segment display
(empty in this case)
```

```
  byte LEDsegmentPins[] = {15, 2, 4, 5, 18, 19, 21}; // Define ESP32 digital pins for the
seven-segment display
  bool resistorsOnSegments = true; // Assign a Boolean value to indicate whether there
are resistors on segments

  // Initialize the seven-segment display configuration
  sevseg.begin(COMMON_ANODE, sevenSegments, CommonPins, LEDsegmentPins,
resistorsOnSegments);

  sevseg.setBrightness(90); // Set the brightness of the seven-segment display to 90%

  if (bootCount == 0) {
    bootCount++; // Increment the boot count if it's the first boot
  }
  Serial.println(bootCount); // Print the boot count to the serial monitor
}

void loop() {
  for (int i = 0; i < 10; i++) {
    sevseg.setNumber(i); // Set the number to be displayed on the seven-segment
display
    sevseg.refreshDisplay(); // Refresh the seven-segment display to show the new
number
    delay(1000); // Delay for 1 second before displaying the next number
  }

  // Put the Arduino into deep sleep mode for 10 seconds
  esp_sleep_enable_timer_wakeup(sleep_time * factor);
  esp_deep_sleep_start();
}
```

This code is designed to facilitate the interaction between an ESP32 microcontroller and a seven-segment display. It begins by including the "SevSeg" library, which allows for easy control of the display. The program initializes various parameters, such as the number of connected seven-segment displays (in this case, one), the digital pins on the ESP32 responsible for controlling each segment of the display, and whether there are resistors on these segments.

One notable aspect of the code is the use of RTC memory to store a boot count variable. Upon each boot, this variable is checked, and if it's the first boot, the count is incremented. This demonstrates a simple form of non-volatile memory storage within the ESP32, useful for tracking boot events or other persistent data.

The code also sets the brightness of the seven-segment display to 90%, ensuring the information displayed is visible without being overly bright. Communication with the ESP32 occurs through the serial interface, allowing for easy monitoring and debugging of the program's behavior.

OBSERVATIONS

We have added a seven segment display to showcase a counter of 10 sec- displaying 0 to 9 sec. When the esp 32 is operating on active mode, the seven segment would display the digits 0 to 9. When the esp 32 is switched to deep sleep mode, the seven segment display wont showcase the digits.

CONCLUSIONS

The deep-sleep was successfully implemented in this project. This project effectively demonstrates the benefits of energy-efficient design, particularly through the successful implementation of deep sleep mode using the ESP32 development board. The use of a timer to wake up the system showcases its adaptability and efficiency, making it a suitable choice for devices that require power conservation. The addition of an active mode, featuring a 10-second display on a 7-segment display, enhances the project's practicality and user-friendliness. This not only highlights the versatility of the system but also strikes a balance between energy conservation and user engagement. The numerical counting on the 7-segment display provides a clear indication of the active state, improving accessibility and overall user experience.

In conclusion, the project underscores the importance of energy-efficient practices in electronic devices. The successful implementation of deep sleep mode and the thoughtful integration of an active mode reflect a commitment to sustainability and practical application. As technology progresses, the need for power conservation becomes more critical, and this project stands as a notable example of how straightforward yet impactful strategies can contribute to a more eco-friendly and resource-conscious future.

REFERENCES

ESP 32 documentation: https://docs.espressif.com/projects/es+idf/en/latest/esp32/
Seven segment display datasheet: https://www.farnell.com/datasheets/2095789.pdf